IMT Institute for Advanced Studies, Lucca

Lucca, Italy

A Voronoi Based Framework for the Definition of P2P Distributed Virtual Environments

PhD Program in Computer Science and Engineering

XXI Cycle

By

Luca Genovali

2009

The dissertation of Luca Genovali is approved.

Program Coordinator: Prof. Ugo Montanari, University of Pisa

Supervisor: Prof. Carlo Ghezzi, Politecnico di Milano

Supervisor: Prof. Fabrizio Baiardi, University of Pisa

Tutor: Prof. Laura Ricci, University of Pisa

The dissertation of Luca Genovali has been reviewed by:

Prof. Dana Petcu, University of Timisoara

Prof. Igor Kotenko, Russian Academy of Sciences

IMT Institute for Advanced Studies, Lucca

2009

Contents

List of Figures

Vita

July 16, 1973	Born, Pietrasanta, Italy
2005	Degree in Computer Science Final mark: 108/110 Pisa University, Italy

Publications

1. A.Bonotti, L.Genovali, L.Ricci, ”DIVES: A Distributed Support for Networked Virtual Environments”, *20th IEEE AINA*, Wien, April, 2006.
2. F.Baiardi, A. Bonotti, L.Genovali, L.Ricci, ”A publish subscribe support for networked multiplayer games”, *Internet and Multimedia Systems and Applications (EuroIMSA 2007)*, Chamonix, March, 2007.
3. F.Baiardi and L.Genovali and L.Ricci, ”Improving Responsiveness by Locality in Distribited Virtual Environments”, *21 ECMS*, Prague, June, 2007.
4. L.Genovali, L.Ricci, ”JaDE: A JXTA Support for Distributed Virtual Environments”, *13th IEEE Symposium on Computers and Communications Program*, Marrakesh, July, 2008.
5. L.Genovali, L.Ricci, ”Voronoi Models for Distributed Virtual Environments”, *ACM CoNEXT*, Student Workshop, Madrid, 2008.
6. L.Genovali, L.Ricci, ”AOI-Cast Strategies for P2P Massively Multiplayer Online Games”, *6th Annual IEEE Consumer Communications and Networking Conference IEEE CCNC*,Las Vegas, Nevada, January, 2009.
7. L.Genovali, A.Quartulli, L.Ricci ”A Voronoi Based Framework for the Development of P2P Distributed Virtual Environments”, Journal of Peer-to-Peer Networking and Applications, Springer, Verlag, Submitted for publication.

Abstract

The diffusion of wide area networks has lead to the definition of novel applications such as the *Distributed Virtual Environments (DVE)*, for instance massively multiplayer games, militar or civil distributed simulations. In a DVE a set of active entities (avatars), interact with each other and with a set of passive objects located in their surroundings. While most DVE are currently still managed according to the client server model, the P2P model has recently been investigated, even if the killer application for P2P has been till now the file sharing one. This thesis investigates the feasibility of a P2P architecture for *Distributed Virtual Environments*. Locality of DVE interactions, modeled through the notion of *Area of Interest, AOI* of each entity, is properly exploited by the P2P communication support to reduce the amount of messages exchanged through the P2P overlay. Furthermore, a mechanism to dynamically acquire knowledge of the state located beyond the AOI and a strategy to preserve consistency of the replicated state is defined. We propose to model a DVE by a *Voronoi Tessellation*, where the sites correspond to peers and the Delaunay triangulation links define the topology of the P2P overlay. Different solutions for the definition of the overlay network and a set of protocols for the propagation of the *heartbeats* and the management of the *passive objects* have been formally defined, implemented and evaluated with respect to consistency and scalability. Finally, this thesis investigates the definition of a *hybrid P2P support* for the development of a DVE, where a hierarchy of peers is defined according to their computational/communication power. The resulting overlay is modeled through a *Weighted Voronoi Diagram*.

Chapter 1

Introduction

Distributed Virtual Environments (DVE)(75) such as military or civil protection distributed simulations and massively multiplayer online games (MMOG), for instance *World of Warcraft*(72) or *Second Life*(76), are currently gaining increasingly attention in the software market.

In a DVE users located at *geographically distributed* hosts interact within *a virtual world* which is populated by user controlled *avatars* or by *computer controlled entities*. Each avatar moves within the DVE and may interact with other avatars and with the passive objects of the virtual world.

Currently the *client server* model is still the reference computational model for this kind of applications, but approaches based on the $P2P$ model (1; 57; 65; 71) have recently been proposed.

In the *client server architecture* a *single server* is responsible both of the notifications of the position updates of the avatars to the interested avatars and of the management of the state which is modified due to the clients interactions. According to this model, each client notifies any event to the central server which updates the state of the DVE and notifies the event to the interested clients. Furthermore the server computes a meaningful ordering of the events. The main disadvantage of this solution is due to the low level of robustness and of scalability due to the presence of the single server.

The definition of a fully distributed architecture for DVE is an actual challenge because of the complexity of these applications which integrate networks, graphics and AI programming. On the other hand, the adoption of a distributed computational model is mandatory to overcome the low scalability of client server architectures.

The $P2P$ computational model has recently received a considerable attention. Most P2P systems developed till now support *file sharing applications*. In this context, several P2P models have been proposed, each one corresponding to a different topology of the *overlay network* connecting the peers. These models range from the *unstructured ones*, to those based on the *Distributed Hash Table* approach (3; 6; 9; 12; 17) which defines a structured overlay in order to bound the number of routing hops required to retrieve an information in the network. Unstructured overlays include centralized architectures, like the pioneer proposal of *Napster* (12; 16), pure P2P architectures, like those of *Gnutella 0.4* (4; 5; 10; 12; 16) and hybrid architectures, like *Kazaa* and *Gnutella 0.6* (12).

On the other hand, only a few proposals of *P2P Distributed Virtual Environments* have been presented till now. The definition of a P2P architecture for this kind of applications introduces novel problems related to the interactive nature of these application, to their consistency requirements and to the presence of concurrent updates of the state of the passive objects of the virtual world. For instance, interactive applications like DVE, require that every event is executed in a particular instant over the time, therefore the concept of time is fundamental for this kind of applications.

In a DVE, a set of *active entities*, represented by *avatars*, interact with each other and with a set of *passive objects*, like weapons, potions, etc. Each avatar moves continuously within the DVE and in general it interacts with the avatars and the objects located in its surroundings only. This *locality property* which is modeled through the notion of *Area of Interest (AOI)* of an avatar, should be properly exploited by a communication support to reduce the amount of messages exchanged through the $P2P$ overlay. On the other hand, since the view of each avatar is constrained by its *AOI*, a mechanism to dynamically acquire knowledge of avatars and of passive objects located beyond the AOI is required.

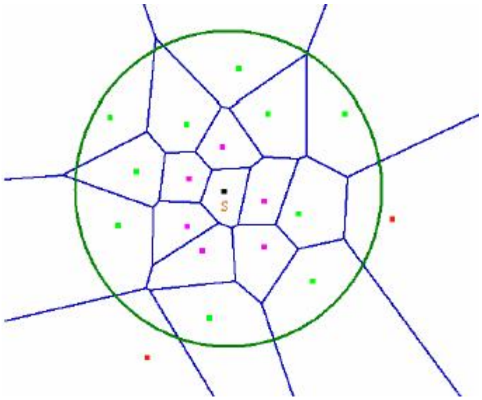


Figure 1: A Voronoi Tessellation

The management of the *state of the passive objects* of the DVE is a further critical issue for the development of $P2P$ supports because it requires to manage the *concurrent updates* that may occur when several avatars concurrently update the same object. The adoption of classical solutions like those based on logical timestamps is not suitable in this case, due to the high number of messages required to implement these mechanisms. Furthermore, since $P2P$ networks are highly dynamic, a set of mechanisms to guarantee *object persistence* should be defined as well.

This thesis investigates the feasibility of exploiting *Voronoi Tessellations* (56) to model DVE applications.

Given a set of sites $S = s_1 ... s_n$ in a $2D$ space, a *Voronoi tessellation* is a space partition that assigns to each site s_i the region $Voro(s_i)$, that includes the set of points which are closer to s_i than to any other site $s_j \in S$, $i \neq j$, according to a given metric. Standard *euclidean metric* is exploited in classical Voronoi tessellations. Two sites are *Voronoi Neighbors* iff the borders of their region overlaps. A *Delaunay triangulation* is a graph that connects the *Voronoi neighbors*. Fig. 1 shows a *2D Voronoi Tessellation*.

We propose to model a DVE by a *Voronoi Tessellation* where sites correspond to peers and the *Delaunay triangulation links* define the *topology* of the P2P overlay. Any passive object in $Voro(s_i)$ is assigned to the peer corresponding to s_i, which becomes the object *owner*, stores its state and manages concurrent updates.

The notion of locality is modeled through the concept of *area of interest*, i.e. a zone of the virtual world surrounding an entity **E** and including those entities that can interact with **E**.

In a DVE, any peer P periodically sends a *heartbeat*, i.e. a message notifying its position to any other peer in its *AOI*. We have exploited *compass routing*, (64) an algorithm that properly exploits the property of *Delaunay triangulations* to compute a multicast spanning tree covering the peers belonging to $AOI(P)$. A large number of routing hops may be required to reach any peer in $AOI(P)$ especially when a *crowding scenario* occurs, i.e. a huge amount of peers is located in $AOI(P)$. Note that crowding often occurs in a DVE, where peers may either be attracted by the same object, for instance a sword or a magic potion, or gather to fight each other. Hence a large delay may have a negative impact on the responsiveness of these applications. To reduce this delay, any peer may dynamically define *direct connections* with a subset of peers in its AOI. These connections are added to the Delaunay links and act as *'shortcuts'* by reducing the notifications delay.

The most challenging issue of our approach is the definition of proper *distributed algorithms* to guarantee that the structure of the Delaunay overlay is correctly preserved and no overlay partition occurs. We propose a *'pass the word'* approach, where peers becomes acquainted with each other through their Voronoi neighbors. A peer receiving an heartbeat H from one of its neighbors N checks if any of its further neighbors Q, $Q \neq N$, is entering $AOI(N)$, and in this case it propagates H to Q. In this way, each peer acts as a *beacon* for its neighbors by 'putting in touch' peers that are not acquainted with each other. A similar approach is adopted to acquire new objects located beyond the *AOI* of a peer.

It is worth noticing that several inconsistencies may rise due to network delays, messages loss, or abrupt peer crashes which often occurs in $P2P$ environments. A certain amount of *replication* is required to avoid irrecoverable situations due to these inconsistencies. For instance, it is

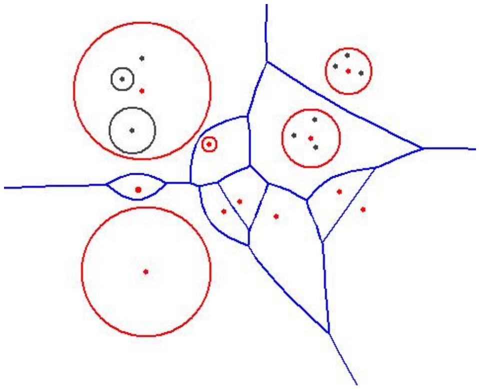


Figure 2: Additive Weighted Tessellations

necessary to guarantee that each peer is always connected to its Voronoi neighbors to avoid the partitioning of the overlay. This may be obtained, for instance, by sending redundant heartbeat messages generated by a peer P to the Voronoi neighbor of its neighbor, even when the latter do not belong to its AOI.

This thesis also investigates the feasibility of exploiting *Weighted Voronoi Tessellations* (56) to model *hierarchical P2P networks*. A Weighted Tessellation assigns a *weight* w_i to each site s_i, so that the size of $Voro(s_i)$ depends upon w_i. The metric exploited by *Additively Weighted Voronoi Tessellations, AWV,* defines the distance of a point from a site s_i as the sum of w_i and the euclidean distance between the point and s_i. Fig. 2 shows an *Additive Weighted Tessellation* where the weight of a peer P is proportional to the extension of the circle centered at P. Notice the presence of peers which have been 'absorbed' by peers with a larger weight located in their surroundings. No Voronoi region is associated with these peers which are hidden to the rest of the world and do not belong to the Delaunay Triangulation.

We exploit *AWV tessellations* to model a *hierarchical P2P Network*, where the weight assigned to each peer is proportional to its computational power. The adoption of *AWV Tessellations* to model hierarchical *P2P* overlays offers several advantages. First of all, larger Voronoi regions are assigned to peers characterized by larger weights. This defines a *load balancing strategy* for passive objects, because the number of passive objects assigned to each peer will be proportional to its weight. Since no area is assigned to hidden peers, they will not manage any object.

Furthermore, peers which have 'absorbed' other peers become *Superpeers* acting as *servers* toward them. An hidden peer P characterized by a low weight may rely on the corresponding *Superpeer* which acts as a *proxy* for P by forwarding any notification generated by P to its neighbors and by notifying to P events generated in its surroundings. Note that the same peer may play a different roles according to the DVE scenario. For instance, the same peer may act as a *Superpeer* if the number of neighbors is bounded by a threshold, while it requires the support of a *Superpeer* if this number exceeds this threshold.

The structure of the thesis is the following one.

Chapter 2 describes the most important P2P architectural models presented in the last years and the most relevant P2P Distributed Virtual Environments presented in the literature.

Chapter 3 presents the formalism exploited in the following chapters for the specification of the heartbeat propagation and passive objects management protocols.

Chapter 4 and Chapter 5 present the original solutions we propose. The mathematical structures upon which our solutions are based are presented in chapter 4, that introduces the compass-routing based spanning tree construction algorithm, proves some properties of the algorithm and finally evaluates the proposed approach through a set of simulations developed through the *Peersim simulator*.

Chapter 5 introduces our original proposal for the management of the passive objects of the DVE in a Voronoi based P2P overlay. A set of experimental results are presented as well.

Chapter 6 presents a hierarchical overlay for DVE exploiting *Additive Weighted Voronoi Diagrams* and shows a set of preliminary experimental results.

Finally, Chapter 7 presents our conclusions and the future works.

Chapter 2

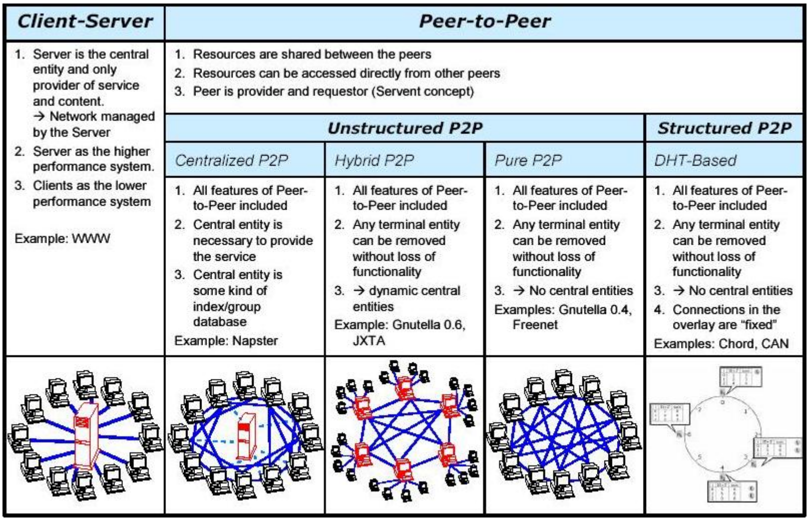
State of the Art

File sharing has been the "killer application" for P2P system till now. Recently further applications like *Content Distribution Networks, CDN*(73), *Computer Supported Collaborative Works, CSCW*(74) and *Distributed Virtual Environments, DVE*(75) have been proposed. Nevertheless, these applications generally exploit an architectural model originally proposed for file sharing. For this reason, this chapter will first briefly review the main P2P architectural model with reference to file sharing applications. We will present the evolution of P2P systems during the last ten years, from the centralized solution of *Napster* to the pure P2P solutions like that exploited by *Gnutella*. Hybrid solution introducing a hierarchy of peers on the basis of their computational/communication power, like *Gnutella 0.6* and *Kazaa* are also discussed. Finally, we briefly present solutions defining a *structured overlay*,like those based on the definition of a *Distributed Hash Table*.

The second part of the chapter describes the most important *P2P Distributed Virtual Environment* recently proposed. We will present both proposal which extend the architectural models originally proposed for file sharing, like *Mopar*,*SimMud* and *JaDE* and more innovative proposals based on the definition of *highly dynamic overlays* like *Solipsis*, *Apolo* and *VON*.

2.1 P2P Models

In this section we review the main P2P architectural models proposed in the last years which are summarized in Figure 3. We will present the evolution of P2P system in the last ten years, first describing the unstructured P2P overlays, then the distributed hash table structured overlays.



Client-Server	***Peer-to-Peer***			
1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system. 3. Clients as the lower performance system Example: WWW	1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept)			
	Unstructured P2P			***Structured P2P***
	Centralized P2P	*Hybrid P2P*	*Pure P2P*	*DHT-Based*
	1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Example: Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Example: Gnutella 0.6, JXTA	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities Examples: Gnutella 0.4, Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are "fixed" Examples: Chord, CAN

Figure 3: P2P Models.

2.1.1 Centralized Unstructured P2P

Napster (13; 17) is a centralized file sharing application (Fig.3) targeted to music files that belongs to the first generation of P2P systems. Each peer establishes a connection with a central server S, and sends to S the descriptors of all the files it is going to share. The central server stores the descriptors in a centralized database. Dynamic connections are defined between the peers, and files are exchanged directly between them, on demand.

If a peer P looks for a file, it sends to the central server a query including a list of keywords describing it. The server searches its database for the file descriptors matching the query, and sends to P a set of pairs (F', Q), where Q is a reference to a peer sharing a file F' matching the query. P chooses a pair according to some criterion and establishes a connection with the corresponding peer.

Even if the file searching step exploits the Client Server computational pattern, the file exchange step occurs directly between pair of peers, according to the P2P model.

The main drawback of this approach is the bottleneck and the single point of failure introduced by the presence of a centralized server.

2.1.2 Pure Unstructured P2P Overlays

Gnutella 0.4(13; 17) and *Freenet*(6; 13) are both examples of pure P2P systems (Fig.3) characterized by the lack of any centralized element in the P2P overlay. Connections between peers are, in general, defined at random. In *Gnutella*, peers exchange *PING messages*, at regular time intervals, to discover new peers in order to increase the knowledge of the network and *QUERY messages* to search files described by a list of keywords. The mechanism for the transmission of these messages is the *TTL enhanced flooding*, i.e. each message is paired with an integer value, the TTL, which is exploited to limit the number of its hops.

The flooding mechanism is the most important drawback of these approaches. As a matter of fact, a large amount of traffic is generated, even if the TTL is exploited. Furthermore, since in general the overlay network does not match the underlying physical network, each message can be routed through a *zig-zag path* on the underlying physical network.

These issues reduce the scalability of pure P2P unstructured systems and may introduce an high delay in the delivery of a message.

2.1.3 Hybrid Unstructured P2P Overlays

JXTA, Gnutella 0.6, Kazaa (13) are examples of a *hybrid P2P systems* belonging to the second generation of P2P(Fig.3). These systems exploit the heterogeneity of the peers to introduce a peer hierarchy defined on the basis of their computational/communication capability. In most cases, this is a *two-level hierarchy* including *Simple Peers* and *Super Peers*. The *Super Peers* generally carry out a richer set of functionalities with respect to the simple ones. For instance, in *Gnutella 0.6* or in *Kazaa*, the resource shared by the peers are indexed only by the Superpeers and the traffic due to the trasmission of the queries is limited to the overlay connecting the Superpeers.

The term *JXTA* stands for *juxtapose*, i.e. it reflects the intention to put P2P system side by side with respect to client-server or Web-based computing. *JXTA* is an open network computing platform designed for P2P computing. Its goal is to develop the basic building blocks and services to support innovative applications for peer groups. *JXTA* conceptually is a set of *open P2P protocols* that allow any connected device on the network (from cell phone to PDA, from PC to server) to communicate and collaborate as peers. *JXTA* defines an *hybrid architecture* that introduces different types of peers i.e. *Minimal edge*, *Full-featured edge*, *Relay* and *Rendezvous* peers and introduces the concept of *Peergroup* which represents a set of peers with a common set of interests, for instance peers belonging to a chat room. Peer belonging to a Peergroup may *publish* the resources they are going to share through *advertisements*. The *Rendez vous peers* act as *Superpeers* by managing a set of *Edge peers* and indexing the advertisement published by them.

2.1.4 Structured P2P Overlays

Both the solution based on a central server and the one based on enhanced flooding are characterized by a low degree of scalability due, in the former case to the bottleneck represented by the central server, in the latter case to the breadth-first search on the unstructured overlay. While hybrid solutions reduce these problems through the introduction of a set of superpeers, the *Distributed Hash Tables, DHT* approach (7; 10; 18) defines a *structured P2P ovelay* so that the no central host is required and the cost of retriving an item in the overlay is reduced.

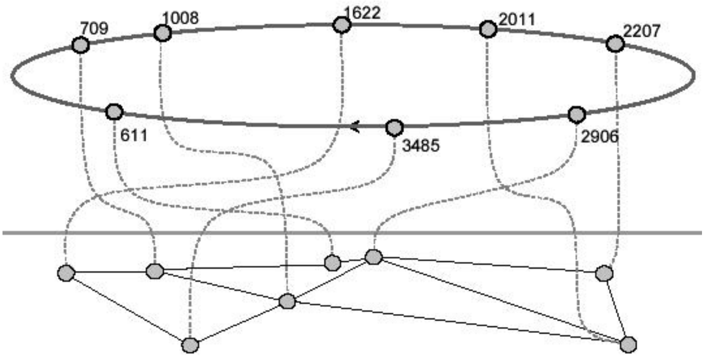


Figure 4: Distributed Hashed Table.

The basic idea of the *DHT* approach is to assign a logical identifier through an *hash function* to both the nodes and the data items and to map both of them to a *common logical address space*. A proper mapping function is defined so that each data item is mapped to a single node.

Figure 4 shows a logical ring overlay over the real network topology. The nodes of the ring are ordered according to a clockwise ordering and each node manages the linear portion of the logical address space ranging from the identifier which is the successor of its predecessor on the ring, and its identifier.

When a data item is stored in the system, a unique ID is assigned to it. Then, the data item itself or a reference to it is stored at the node that manages the subset of identifiers including ID.

The key concept of the DHT approach is to define a *routing algorithm* able to reach the node storing a data idem through a limited number of steps, for instance in most DHT the number of steps is *logarithmic* with respect to the number of nodes of the DHT. For this reason each node stores a *routing table* including a proper subset of nodes of the DHT. This subset of nodes is chosen so that the routing respects the defined bound on the number of routing steps.

Suppose, for instance, that a node submits a query for a data item

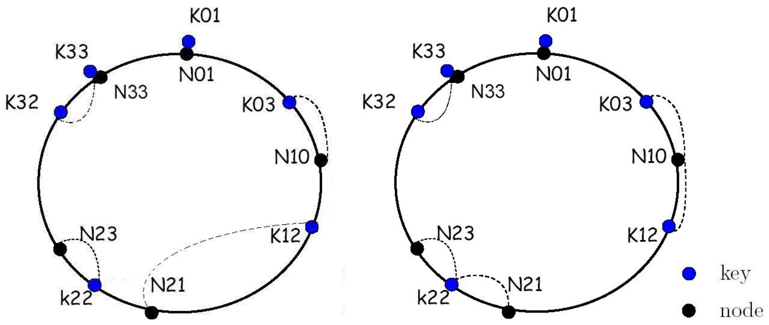


Figure 5: Chord and Pastry.

identified by ID. A node that receives the query selects among its neighbors, i.e. the nodes included in its routing table, the one whose identifier is *numerically closest* to the ID.

These basic ideas have been implemented in several different DHT which differ in the organization of the identifier space:

- **Chord:** In *Chord*(4; 7; 10; 13) the nodes are organized according to a circular ring where nodes are ordered clockwise. Each data item, characterized by a key **K** generated to a *SHA-1 hash function*, is stored at the node whose ID is the first one on the logical ring which is greater than or equal to **K** . For instance in Figure 5(left side), the key $K01$ belongs to node $N01$, $K03$ belongs to $N10$, $K12$ belongs to $N21$, $K22$ belong to $N23$, and both $K32$ and $K33$ belongs to $N33$.

- **Pastry:** in the *Pastry* DHT (13; 18) a key K is stored to the node N whose ID is numerically closest to K, as shown in figure 5(right side), where, $K01$ belongs to $N01$, both $K03$ and $K12$ belongs to node $N10$, $K22$ belongs to $N21$ and $N23$ because both of them satisfy the requirements, and both $K32$ and $K33$ belong to $N33$.

- **CAN:** In *CAN*, *Content Addressable Network* (13), the identifier space is a *d-dimensional space*. The space is partitioned into a *set of zones*,

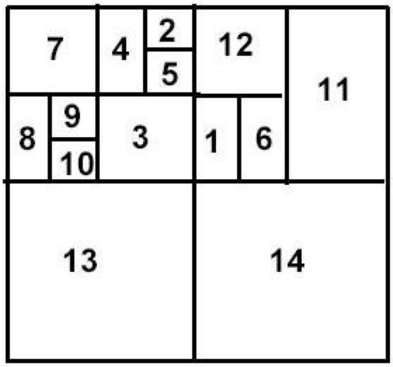


Figure 6: CAN.

each peer is paired with a zone and manages any key belonging to it. The routing table of a node N stores the neighbors of N, i.e. the nodes paired with zones whose border overlaps that of the zone managed by N. In figure 6, we assume, for the sake of simplicity, a two dimensional space. When a new item is stored in the CAN DHT, the hash function is exploited to pair a binary key with the item, then the binary sequence is divided into two parts which are the coordinates that define the zone Z where the item is mapped. The key is then stored at the node managing Z. CAN exploits a *greedy routing strategy*, i.e. when a node receives a key K, it propagates K to the node which is nearer to the zone where K is mapped.

2.2 P2P Overlays for DVE

This section presents the most important proposals of *P2P Distributed Virtual Environment presented* in the literature. First we will introduce two proposals which exploit the *Pastry DHT* for the development of a

P2P DVE overlay. In the first proposal, *MOPAR*, the DHT is exploited to define a hierarchical network, in the second one *SimMud* for the management of the passive objects of the DVE. The following proposals, i.e. *Solipsis*, *APOLO* and *VON* are based on the definition of a *highly dynamic overlay*. Finally we present two approaches *DiVES* and *JADE* which introduce several ideas developed in this thesis.

2.2.1 MOPAR: Mobile Overlay P2P Architecture

MOPAR, Mobile Overaly P2P Architecture(23) is a fully distributed P2P infrastructure supporting DVE applications which defines an overlay network using both a *Pastry based DHT* and an *hybrid P2P architecture*. *Mopar* decomposes the virtual environment into *hexagonal cells*, which, despite their discrete nature, assure a continuous view to all the participants.

MOPAR introduces three types of nodes based on their roles in the interest management scheme: *master*, *slave* and *home* nodes.

The original idea of *MOPAR* is to define a hierarchical overlay for DVE and is motivated by the need of reducing the number of messages exchanged among the peers for the management of their *Area of Interest*. *MOPAR* introduces a *Master Node* for each cell of the DVE. Each cell has at most one master node, but can have multiple slave nodes. The *Master Node* of a cell acts as a "beacon node" for the slave nodes by notifying them the new peers entering their *Area of Interest*. Slave nodes query the Master Node to build direct connections with their neighboring slave nodes, therefore, they do not need the master nodes involvement for notifying the accurate positions to their neighbors. In the same way, they also know when a participant is leaving their AOI. Each *Master Node* is connected to the *Master Nodes* of its 6 neighboring cells. In this way each *Master Node* covers an enlarged area and is able to notify its slave nodes about new peers entering their *Area of Interest*.

MOPAR pairs with each cell a *Home Node* which is a *Virtual Node* which acts as a registration point for master nodes and stores the state of the passive objects of that cell. Each hexagonal cell in *MOPAR* is characterized by its coordinates which are the unique *Cell ID* of the cell. *MOPAR* hashes the *Cell ID* to a 128-bit ID by applying SHAI-1 hash function getting the *HexID* of the cell, afterward it exploits the *Pastry DHT* to obtain the node N whose Pastry identifier is *numerically closest to the HexID*. Node N is now identified as the *home node* for this cell. Note that while each cell has a single *home node*, a node may be *home node* for a set of cells.

If a newly joined node finds that there is no master node registering for this cell after querying the home node, it registers itself as the master node; otherwise, it becomes a slave node. Master nodes build direct connections with the neighboring master nodes by querying the home nodes of the neighboring cells. The *home node* of a cell may be dynamically modified during the life-time of the DVE, and this step is supported by the underlying DHT.

2.2.2 SimMud

SimMud (48), is a support for *Massively Multiplayer Games* built on top of *Pastry* 2.1.4, a widely used P2P DHT, and use *Scribe*, the multicast infrastructure built on top of Pastry, to disseminate *game state*. The proposal exploits locality of interest typical of these applications.

Pastry maps both the participating nodes and the application objects onto random, uniformly distributed IDs in a circular 128-bit name space, and implements a distributed hash table to support object insertion and lookup. Objects are mapped onto live nodes whose ID is numerically closest to the object ID. "Closeness" in this context is limited to the numerical ID, no geographical or topological closeness is considered.

Scribe, instead, is an applicative scalable multicast infrastructure built on top of Pastry. Multicast groups are mapped to the same 128-bit ring of identifiers. A multicast tree associated with the group is built by merging the Pastry routes from each group member to the group ID's root, which also acts as the root of the multicast tree. Messages multicast from the root to the members uses reverse path forwarding.

The design of *SimMud* is based upon the limited movement speed and sensing capabilities of the avatars of the DVE. The world is statically *partitioned into regions* and the nodes in the same region form an *interest group* for that portion of the map, so that state updates relevant to that part are disseminated within the group only. A node changes group when it moves from one region to another. Additionally, objects in a given region have to communicate only the part of their state that is visible to nodes.

A live node whose ID is the closest to the region ID serves as the *coordinator* for that region. Besides coordinating all the shared objects of the region, the coordinator also acts as the root of the multicast tree, as well as the distribution server of the region map. The load can be distributed by creating a different ID for each type of object in the region, thus mapping them on to different peers.

SimMud defines different classes of game state and pairs different consistency maintenance strategies with each class.

- The *Player state* is accessed according to a *single-writer multiple-reader* pattern. Each player updates its own location as it moves around. Player-player interactions, such as fighting and trading, only affect the states, e.g., life points, of the players involved. Since position change is the most common event in a game, the position of each player is *multicasted* at a fixed interval to all other players in the same region. The interval is determined during game design, according to the requirements of the game.

- The *Object State* is managed by a coordinator-based mechanism to keep shared objects consistent while introducing a certain *degree of replication*. Each object, characterized by a key **K**, is assigned to a *coordinator*, managing any update of that object. The coordinator is the node **N** whose ID is the closest to the key **K**. Similarly, the next node **M** whose ID is closest to **N** ID's, will be the owner of the object's replica. The coordinator both resolves conflicting updates, and stores the current object value. Successful updates are multicasted to the region to update each player's local copy.

- *The map* is a non-graphical, abstract description of the terrain of a region. Graphic elements for the terrain and players are typically installed as part of the game client software, and can be updated through standard software mechanisms.

SimMud uses shared state replication to manage the crash of some peers. The copies are kept consistent in spite of node and network failures through a lightweight primary-backup mechanism which tolerates fail-stop failures of the network and nodes. These failures are detected using regular game events, without any additional network traffic.

2.2.3 Solipsis

Solipsis, introduced in (2; 3), is a *massively shared virtual reality system* based on a network of peers. It is one of the first system that does not rely on any server nor on IP multicast, and its goal is to scale to an unbounded number of participants and to be accessible by any computer connected to the Internet. Each peer implements a subset of the entities of the virtual world where each entity is identified by an unique id and is implemented by a peer. The only elements of the Solipsis world are the entities, which may be *virtual objects* and *avatars* where the only difference between these entities is that an avatar is associated with a user. Each avatar "perceives" its surroundings by a set of interactions with *adjacent avatars* that are the avatars located in its surroundings.

The global properties of a *Solipsis* network should match with the DVE application features. In particular, Solipsis should preserve *consistency* and support the ability of an avatar to move all over the DVE. This is guaranteed if each entity respects two local properties:

- **Local Awareness:** Each entity perceives only a part of the virtual world, or *Awareness Area*, inhabited by some entities and it should be aware of all updates to the entities located in this area. In other proposals, this area is referred as the *Area of Interest* of the entity and is generally defined by a disk centered at the entity. The *Local Awareness* property is satisfied if the peer P paired with the entity E is connected to any peer P' paired with an entity E' belonging to *Awareness Area* of E. If this property is satisfied, when an entity E

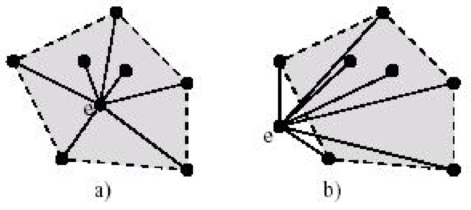


Figure 7: The convex hull of the adjacent of **e**. In situation **a**, **e** respects the Global Connectivity property while it does not in **b**.

updates its virtual representation, the corresponding peer should inform the peers managing entities in the *Awareness Area* of E only.

- **Global Connectivity:** In order to ensure the *Local Awareness* property, an entity should only rely on its adjacent entities. If it does not know any entity in a sector of the DVE, it is not able to become aware of an entity arriving from this sector. Conversely, if it moves towards a sector with no known entity, it will hardly get aware of entities located in this sector. Based on a set of computational geometry notions, the Global Connectivity property guarantees that an entity will not "turns its back" to a portion of the world. The global connectivity property is verified for an entity **e** when the position of **e** belongs to the convex hull defined by the set of entities adjacent to **e**, as shown in Fig.7. If this property is satisfied, the risk of an overlay disconnection is reduced and the isolation of a set of entities of the DVE may be avoided.

The graph defining the connections between the peers describes the *P2P overlay network* of the DVE. Note that the overlay is *highly dynamic*, because the connections between the peers change as they move. Each peer P exchanges data such as video, audio, avatars movements or any kind of events affecting the representation of the virtual world through the overlay links.

Let us now describe the procedures proposed in (2; 3) to guarantee that both the *Local Awareness* and the *Global Connectivity* properties are satisfied.

In order to satisfy the *Local Awareness property*, an entity should know all the entities in its surroundings. Because of the mobility of the peers, some other entities may dynamically enter into its *Awareness Area*. *Solipsis* proposes a *spontaneous collaboration* scheme where each entity sends a message when it detects an entity entering into the *Awareness Area* of another entity.

Let us consider an entity e and suppose that the peer managing e is connected two a pair of peers managing respectively, e' and e'' and suppose that e' enters the *Awareness Area* of e''. In this case e must "put in touch" e' and e''. The following events may force e to send a message to e':

- e' moves closer to e'';
- e'' moves closer to e';
- e moves away from e'';
- e'' Awareness Area grows up;
- e' does not belong to set of entities adjacent e at time Δt, while it belongs to this set at time t.

To guarantee the *Global Connectivity Property* an entity e must order its adjacent entities according to a *counter clockwise* ordering and detect for each adjacent entity e' such that the successor of e' in the ordering is e'', if the counterclockwise angle defined by (e', e, e'') is smaller than π. When an entity e detects two consecutive adjacent entities e' and e'' violating this property it enters a *Global Connectivity* restoring phase. In this phase, e sends a *connectivity restoring* message to e' which in turn searches among its adjacent entities, one entity e''' such that (e', e, e''') is smaller than π and (e''', e, e'') is smaller than π. If such an entity e''' does exist, then e''' is notified to e which inserts it among its adjacent nodes,

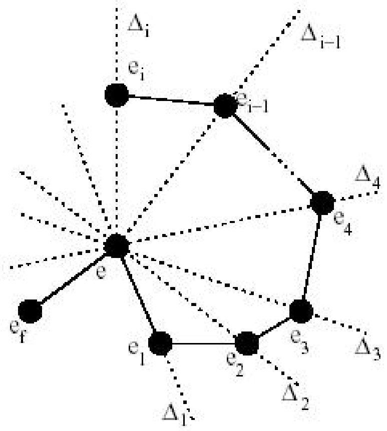


Figure 8: Restoring the Global Connectivity of an entity *e*.

otherwise a new *connectivity restoring* message is sent to the successor of e' which, in turn, checks among its adjacent nodes one which may restore the *global connectivity* property at e. This process is shown in Figure 8. e detects that the counterclockwise angle defined (e_1, e, e_f) is greater than π and sends a *connectivity restoring*message to e_1. If e_1 respects the global connectivity property, it is connected to an entity e_2 lying in the half plane determined by Δ_1. If $(e_2, e, e_f) < \pi$, then e inserts e_2 among its adjacent entities, otherwise the procedure is recursively applied. The algorithm ends when *e* finds an entity that forms with e_f an angle $< \pi$.

The *Solipsis* virtual world is able to handle thousand of entities: both participants and static objects. This is due both to the lack of a server and to the underlying distributed algorithms which rely on *local communications* only. In this way, the global amount of messages exchanged on the underlying overlay is proportional to the number of entities in the virtual world.

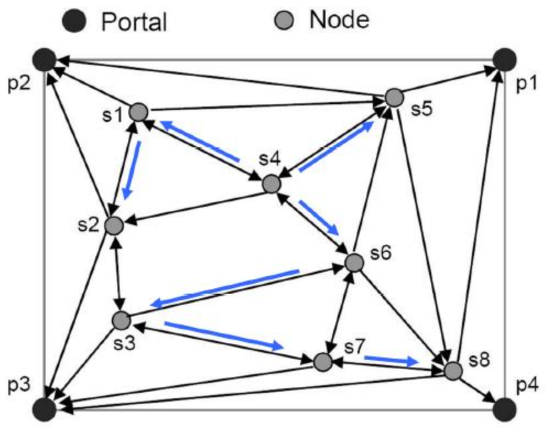


Figure 9: The APOLO Overlay.

2.2.4 APOLO: Ad-Hoc Peer-to-Peer Overlay Network for Massively Multiplayer Online Games

APOLO (54) proposes a protocol for the distributed definition of an overlay able to support efficient local group communication where the peers are able to self-organize. While each node creates a small number of links to the nearby nodes, the links are consistently re-organized only using soft-state local information.

In *APOLO*, the two-dimensional plane representing the DVE is partitioned into *quadrants* and each node maintains four links to the nearest neighbors in each quadrant, as Figure 9 shows. Note that *links are directed* because the existence of a link between the node n_1 and the node n_2 does not imply the existence of a link in the opposite direction.

To guarantee the connectivity of the overlay, four special-purpose virtual nodes, namely the *Portals* are defined. In Figure 9 these are the nodes P_1, P_2, P_3, P_4. The portals are located at the corners of the plane and are primarily used to guarantee the link property of *APOLO*, i.e. for each node a link to the nearest neighbor for each quadrant must be defined.

Otherwise, the nodes adjacent the borders of the virtual world may not connect to the nearest neighbor on each quadrant. The main objective of *APOLO* is to reduce the number of links which should be managed by each node.

APOLO develops a *2-hop beaconing protocol* to support *node mobility*. As a matter of fact, the overlay topology should be continually updated because of node mobility, unexpected node failures and join/leave of new nodes. Each node n periodically generates a *beacon message* which contains its up-to-date location information. The beacon message is recursively forwarded up to 2 hops away from n. In this way, each node knows the up-to-date locations of all its 2 hops far neighbors, and may identify the correct nearest neighbors and create direct links to them. In general, *m-hop beaconing*, $m \geq 2$ may be adopted to further reduce overlay inconsistencies.

APOLO considers different *message coverages* depending from message types. For instance, any event should be notified to nearby nodes, while only position updates are notified to distant nodes. This corresponds to define different *content based multicast mechanisms*. Multicast is based on the dynamic definition of a *spanning tree*. Each node n in *APOLO* is able to autonomously detect its parent in the multicast tree rooted at r by detecting its nearest neighbor which belongs to the same quadrant of r, with respect to the coordinate system whose origin is at n. For instance in Figure 10, node CN determines node $A1$ as its parent for a root node SN because $A1$ and SN locate in the same quadrant with respect to CN.

When a node n receives a multicast message, it forwards the message according the following rule. If the root node r and n are both positioned in the same quadrant, r sends the message to its neighbors belonging to the Area of Interest of r and such that the quadrant relations are satisfied.

Note that the unidirectional links make the message transmission inefficient. A node cannot forward the message to the nearby node even if there the two nodes are connected. As Figure 9 shows, each node is connected to other nodes through directed links. When node $s4$ multicasts a message, the message to node $s8$ passes through nodes $s6$, $s3$, $s7$.

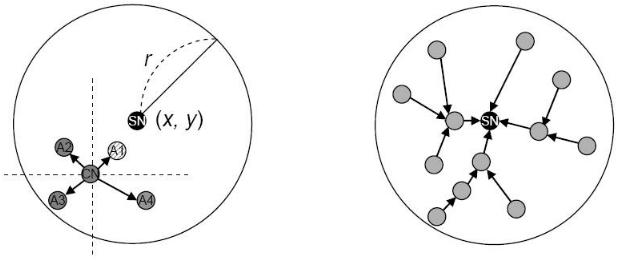


Figure 10: APOLO Multicast.

Although there is a connection between $s6$ and $s8$, the message cannot relay on this connection due to the unidirectional link.

2.2.5 VON: Voronoi Based Overlay Network

Voronoi based Overlay Network (VON)(1; 8; 9) is a P2P overlay network based on *Voronoi Diagrams* (56) which preserves high consistency of the overlay topology in a bandwidth-efficient manner.

A *Voronoi Diagram* (Fig.11) is a mathematical construction that, given **n** nodes on a plane, partitions the plane into **n** Voronoi regions, where a Voronoi region of a node **x** includes all the points of the plane which are nearer to **x** with respect to any other node.

VON exploits the concept of *Area of Interest, AOI,* to increase the *scalability* of the application. The *AOI* of a node n defines a circular or rectangular area surrounding n. Since only events generated within its AOI are relevant to n, the AOI reduces the number of messages exchanged by the application thus considerably increasing its *scalability*. An example is shown in Figure 11, where the green circle represent the AOI of the peer **S**, i.e. the area including peers which generate events S is interested in.

The initial proposal of *VON* defines a *direct connection model*, where any node of the DVE is directly connected to all the nodes located in its AOI. Due to the limited bandwidth of each node, this model may constrain the number of neighbors that may appear within the area of interest of a given node. *VON* defines different kind of neighbors of a

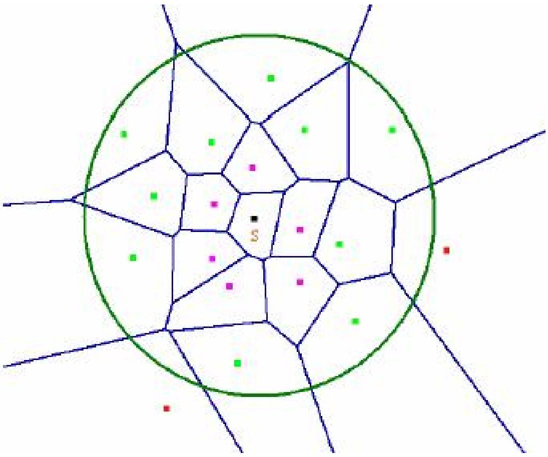


Figure 11: Voronoi Diagrams and AOI.

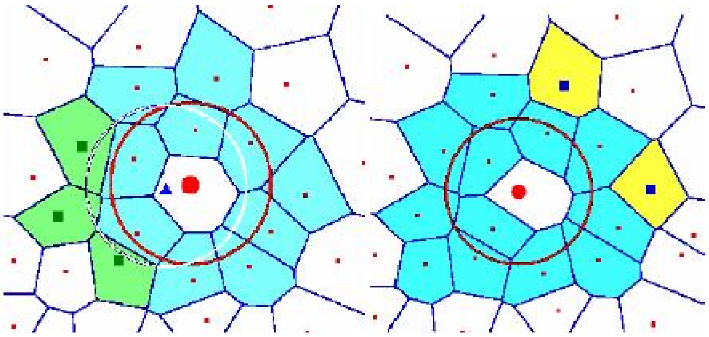


Figure 12: VON: Discovering New Neighbors

node. The *enclosing neighbors* of a node *n* are the nodes whose regions immediately surround the Voronoi region defined by *n*, the *boundary neighbors* are the nodes whose Voronoi region intersects the border of $AOI(n)$, while the *AOI neighbors* are all further nodes belonging to $AOI(n)$. In Figure 11 the pink nodes are the enclosing neighbors of S, the green nodes are boundary and AOI neighbors of S. Each node keeps a Voronoi Diagram including its enclosing, boundary and AOI neighbors.

In *VON*, each node acts as a *"watchman"* for another one in discovering approaching neighbors. When an entity moves (Fig.12), it sends its new position to all the neighbors belonging to its Voronoi Diagram. If the receiver is a boundary neighbor, it performs an *overlap-check*, i.e. it checks whether the Voronoi region of one of its enclosing neighbors overlaps the AOI of the mover. The receiver notifies the mover if a new overlap occurs, i.e. previously disjoint regions currently overlap. In this case the boundary neighbor *explicitly notifies* the mover about the new neighbors. This allows the moving entity to become aware of neighbors outside its AOI with minimal network overhead, since position notification may be exploited to discover new neighbors.

If a node leaves the DVE or fails, its neighbors update their Voronoi Diagram by removing that node.

This basic model of *VON* has been successively refined (8; 52; 53). As a matter of fact, the *direct connection* model may require a large amount of bandwidth, especially when *crowding* occurs. The most recent models reduce the number of connections by linking each node with its enclosing neighbors only. Event notifications are propagated to each AOI neighbor by *forwarding* notification through neighbor nodes.

VON Forwarding Models

The first forwarding model for *VON* has been introduced in (9). This model not only requires, for each node n, approximately a constant number of connections, but can exploit *aggregation* and *compression* to reduce the bandwidth consumption of n. Thus, this model enables a larger amount of nodes within the AOI of a node with respect to the direct connection model. However, as shown in Figure 13, this model introduces a certain

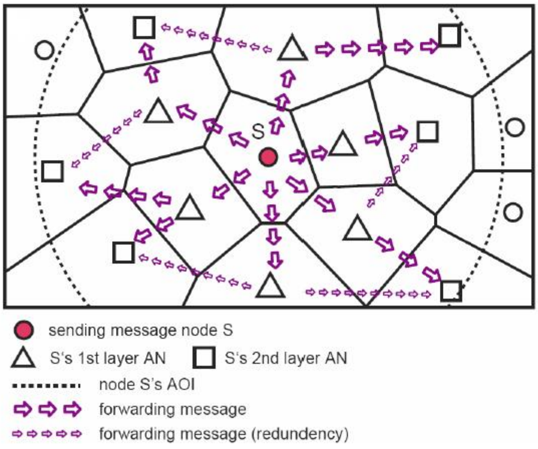


Figure 13: The forwarding path on VON Forwarding model.

amount of *redundancy*, i.e. a node may receive the same notification by a set of neighbors.

To remove this redundancy, two further forwarding models, *VoroCast* and *FiboCast*, have been proposed in (52). Both models have been introduced to reduce the bandwidth requirement of the direct connection model by limiting the connections of each node to its enclosing neighbors only.

VoroCast exploits Voronoi Diagrams to build a *spanning tree* covering all the AOI neighbors of an entity, while *FiboCast* dynamically adjusts the propagation range of a notification by exploiting a *Fibonacci sequence*, so that AOI neighbors of a node n receive updates at frequencies depending on their distance measured as hop counts from n.

VoroCast constructs a *multicast spanning tree* rooted at a node n and sends the notifications of events generated by n along the edges of this tree. In this way any redundancy in the transmission of the messages is avoided. Like in *VON*, the AOI is partitioned by a Voronoi diagram based on the coordinates of the AOI neighbors.

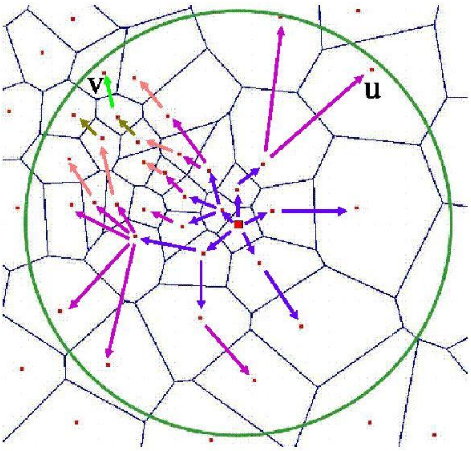


Figure 14: An unevenly crowded situation.

Messages generated by a root node **R** are first sent to all one-hop neighbors of **R**. Upon receiving the message, an intermediate node **x** forwards it to a *uniquely selected child* node within the AOI of **R**. To construct the spanning tree, the same node should be selected as child by a single neighbor node. *VoroCast* chooses as parent of a node n in the spanning tree, the neighbor of n which is nearer to the root **R** of the spanning tree. This requires that each node knows not only its one hop neighbor,but also the two-hop neighbors, which are the one-hop neighbors of its one-hop neighbors. In this way each node n may evaluate if a node x is its child in the spanning tree by comparing its distance from the root r of the spanning tree with the distance of any neighbor of x from r.

FiboCast adjusts the *message frequency* based on a *Fibonacci sequence*, which contains a series of numbers where each is the sum of the two previous ones (e.g., 0, 1, 1, 2, 3, 5, 8...). Different sequences can thus be created with different initial numbers.

This model has been defined to reduce the number of messages exchanged in *crowding*. As a matter of fact, when nearby nodes are sending simultaneously a notification, each node has to send its own messages plus those coming from its neighbors. Consequently, bandwidth depletion may still occur.

Consider, for example, the red node in Figure 14 and observe its AOI. The left side of the AOI is more crowded, so even if both node u and v are roughly equidistant from the red node, node u may "observe" the red node more easily since it belongs to the less crowded area and a smaller number of nodes are located in between itself and the red node. On the other hand, the red node may appear "obscured" to v due to the large amount of nodes located between them. In this case, it makes sense for the red node to send its notifications more frequently to u than to v. This can be obtained by *adaptively adjusting* the *transmission frequency* so that neighbors requiring more hops to be reached receive messages less frequently.

FiboCast adds two more fields to each notification, the *maximal hop count* and the *current hop count*. The *maximal hop count* is set according to a Fibonacci sequence, while the current hop count is incremented at each hop and it acts as a TTL. A notification forwarding stops when the vale of the current hop count equals the maximal hop count. As Fibonacci grows slowly at first but quickly later, nodes would receive messages with gradually decreasing frequency when the number of hops required to reach them starting from the root increases.

2.2.6 DiVES: A Distributed Support for Networked Distributed Virtual Environments

DiVES(19), that was developed by us before the work on this thesis, is a distributed support for the development of Networked Distributed Virtual Environments which exploits the *publish subscribe interaction model* to define a flexible communication support and to implement the *Areas of Interest*.

In a publish/subscribe system, hosts deliver *events* and publish them by *notifications*. Furthermore, hosts may express their interest in an event or in a pattern of events, in order to be notified of any event generated by a publisher and matching their interest. Matching of subscriptions and publications is generally supported by a *network of brokers* which also routes notification to interested subscribers.

In *DiVES* each host *publishes* its current position in a $2D$ virtual world through a notification. Furthermore each node delivers a *subscription* which defines the zone of the DVE corresponding to its *Area of Interest* and is described by a *filter*. The brokers of the *Broker Network* match the notifications with the corresponding subscriptions.

DiVES defines an *acyclic peer to peer network* of brokers to support an event based communication framework. The network can be dynamically reconfigured and it can tolerate broker crashes by a proper recovery mechanism.

DiVES also exploits a set of optimization strategies of the basic publish/subscribe routing mechanism defined through an accurate analysis of the information exchanged in *DVE applications*. The message traffic on the network is reduced by packing notifications and filters into a single message. Furthermore, approximated filters are introduced to further reduce message traffic. Advertisement are exploited to optimize the routing of filters.

Two alternative strategies can be adopted to implement an *Area of Interest*, the *Cell Based* and the *Entity Based* strategy. Both of them exploit *Communication Groups*. A *Communication Group* is a pair defining a communication channel and a group of hosts interested in receiving all the messages sent to that channel.

Cell Based

The *Cell based approach* defines an approximation of the *Area of Interest* by statically partitioning the shared world into cells and by pairing a different *Communication Group* to each cell.

A communication group is defined as the set of nodes whose subscriptions intersect a given cell C. Any notification delivered by any node belonging to C will be forwarded to all these nodes. The *Area of Interest* of a node is approximated by the region including the cell where it is currently placed and the surrounding ones. Each node dynamically joins and leaves communication groups while moving in the virtual world, during the evolution of the game. Each node notifies its position to the communication group corresponding to its cell and joins the communication groups corresponding to the surrounding cells. In this way, the node receives any message sent by nodes belonging to the surroundings areas.

This is implemented in *DiVES* by defining proper filters. The virtual world is partitioned into a set of square cells and each node delivers filters describing the square region including its cell and the neighboring ones. *DiVES* supposes that the Area of Interest of any node is always included in a cell of the virtual world. The filter delivered by a node *P* is not modified until *P* moves within the same cell. Each node is connected to a single broker, *Broker(P)*, and, for any movement, *P* notifies its new position to *Broker(P)*. When *P* crosses the border of a cell it notifies this event to *Broker(P)*, that exploits both the position of *P* and the map of the shared world to define its new filter.

For the sake of simplicity, let us suppose that a broker *B* is connected to a single node *P*. When *B* receives a notification from the network, it computes the Area of Interest of *P* by exploiting both the last position notified by *P* and the characteristics of *P*. Notifications received from nodes located outside the Area of Interest *P* are not sent to *P*. In this way, the notification filtering is performed by *Broker(P)* and *P* is not overwhelmed by useless messages. Both input bandwidth and computational load of *P* are therefore optimized.

The trade off of this approach is between the number of subscriptions, i.e. filters delivered on the network and the number of useless notifications delivered to each broker. Since a node delivers a new filter when crossing a cell boundary only, the number of delivered subscriptions can be controlled by tuning the size of the cells. Small cell size defines a good approximation of the area of interest of a node, but implies a large number of subscriptions.

Entity Based

The *Entity Based* approach pairs a distinct communication group with each node *P*, and *P* sends its position to this communication group. Each

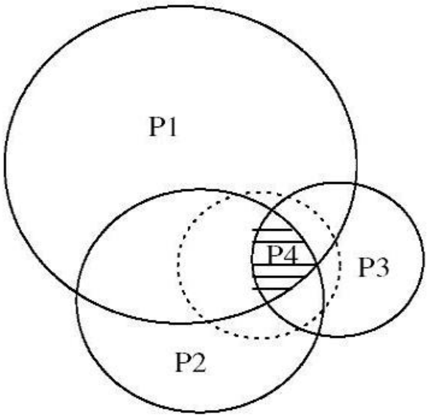


Figure 15: Entity-Based group definition

node joins the communication groups of all the nodes belonging to its Area of Interest and a communication group includes all the nodes whose areas of interest overlap.

Consider, for instance, Fig.15. The dashed area, corresponding to the intersection of the areas of interest of *P1*, *P2*, *P3*, defines a communication group including these nodes. Any notification sent within this area will be received by these nodes. It is important to notice that, even if a node *P* belongs to the area of interest of node *Q*, this does not imply that *Q* belongs to the area of interest of *P*. For instance node *P*4 belongs to the intersection of the area of interest of *P1*, *P2*, *P3*, but its area of interest, shown in the figure by the dashed line, does not include any of the previous nodes. This implies that all the notification of *P4* will be sent to *P1*, *P2*, *P3*, but no notification of *P1*/*P2*/*P3* will be sent to *P4*.

In entity-based *DiVES* each node *P* delivers a filter defining its exact Area of Interest. In this way, the position of each node belonging to this area will be notified to *P*. This approach cannot be directly implemented because it requires the exchange of a large amount of filters. As a matter of fact, the area of interest of any node *P* is modified at each movement of P. This implies that a new filter is generated for each movement of any node and this is the main drawback of this approach. On the other hand, since filters are more accurate than in the cell-based approach, routing of notifications improves and no useless notification are sent to a broker. Several optimizations to improve the effectiveness of this method can be exploited, and the resulting approach is a compromise between the cell-based and the entity-based approaches whose purpose is to reduce the message traffic.

Optimizations

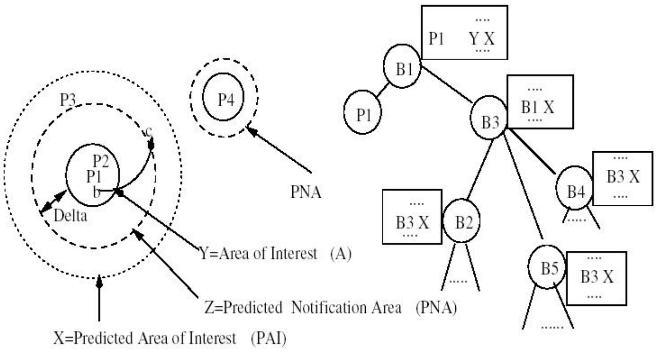


Figure 16: The Optimized Routing Algorithm.

DiVES considers the behavior of each node during an interval of time Δt and defines the *Predicted Notification Area* (PNA) of a node P as the set of positions that can be reached by P starting from its current position, traveling along a straight line, during the interval Δt. This set of positions is defined by a circle whose center is the current position of P and depends on the speed of P. *DiVES* also defines the *Predicted Area of Interest (PAI)* of P as the subspace of the virtual world including all the areas of interests corresponding to any point in the *Predicted Notification Area*. It can be easily shown that any *Predicted Notification Area* is a subset of the corresponding *Predicted Area of Interest*.

A node *P* subscribes its *PAI*, rather than its exact Area of Interest. The same subscription remains valid as long as the current Area of Interest of a node is included in its *PAI*. When adopting this approach the amounts of filters exchanged through the network, can be tuned by modifying Δt. Since Δt can be updated according to the characteristics and the state of each node, the resulting strategy is more flexible than the cell based approach. As in the cell-based approach, brokers filters incoming notifications according to the area of interest of their nodes.

As an example consider in Figure 16. The left part of the figure shows four nodes at different positions of the virtual world. The right part shows the brokers network and the relevant part of their routing tables. Let us consider node P_1. When it notifies its position to B_1, B_1 computes the area of interest Y of P_1, and its $PAIX$. Then B_1 forwards the PAI to its neighbors and the PAI will be flooded on the network. Since both P_2 and P_3 belong to the PAI of P_1, their positions will be notified to P_1 by the routing algorithm. B_1 discards any notification it receives from P_3, because it does not belong to Y . On the other side, the notification of P_2 will be sent to P_1 because it belongs to Y . Note that Y is notified to B_1 anytime P_1 moves, but B_1 does not propagate Y on the network. The position of P_4 is not routed to P_1, because P_4 does not belong to the PAI of P_1. Let us now suppose that P_1 moves from position b to position c. B_1 detects that the Area of Interest of P_1 does not belong to the PAI of P_1. The new PAI of P_1 is computed and flooded on the network. If P_4 now belongs to the PAI of P_1, its notifications will be routed to P_1.

A further reduction of the number of messages flowing in the network is obtained by an accurate analysis of the kind of messages produced by DVE applications. A new *Predicted Notification Area* is delivered at the same time of a new notification. A new *Predicted Area of Interest* is delivered when the current area of interest is not included in the previous *Predicted Area of Interest* and this is associated again to a movement of a node and, hence, to a notification. In this way, a larger amount of information is packed into the same message.

DiVES defines different kinds of messages. For instance,$position(x, y)$ corresponds to the notification of a new position, while $positionpai(x, y, pai)$

corresponds the notification of a new position and of a new PAI. Further messages are exploited in advertisement based routing. Each broker B_i routes a message $position(x, y)$ according its routing table, while the message $positionpai(x, y, pai)$ implies both the forwarding of the notification $notify(x, y)$ according to the routing table and the forwarding of the filter PAI to all the neighbors. The described approach is based on a simple filter routing strategy, that is flooding.

A mechanism based on *advertisements* is exploited to optimize the routing of filters on the network. In *DiVES*, the advertisements correspond to *Predicted Notification Areas*. As a matter of fact, *PNA* defines the positions each node can reach during the next interval of time Δt. These positions define the notifications P is going to deliver in the next interval. *Advertisements*, i.e. $PNAs$, are periodically injected in the network. Since a notification is considered valid only if it belongs to the last advertisement delivered on the network, a new PNA is produced when the node approaches the border of its current PNA. The distance from the border depends upon the latency of the underlying network, because any broker has to be notified before the node crosses its PNA. Routing of advertisements is based on flooding. As for PAI, a single message *positionpna(x, y, pna)* carries both a notifications and a PNA. Each broker propagates notifications according to the routing table and forwards advertisements to all its neighbors. Routing of filters is optimized, because each broker forwards any filter *F* only to the neighbors storing in their advertisement table at least an advertisement overlapping *F*. The dotted circle around P_4, in Figure 16, represents the PNA of P_4. This information is flooded in the network and is registered in the advertisement table of each broker.

Let us suppose that P_1 is located at b and that broker B_3 receives from B_1 the PAI of P_1. Suppose also that B_4 = *Broker(P4)* Since no overlap exists between the PAI of P_1 and the PNA of P_4, this PAI is not sent to B_4.

A further optimization is related to the reduction of the entries of the routing tables. A broker can *merge* the filters corresponding to existing routing entries and forward the merged filter to a subset of its neighbors.

Merging filters introduces another level of approximation in the routing algorithm. *DiVES* adopts a simple strategy to merge filters. Two filters are merged if and only if the size of the resulting region does not exceed a given threshold. This guarantees that the corresponding filter defines a limited region of the shared world.

2.2.7 JaDE: a JXTA Support for Distributed Virtual Environments

JaDE(71), which was developed by us during the analysis of instruments used in this thesis, is a P2P support for the development of Distributed Virtual Environments that improves *DVE* scalability through the notion of Area of Interest. *JaDE* defines a set of protocols to support both the active entities and passive objects of the *DVE*. The state of passive objects is replicated on a set of peers to increase the reliability and the responsiveness of the application. Since passive objects may be concurrently updated by the active entities of the *DVE*, a novel consistency protocol is defined together with a set of mechanisms to guarantee the persistence of passive objects in a *DVE* environment.

JaDE: Protocol Specification

A basic choice for the definition of *JaDE* protocols is the adoption of the concept of Area of Interest to improve the *DVE* scalability. *JaDE* statically partitions the *DVE* into a set of regions whose shape and extension depend upon the characteristics of the *DVE*. *JaDE* assume that, a peer P located in a region R of the *DVE* renders, at any instant of time, the events occurring in R only. For this reason, at any instant of time, the *AOI* of P includes at least R. Each peer periodically

- Sends its current positions to the any other one in its *AOI* through an *heartbeat message*.
- Receives the position of other avatars in its *AOI*.
- Updates its local view of the *DVE* through the received messages.

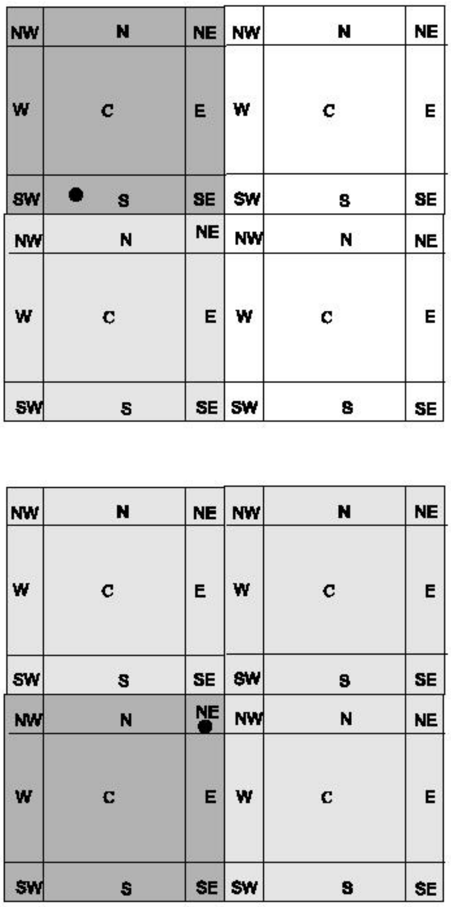


Figure 17: Definition of Area of Interests.

The *DVE* is decomposed into a set of equal, square regions as those shown in Figure 17 where the main region of a peer is the *DVE* region where a peer is located. Since a peer dynamically moves within the *DVE*, its *AOI* may change and anytime it enters a new region *R*, it must be initialized with the state of each passive and active entity located in *R*. Then, as far as the peer stays in *R*, it is interested in any event occurring in its *AOI*, such as the update to the position of any other peer or to the state of a passive object in *R*. Since in a *WAN* the latency of any notification mechanism cannot be neglected, each region of the *DVE* is further partitioned into a *central zone* C and eight peripheral zones, as shown in Figure 17. When the peer *P* stays in *C*, its *AOI* overlaps its main region. When the peer approaches the border of its main region and enters a peripheral zone, it starts prefetching the state of the entities of the region *R* it is going to enter. In this way, it initializes its *AOI* before entering *R*. Note that any peer in *R* must promptly detect the presence of *P* as well. A straightforward implementation of this prefetching mechanism just requires the extension of the *AOI* of a peer when it enters a peripheral zone. For instance, the *AOI* of the peer displayed by the black circle in the top part of Figure 17 includes its dark grey main region *R* and the southern light grey neighbor region of *R*. For the same reason, the *AOI* of the peer located in the northeastern peripheral region in the bottom part of the Figure 17 overlaps the whole *DVE*. To avoid that peers belonging to the new region perceives a delay in detecting a new peer entering their main region, the entering peer should notify its presence to any peer in its extended *AOI* as well. To reduce the number of events that are prefetched from a neighboring region before entering it, the size of the enlarged *AOI* may be reduced. For instance, a peer entering a new region may be interested in initially perceiving the state of the entities close to the border of the new region while acquiring the knowledge of the state of the whole region later.

JaDE adopts the *Sequential Consistency*(30; 31; 34) model which guarantees that, while any interleaving to passive objects updates may be accepted, all the peers observe the same interleaving of the updates of the objects in their *AOI*. *Sequential Consistency* may be easily implemented in client server architectures where a central server manages the state of any passive object forwarding them to any interested client, while its implementation in a P2P environment is more complex. A similar solution may be adopted in a P2P environment as well by the *dynamic election* of one of the peers of a region *R* to manage the state of the passive objects. Even if this solution is more scalable, because it distributes the load among a set of servers, the election of a single server for each region still introduces a bottleneck resulting in both a lower *DVE* responsiveness and a lower reliability. On the other hand, a fully distributed solution which replicates the state of any passive object to each peer of a region may be adopted. Here, each peer holds a local copy of the objects of its *AOI* and updates their state by accessing its copy. This approach increases the reliability of the *DVE* because the crash or the voluntary departure of a peer does not imply the loss of the objects of a region. Also the *DVE* responsiveness improves because concurrent updates are possible. However, a mechanism to preserve the consistency of replicated copies in spite of concurrent updates has to be adopted. Note that this situation often occurs in a *DVE* because an object may ”attract” peers so that a typical *DVE* scenario is a crowd of peers that try to modify the same object. *JaDE* exploits object replication to improve responsiveness and reliability. Each peer stores in a local cache, the *Object Cache*, the state of any object in its *AOI*. The cache is initialized when the peer enters a region and flushed when it leaves the region in order to avoid to overwhelm the cache with a large amount of useless information. As far as concerns object consistency, several approaches have been proposed to guarantee the consistency of multiple replicated copies in the presence of concurrent updates. It is worth noticing that while a process of a concurrent application is not generally aware of the other processes which may concurrently update a copy of data, in a *DVE* each peer is always aware, because of the heartbeats notifications, of the positions of the other peers of its main region and it may detect when a crow of peers gathers around a shared object. *JaDE* exploits this property to optimize the consistency protocol. This protocol is described in the following section.

Another important issue in the definition a P2P support is the definition of a set of mechanisms to guarantee the persistence of the objects belonging to regions which are not inhabited by any peer. A region may be inhabited because either no peer has still visited it or every peer has left it. *JaDE* assumes that each peer holds a map of the whole *DVE* that includes static objects, like landscapes, trees and other graphical elements. Some of these objects may be modified by the peers, while others are immutable. We are interested in objects which may be modified dynamically. If the peer modifies some object, the object is activated, i.e. a data structure is allocated by *JaDE* to store its state. This state is replicated to the peers of that region and, to avoid that the state is lost if all of them exit the region, the last peer leaving the region stores the state of any object in a *Backup Cache*. A more complex situation is that where the last peer of the region leaves the *DVE*. If its departure is voluntary, this peer may choose another one in the *DVE* and send to this peer the content of its local cache. The chosen peer stores these objects in its *Backup Cache*. In both cases, the peer holding the objects of the region becomes the *Backup Peer* of the objects. Note that a set of *Backup Peers* may be defined to take into account abrupt peer crashes. It is worth noticing that the identity of the *Backup Peer* must be notified to all the peers in the *DVE* because they need to find out the objects when they enters the region. As a matter of fact, when a peer *P* enters a region it first checks if it is inhabited by any peer. In this case, it chooses at random a peer in the region and asks it for the region objects, otherwise *P* must contact the *Backup Peer*.

JaDE: Passive Object Consistency

JaDE consistency protocols exploit the relative positions of the peers and the knowledge of the maximum latency of the underlying notification mechanism to detect scenarios where replicated objects may be concurrently updated. It is worth noticing that in a *DVE* each peer may update an object only if it is close to it. For instance, a peer should be close to a magic potion to drink it. If the peer is far from the potion, it may throw a stone to break the bottle containing the portion. In any case, for each object *O*, we can define an *Update Area*, that is the portion of the *DVE* region where a peer must be located in order to modify O. This area is different for different kind of objects. In *JaDE*, it is a circle centered at the object location. The updates performed by a set S of peer in the *Update Area* of an object O should be considered as concurrent ones because it is likely that a peer in S modifies O before receiving a previous update of O by another peer in S. On the other hand, if the latency of the underlying notification mechanism is high, even a peer located outside the *Update Area* of O may enter it and update O before receiving the updated value of O. For this reason, *JaDE* defines the *Conflict Area* of each object as a larger, circular area centered at the object. The radius of the *Conflict Area* of an object O should be defined so that the time interval required to reach the *Update Area* of O from any point in its *Conflict Area* is smaller than the interval of time required to notify an update to O to any peer of the region. The radius of the *Conflict Area* depends upon both the larger peer speed and the maximum latency of the mechanism to notify updates. *JaDe* assumes that the *Update Area* of an object is always included in its *Conflict Area*. When a peer P in the *Update Area* of an object O modifies it, P checks if the *Conflict Area* does not include any other peer and, in this case:

- Reads the current state of O from the local cache, because its copy of O is up to date
- Notifies the update to any peer in its main region, since they will receive the update before entering the *Update Area*.

On the other hand, if P finds out at least another peer in the *Conflict Area*, it should exploit a mechanism to guarantee the consistency of O in presence of potential concurrent updates. Totally-ordered Multicast based on Lamport's *timestamps*, may be exploited to guarantee that each peer orders concurrent updates in the same way since the update messages are delivered in the same order to each peer. However, the implementation of this mechanism requires a high amount of messages, that results in a low scalability and prevents its adoption in a large distributed systems. *JaDE* solution is based upon the distributed definition of a *coordinator* for each object of a region R. In *JaDE* each peer may create a new object, which is not present on the static map of the *DVE*, or it may activate an object, if the object is initially present on the static map, but it has not been modified yet by any peer. The peer which creates or activates an object O becomes the coordinator of O. As soon as it is elected, the *coordinator* informs all the peer in its main region of the election and detains the coordination of the object as long as it remains within the region. When the *coordinator* leaves R it passes the coordination to another peer in R. The coordinator is the unique owner of the object O, it holds the up to date state of O and it serializes the updates when concurrent updates to O occurs. When a peer updates an object whose *Conflict Area R* is not empty, it sends the update to the *coordinator* which resolves the update conflicts among the peers in the *Conflict Area*. Then the *coordinator* sends the state of the updated object to any peer of its main region. *JaDE* exploits a timestamp based mechanism to resolve the conflict among peers trying to activate the same object concurrently so competing for the acquisition of the coordination of the object.

2.3 Conclusions

All the proposals described in this chapter are characterized by the general idea to obtain, for each peer, the *maximum degree of consistency* in the portion of DVE close to it. Furthermore, each protocol is designed to cope with problems such as the *crowding* and the *persistence* of the state of the DVE.

In *SimMud* direct links are maintained between neighbors, so transmission latency is reduced (e.g. messages are exchanged directly, not through intermediate nodes). However, constant exchange of neighbors list introduces network overhead (if 10 nearest neighbors are kept, one exchange requires receiving updates of 10x10 nodes). The more serious problem is keeping the topology fully-connected. Since only a finite number of nearest neighbors are maintained, groups of users may lose contact to each other if they are far away each other.

In *Solipsis* an inconsistent topology may occur during normal operation occasionally, since an incoming node may be unknown to directly connected-neighbors, proper neighbors discovery is thus not guaranteed. In some other cases, proper neighbors discovery could be slow as it may require a few queries.

In *Apolo* the restricted direction links make the message transmission inefficient. A node cannot forward the message to the nearby node even if there is a link between the two nodes.

In *Jade* the biggest limitation is the presence of static areas of interest. One of the main advantage of $Jade$ is the definition of a mechanism for the management of the persistence of the passive objects and the protocols for to support the concurrent updates of the objects.

The main drawback of *Dives* is the presence of a set of brokers implementing the publish subscribe support which introduce an indirection level in the transmission of the events between the peers.

Von is, according to our point of view, the best approach of those introduced in this chapter. However, the approach needs the definition of proper mechanism for the management of the passive objects, to balance the load among the peers of the DVE, and for the management of the persistence of the state.

In the following chapters we will define a scalable approach for the management of the DVE based on Voronoi tessellations. Our proposal will extend those presented in the literature in several directions. First of all, a set of mechanisms to support passive objects management will be defined. The proposed solution will manage both the consistency and the persistency of the DVE state by exploiting the properties of the Voronoi tessellations. We will also propose and evaluate an hybrid P2P architectures based on *Additively Weighted Voronoi graphs* (Apollonius graphs)(56), where each peer may have a different role in the P2P network according to its computational/communication power. This architecture will manage an heterogeneous network that includes peers characterized by different network bandwidth, CPU power and memory capacity. Moreover, passive objects will be assigned to peers such that the load for the management of the passive objects is balanced.

// Chapter 3

Location Aware Reactive Computations

This chapter presents some formalisms which can be exploited to specify the behavior of DVE applications. In particular, we will introduce *Mobile Unity*(67) and *Ambient*(66). *Mobile Unity* is an extension of *Unity*(70) supporting components location and transient interactions. *Ambient* proposes a modal logic to describe the structural and computational properties of distributed and mobile computations. Both formalisms are able to model *locality in the interactions* among the components of a distributed application, which is a basic issues for DVE applications. Furthermore, the reactive behavior of a peer can be naturally modeled by the reactive statement of *Mobile Unity*. For these reason, *Mobile Unity* has been exploited as a specification formalism for both the heartbeat routing protocol defined in Chapter 4 and for the protocol for the management of the passive objects, which has been defined in Chapter 5.

3.1 Ambient

The inspiration for *Mobile Ambient*(66) comes from the potential for mobile computation over the World Wide Web. The geographic distribution of the Web introduces the problem of modeling the mobility of computations. There are two distinct areas of interest in mobility: mobile computing, which describe the ability to use technology that is not physically connected or in remote and mobile environment, and mobile computation, which concern mobile code that moves between devices (applets, agents, etc.). *Mobile Ambient* aims to describe both these aspects of mobility within a single framework that encompasses mobile agents, the ambients where agents interact and the mobility of the ambients themselves. With these motivations, *Mobile Ambient* adopts a paradigm of mobility where computational ambients are hierarchically structured, where agents are confined to ambients and where ambients move under the control of agents.

The main characteristics of an ambient are the following ones:

- *An ambient is a bounded place where computation happens*. The interesting property here is the existence of a boundary around an ambient. Examples of ambients, in this sense, are: a web page (bounded by a file), a virtual address space (bounded by an addressing range), a Unix file system (bounded within a physical volume), a single data object (bounded by 'self') and a laptop (bounded by its case and data ports). Non-examples are: threads (where the boundary of what is 'reachable' is difficult to determine) and logically related collections of objects. A boundary implies some flexible addressing scheme that can denote entities across the boundary. Examples are symbolic links, Uniform Resource Locators and Remote Procedure Call proxies. Flexible addressing is what enables, or at least facilitates, mobility.

- An ambient is something that *can be nested* within other ambients. For instance, administrative domains are (often) organized hierarchically. If we want to move a running application from work to home, the application must be removed from an enclosing (work) ambient and inserted in a different enclosing (home) ambient. A laptop may need a removal pass to leave a workplace, and a government pass to leave or enter a country.

- An ambient is something that can be moved as a whole. If we reconnect a laptop to a different network, all the address spaces and file systems within it move accordingly and automatically. If we move an agent from one computer to another, its local data should move accordingly and automatically.

3.1.1 The Folder Calculus

An ambient can be represented as a *folder*. A folder confines its contents: something is either inside or outside any given folder. Each folder has a name that is written on the folder tag. Folders are naturally nested, and can be moved from place to place. The computational aspect of the calculus is represented by assuming that folders are active. In addition to subfolders, folders may contain entities that cause the folder to move around, and are moved together with their folder. The folders example can be used to better explain the textual syntax of *Mobile Ambient*. As we can see in the table showed in Figure 18 there is a one-to-one correspondence between textual syntax and visual syntax; therefore, it is possible to freely mix them, if desired, nesting either one inside the other expressions. As shown in the table, P, Q, R are used to range over ambient and process expressions, and M, N to range over message expressions. The creation of a new name is written $(\nu n)P$ where the Greek letter $\nu(nu)$ binds the name n within the scope P. An ambient is written $n[P]$ where n is the ambient name, where the brackets denote the ambient boundary, and where P is the contents of the ambient.

3.1.2 An Example: The Adobe Distiller

Adobe Distiller is a program that converts files form the Postscript format to Adobe Acrobat one. The program can be set up to work automatically on files that are placed in a special location. In particular, when a user drops a Postscript file into an inbox folder, the file is converted to Acrobat format and dropped into a nearby outbox folder. Figure 19 describes such a behavior. The distiller folder contains the inbox and outbox folders mentioned above; outbox is initially empty. The input folder contains the file the user wants to convert, in the form of a message. The

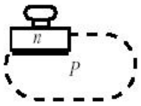
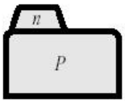
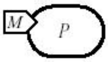
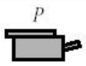
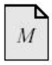
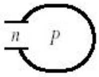
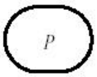
$(\nu n)P$		New name n in a scope P.
$n[P]$		Folder (ambient) of name n and contents P.
$M.P$		Action M followed by P.
$P \mid Q$	$P \quad Q$	Two processes in parallel. (Visually: contiguously placed in 2D.)
0	○	Inactive process (often omitted).
$!P$		Replication of P.
$\langle M \rangle$		Output M.
$(n).P$		Input n followed by P.
(P)		Grouping.

Figure 18: Mobile Ambient textual and visual syntax.

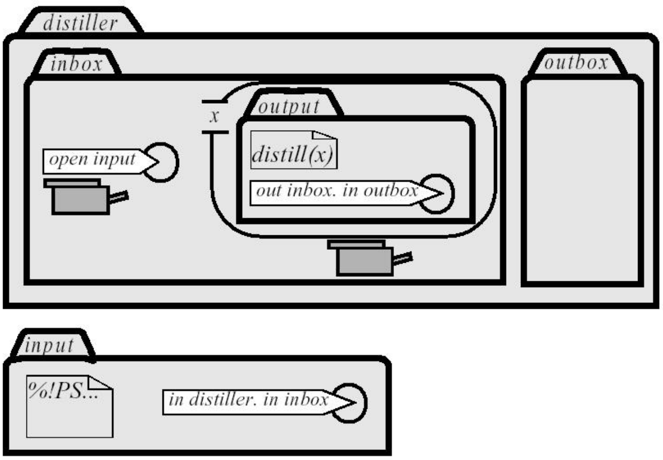


Figure 19: Adobe Distiller as folders representation.

input folder contains also a gremlin that moves the input folder into the inbox.

The inbox contains the program necessary to do the format conversion and drop the result into the outbox. First, any input folder arriving into the inbox must be opened to reveal the Postscript file. This is done by the copy machine on the left. Then, any such file is read and this is done by the copy machine on the right. As a result of each read, an output folder is created to contain a result. Inside each output folder, a file is distilled (by the external operation distill(x)) and left there as an output. The output folder is moved into the outbox folder. It should be noted that the program above represents highly concurrent behavior, according to the reduction semantics of the folder calculus. Multiple files can be dropped into the inbox and can be processed concurrently. The opening of the input folders and the reading of their contents is done in a producer-consumer style.

This, instead, is the textual representation of the Adobe Distiller example:

```
Distiller[
    inbox[
        !open input |
        !(x) output[.distill(x). | out inbox. in outbox]] |
    outbox[]]
|
Input[."%!PS...". | in distiller. in inbox]
```

3.2 Mobile Unity

Mobile Unity (67) is an extension of *Unity* which provides a programming notation that captures the notion of *mobility* and of *transient interactions* among mobile nodes and includes an assertion-style proof logic. Code mobility is defined informally as the capability to *dynamically reconfigure the binding* between code fragments and the location where they are executed. This reconfiguration may occur in different scenarios. For instance in an agent based computation, an agent may migrate and be executed on different physical nodes. On the other hand, even if a computation C is tied to a single host H, the environment where C is executed may change if H is a mobile device, due to network connections and disconnections. The *Mobile Unity* model adheres to the minimalist philosophy of the original *Unity* so that it focuses only on essential abstractions needed to cope with the presence of mobility. In section 3.3, we will show that the mobility model of *Mobile Unity* and its notion of locality aware interactions can be naturally exploited to define the behavior of peers in a DVE. *Mobile Unity* will be exploited in Chapter 4 to define the heartbeat routing protocol, and in Chapter 5 to define the protocol for the management of the DVE passive objects.

3.2.1 Location Aware Computations

Mobile computing systems must operate under conditions of transient connectivity. Connectivity will depend on the *current location* of the components and therefore location is part of the *Mobile Unity* model. In *Mo-*

bile Unity, the unit of mobility is a program. Migration is captured by augmenting the program state with a *location attribute*, defined by a location variable λ, whose change in value is used to represent *motion*. In this way, *Unity* is augmented with an explicit representation of space and of its properties. *Mobile Unity* like standard *Unity* does not constrain neither the types of programs variables nor the types of the location variable λ. This variable may define single or multi-dimensional coordinates, it may model the latitude or the longitude of a physically mobile platform, or may be a network or memory address for a mobile agent. A program may have explicit control over its own location by the assignment of a new value to the variable modeling its location.

For instance, a mobile receiver might contain the following statement

$$\lambda := NewLoc(\lambda)$$

where the function $NewLoc$ returns its new location, given the previous one. In general, such an assignment could compute a new location based on arbitrary portions of the state of the program as well. Even if the process does not have explicit control over its own location, we can still model its movement by an internal assignment statement that is occasionally selected for execution.

3.2.2 System specification

This section first review the structure of a standard Unity program, afterwards the extensions defined by Mobile Unity are introduced.

Unity programs are set of assignments that are selected for execution in a *weakly fair manner*, that implies that in an infinite computation each statement is scheduled for execution *infinitely often*. Furthermore each statement is executed atomically.

Figures 20 and 21 show a simple system specified by Unity. The figures show a sender and a receiver exchanging bit values. We can note that each program first introduces, in the **declare** section, the variables it uses. Note that abstract variable types such as sets and sequences can be used freely. The **initially** section may define an initial values for the variables of the program, if a variable is not referred in this section, its initial

```
program sender
    declare
        $bit : boolean$
    initially
        $bit = 0$
    assign
        $bit := 0$
    $\Box$ $bit := 1$
end
```

Figure 20: The Sender

```
program receiver
    declare
        $bit : boolean$
    $\Box$ history : sequence of boolean
    initially
        $bit = 0$
    $\Box$ $history = \epsilon$
    assign
        $history := history * bit$
end
```

Figure 21: The Receiver

value is a default value defined only by its type. The core of a Unity program is the **assign** section which consists of a set of atomic assignment statements.

The assignment in these programs are single assignment statements, but, in general, several right hand expressions may be atomically assigned to several left hand variables. In this case all the right hand expressions are evaluated in the current state before any assignment to variables is made. The basic execution mechanism of Unity is a *nondeterministic fair interleaving* of the assignment statements. Each statement produces an atomic transformation of the program state. At each computation step, one statement is selected for the execution and the program state is atomically modified according to this statement. Fairness is guaranteed, that is no statement is excluded from selection forever.

Unity introduces a *static composition mechanism*, defined by the operator $\Box$, to construct a new system from several program units. For instance $sender\Box receiver$ is a system including the *union* of the program variables of $sender$ and of $receiver$, where the variables with the same name refer to the same physical memory location, the union of the assignment statements of the two program units and the intersection of the initial conditions. The resulting assignments are executed according a fair atomic interleaving.

Note that the variable bit, which is defined both in the sender and in the receiver, is the shared medium exploited for their interaction. The sender writes an infinite sequence of 0 and 1 to this variable, while the receiver occasionally reads a value from this variable.

If the two programs sender and receiver represent mobile component, or software running on mobile hardware, then it is not appropriate to represent the resulting system by a *static composition* of the sender and the receiver. Composition by standard Unity union prohibits dynamic reconfiguration and disconnection of the components that characterize the mobile computing systems. For instance, in standard Unity, the variable bit is shared between the sender and the receiver during the entire execution, thus inhibiting disconnections and reconnections.

Mobile Unity instead ensures the *isolation of the namespaces* of the individual processes, and assumes that variables that are associated with distinct programs are distinct even if they have the same name. For example the variable *bit* in the sender is no longer shared with the variable *bit* in the receiver, they are distinct variables and can be specified by prefixing the name of the component in which they appear to their name, for instance *sender.bit* or *receiver.bit*.

Figure 22 illustrates the structure of the sender receiver system specification in *Mobile Unity*. To start we find the system name declaration, then a set of programs, for instance the two programs explained above, with the addiction of the program variable λ, which stands for the current location of the program. The two programs are here type declara-

```
System sender-receiver
    program sender at $\lambda$
    ...
    end
    program receiver at $\lambda$
        ...
    end
    Components
       receiver at $\lambda_0$
    □ sender at $\lambda_0$
    Interactions
        ...
end
```

Figure 22: System specification in Mobile Unity.

tions instantiated in the **Components** section.

The *Interaction Section* defines a set of *transient interactions* among program instances. These interactions should be *context dependent*, i.e. they should occur only when certain conditions are verified. Suppose, for instance, that the sender and the receiver can only communicate when they are at the same location. In this case the Interaction section should include the following statement

$$receiver.bit := sender.bit \textbf{ when } receiver.\lambda = sender.\lambda$$

The statement copies the value of $sender.bit$ to $receiver.bit$ when the two program units are at the same location.

Finally note that *Mobile Unity* extends the set of constructs of standard Unity by the following constructs:

- *Transactions*: provide a form of sequential execution. They consist of sequences of assignment statements which must be scheduled in the specified order with no other statements interleaved in between. The assignment statements of standard *Unity* may be viewed as singleton transactions.

- *Inhibitors*: provide a mechanism for strengthening the guard of an existing statement without modifying the original. This construct permits us to simulate the effect of redefining the scheduling mechanism so as to avoid executing certain statements when their execution may be deemed undesirable.

- *Reactive statements*: provide a mechanism for extending the effect of individual assignment statements with an arbitrary terminating computation. Since these statements have been widely exploited for the specification of our protocols, they will be described in the following section in more details.

3.2.3 Reactive Statements

A construct unique to *Mobile Unity* is the *reactive statement* which provides a mechanism for extending the effect of individual assignment statements with an arbitrary terminating computation. The syntax of an assignment statement is the following:

s reacts-to p

where ***s*** is an assignment which is extended by the reaction clause ***p***, where ***p*** is an arbitrary predicate. The informal semantics of a reactive statement is that any assignment A which makes the predicate ***p*** true triggers a set of reactions, i.e. all the assignment guarded by the clause ***p*** must be executed. Operationally, we can think of each assignment in a *Mobile Unity* program as being extended with the execution of all triggered reactions up to such point that no further state changes are possible by executing reactive statement alone. More formally, the set of reactive statements forms a program that is executed to fixed point after each atomic state change by the assignment of the program. Clearly this program must be terminating.

It is worth noticing that both event driven computations and interrupt processing mechanisms can be easily modeled by *MobileUnity*.

Figure 23 shows a program exploiting the **reacts-to** construct. It consists of two non-reactive statements (one of which is a transaction in-

```
program simple-example
    declare
        x,debug : integer
    initially
        $x = 0$
      □ debug = 0
    assign
        $s :: x := x + 1$
      □ t :: $\langle x := x + 1; x := x - 1 \rangle$
      □ inhibit s when x $\geq 15$
      □ debug := x reacts-to $x > 15$
end
```

Figure 23: React-to example

cluding the sequence of statements between the angle brackets), one inhibiting clause, and one reactive statement. The statement s increments x by one. The statement t is a transaction consisting of two substatements. The first increments x by one. The second decrements x by one. The programmer might add the inhibiting clause to prevent x from being incremented past 15. This prevents statement s from performing this action, but the statement t may still execute and temporarily increase x to 16. In this case, the reactive predicate becomes true, the reaction is immediately triggered and the assignment of x to the variable debug is executed.

Reactive statement can be exploited to model reactive interactions in the Interaction section. Consider again the program shown in Figure 22 and suppose that the Interaction section includes the following statement

$$receiver.bit := sender.bit \textbf{ when } receiver.\lambda = sender.\lambda$$

This statement is considered as an additional program statement which is executed in interleaved fashion with other program statements. This does not guarantee that each value written in the bit variable by the sender is received by the receiver. To guarantee that each value written to the value bit by the sender is received by the receiver when they are at the same location, the following statement can be exploited

$$receiver.bit := sender.bit \textbf{ reacts-to } receiver.\lambda = sender.\lambda$$

The reactive statement is scheduled for the execution as soon as the two program unit are at the same location. Note that the reactive statements can be treated as higher priority statements and that all reactive statements have the same priority.

3.3 Modelling DVE by Mobile Unity

As shown in the previous sections, *Mobile Unity* has been introduced mainly to describe mobile computations. The location variable λ may be exploited to define the location of a program unit, for instance the IP address of the host where the program is currently executed. Interactions among program unit may be location aware, that is they may occur only if the locations of the interacting programs satisfy some relation.

These characteristics are exactly those which are needed to model the behavior of peers in a DVE application. As a matter of fact, there is nothing in the model that precludes a more abstract view of space. In our case we can exploit the location variable λ to define the position of the peer in the virtual space and the new value of λ is assigned according to the mobility model defined by the DVE application.

The time is a very important aspect of a DVE because of its real-time nature. On the contrary of Mobile Ambient model, Mobile Unity, allows, in a simple way, to model the concept of time (with an incremental counter) and then to specify and manage the real-time actions among peers trough the **react-to** statement.

Note also that the interactions among the peers of the DVE are location aware, for instance a peer sends an heartbeat to the peers which are located in its *Area of Interest*. These interaction can be modeled by a set of clause in the Interaction section of Mobile Unity, where each interaction may be guarded by a predicate defining a relation between the positions of the interacting peers.

Finally, the events characterizing our protocols may be naturally captured by the reactive statements of Mobile Unity, For instance, as we will see in the following, a reactive statement could be useful to model the routing protocol for the propagation of the heartbeats among the peers. For instance, the transmission of an heartbeat executed by a peer P may cause the triggering of a set of reaction corresponding to the reception/ forwarding of the heartbeat by the peers belonging to the Area of Interest of P.

Then the **react-to** statement of Mobile Unity allows to model the event-driven interaction of a DVE in a more natural way than Mobile Ambient does.

Chapter 4

Voronoi Based Overlays for DVE

4.1 Introduction

The definition of a scalable communication support is a basic issue for the wide diffusion of *DVEs*. Several proposals exploit the concept of Area of Interest, *AOI* (1; 57), to define a scalable communication support. The implementation of the *AOI* concept requires the definition of a *highly dynamic P2P overlay*. This chapter presents our solution for the definition of a *dynamic overlay network* for *P2P DVEs*. The chapter first introduces the mathematical concepts upon which this solution is based, then a strategy for the definition of the dynamic overlay and a routing strategy for heartbeats propagation is defined. Finally, a set of experimental results are presented.

Section 4.2 introduces the general strategy we exploit to define the concept of *Area of Interest* and the problems related to particular situations which may occur in a DVE, like *crowding scenarios*. Section 4.3 introduces Voronoi diagrams and defines the mathematical structures which are exploited in our approach, while Section 4.5 shows how the *P2P overlay* may be defined by exploiting *Voronoi Tessellations* and the corresponding concept of *Delaunay graph*. Section 4.4 introduces *Compass*

Routing, a routing strategy defined in (69) for general *Delaunay networks*, while Section 4.4.1 shows that *Compass Routing* can be exploited to route heartbeats within a constrained area like the *Area of Interest* of a peer. The routing strategy we propose is introduced in Section 4.5 which includes a formal specification of the routing based on the *Mobile Unity* framework introduced in chapter 3. Section 4.6 includes considerations about overlay partitioning. Finally Section 4.7 shows a set of experimental results resulting from a set of simulations developed through *PeerSim* and a prototype developed on the *GRID5000* platform, in this section will be introduced a new mobility model to test the behavior of the overlay under more realistic assumptions.

4.2 Improving DVE Scalability by Areas of Interest

The *Area of Interest*, *AOI*, of a player P is a region of the virtual world surrounding P and such that P is aware only of other players and passive objects located in this area. Each peer generally notifies *its position* to the others ones located in its *AOI* with a high temporal frequency (up to 5 times per second). This message is generally referred (52; 53) as *heartbeat*.

Each peer may propagate within its AOI other events as well, like actions executed by the corresponding avatar or updates to passive objects. However these events are less critical, since they occur less frequently.

As discussed in Chapter 2, several proposals exploit the concept of *Area of Interest* to define a scalable communication support (1; 57) for the *DVE*. As a matter of fact, the *AOI* reduces the number of communications within the DVE since each event e is notified only to peers which are interested in e.

The implementation of the *AOI* concept requires that a peer P dynamically defines a set of connections with all the peers located in its *AOI* or with a subset of these peers. In the first case, the heartbeats are directly sent from P to the other ones, while in the latter case, an *AOI-cast* mechanism should be defined, i.e. a routing strategy to propagate the heartbeat to all the peers located in the *AOI*. In both cases, the over-

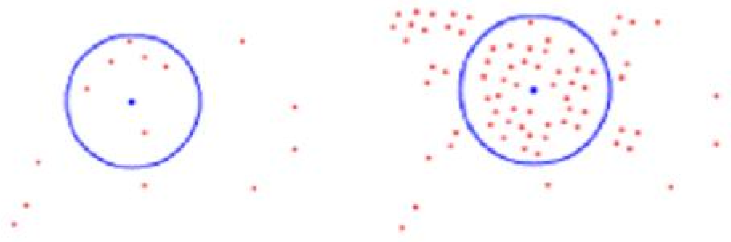


Figure 24: DVE Scenarios

lay changes over time because, due to the movement of the peer, new peers may enter its *AOI*, while others may leave it. Furthermore, a set of mechanisms should be defined to guarantee the connectivity of the overlay when the *DVE* is scarcely populated, i.e. the *AOI* of the peers are empty.

Note that even if the *AOI* increases the scalability of *DVE*, it may be not enough when a *crowding scenario* occurs. In a crowding scenario, like that shown in the right part of Figure 24, an high number of avatars is located within a small virtual space. Figure 24 compares a crowded scenario (shown in the right part) vs. a not crowded one (shown in the left part).

Different crowding scenario may occur in a DVE.

- *Battle Crowding.* In *player vs. player (PVP) MMOGS*, a large amount of avatar may meet in a DVE region to fight a virtual battle. In this situation the speed of the avatars is generally high, hence their interaction pattern changes very frequently. While interactions among avatars are very frequent, interaction between the avatars and the passive objects of the virtual world are less frequent.

- *City Crowding.* In a DVE virtual city, avatars generally gather in order to carry out some social activity, like buying objects, eating or drinking and so on. As a consequence, the interactions with passive objects are very frequent, and the speed of the avatars is generally low.

Any proposal must handle crowding in a very scalable way to be eligible to support a real-time DVE. The concept of AOI may not suffice to define a scalable communication, since the amount of peers in the *AOI* may be too high when a crowding scenario occurs.

Two different approaches may be exploited in this case. One (1) is based on the definition of a set of *direct links* between a peer and any other one in its AOI. This solution *minimizes the latency* because it avoids a large amount of routing hops for the propagation of the heartbeats within the crowded *AOI*. Its main drawback is that it increases the number of connections of each peer because, in a crowding scenario like the ones previously described, a peer should manage a large number of connections, since a large amount of peers are located in its AOI. The approach of dynamically enlarging or shrinking the size of the *AOI* (1) according to the bandwidth of the peers is not fair, because the size of the *AOI* depends upon the semantics of the application and players with a larger *AOI* could be favored.

An alternative approach is the definition of an *AOI-cast mechanism* (52), i.e. an application level multicast constrained within the boundary of the area of interest. In this case each peer P is connected to a subset of the peers belonging to its AOI (for instance the nearest ones) and any event is sent to these peers only. A suitable *routing mechanism* should then be defined to propagate any event to each peer located in the AOI of P. The simplest approach is based on *flooding*, i.e. each peer receiving a notification should propagate it to all its neighbors. This approach generates a large amount of redundant messages and presents evident scalability problems. A more refined approach is based on the definition of a proper *routing mechanism* which dynamically computes a spanning tree including all the peers of the *AOI* and then exploits these links to notify the heartbeats to the peer of the *AOI*. Both solutions are based on *forwarding*, i.e. any heartbeat is routed to the peers in the *AOI* through a sequence of intermediate peers.

The main advantage of this approach is that the number of connections which must be managed by each peer is reduced with respect to previous solution. An obvious drawback is the *high latency* in the delivery of an event, especially in crowding scenarios. In this case several routing hops may be required to notify an event due to the large amount of peers located in the *AOI*. The resulting latency may be not tolerable in a *MMOG* and may compromise the interactivity of the application.

In the following, we will propose *an intermediate solution*, which defines direct connections between a peer P and *a subset* of the peers in its *AOI*, i.e. the peers nearer to P, and exploits forwarding to reach any other peer of the *AOI*. Our solution is based on the definition of a *Voronoi based overlay* and of a routing algorithm which exploits the mathematical properties of the corresponding *Delaunay Triangulation*. The following sections will describe our approach in more detail.

4.3 Mathematical Definitions

4.3.1 Voronoi Diagrams and Delaunay Triangulations

In this section we introduce the mathematical concepts required by our approach.

A *Voronoi diagram*, (56) also referred as *Voronoi tessellation*, is a special kind of decomposition of a metric space determined by the distances of the points of the spaces to a specified discrete set of objects in the space, i.e. the *sites*.

Let us denote the Euclidean distance between two points p and q by *dist(p,q)*.

Definition 1 *Let $S = \{s_1, s_2, ..., s_n\}$ be a set of n distinct points in the plane, i.e.* the sites. *The $VoronoiDiagram$ of S is a partition of the plane into n cells, one for each site in S, such that the point q belongs to the cell corresponding to a site s_i if and only if $dist(q, s_i) < dist(q, s_j) \forall s_j \in S, i \neq j$.*

In the following, we will denote the Voronoi Diagram of S by $Vor(S)$ and the cell corresponding to a site s_i by $V(s_i)$. The left side of Fig. 25 shows the Voronoi Tessellation defined by the set of sites represented by black dots. Each colored region represents $V(s_i)$, where s_i is the site corresponding to the black dot belonging to the region.

In a *Voronoi based* DVE model, the sites correspond to the peers and the DVE is partitioned among the peers according to the Voronoi tessellation.

This decomposition is motivated by the following reasons

- this decomposition naturally defines an assignment of the *passive objects* to the peers. Each passive object O is assigned to the peer P such that O is located in $V(P)$. The management of passive objects will be discussed in chapter 5.
- the dual structure of the *Voronoi Tessellation*, i.e. the *Delaunay Triangulation* is exploited to define the overlay $P2P$ network.

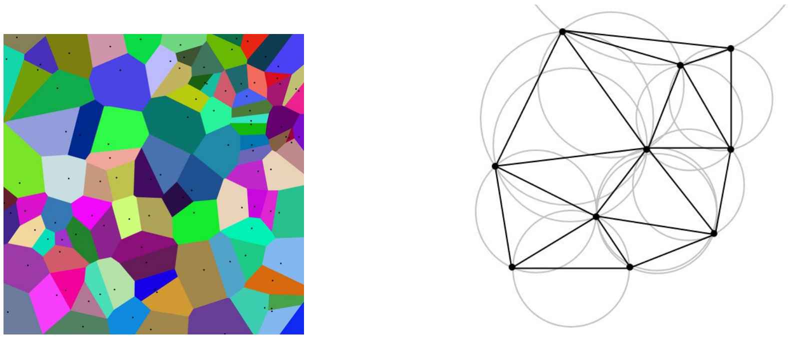


Figure 25: A Voronoi diagram and a Delaunay Triangulation

A *Delaunay Triangulation* is a mathematical structure dual with respect to the *Voronoi Tessellation*.

Definition 2 *A* Delaunay triangulation Dt(P) *for a set P of sites in the plane is a triangulation, i.e. a partition of the plane into a set of triangles, such that the circumcircle of any triangle in* Dt(P) *is empty, i.e. it does not include any other point in P.*

The right side of Figure 25, shows the delaunay triangulation for a given set of points and the circumcircles corresponding to the triangles.

Given a set of n sites $S = \{s_1, s_2, ..., s_n\}$ of the plane, the *Delaunay triangulation* is the *dual structure* of the Voronoi diagram, where the sites correspond to the vertexes of the triangles, and an edge of a triangle connects two vertexes s_1, s_2 if and only if $V(s_1)$ and $V(s_2)$ share a common edge, i.e. s_1 and s_2 are Voronoi neighbors.

Figure 26 shows a *Delaunay Triangulation* on the top of s *Voronoi diagram*, where the borders of the Voronoi regions are shown by dotted lines and the corresponding *Delaunay Triangulation* links are shown by continuous lines.

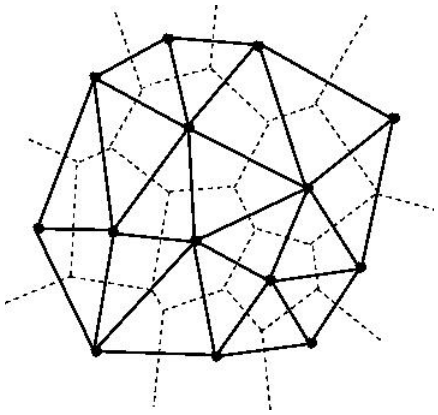


Figure 26: A Delaunay triangulation on top of a Voronoi diagram

A *Delaunay Triangulation* is characterized by several interesting properties.

- *Local equiangularity* for any pair of triangles whose union is a convex quadrilateral, the replacement of their common edge by the alternative diagonal does not increase the minimum of the six interior angles of the triangles.
- No *Delaunay Triangulation* exists for a set of points on the same line
- The *Delaunay Triangulation* defined on a set of sites S=$\{s_1, ..., s_i, ...s_n\}$ always includes the *convex hull* of S.

- For 4 points on the same circle (e.g., the vertexes of a rectangle) the Delaunay triangulation is not unique: clearly, the two possible triangulations that split the quadrangle into two triangles satisfy the Delaunay condition.

- *Convex hull:* The union of all the triangles of the triangulation is the convex hull of the points, i.e. the exterior face of the Delaunay triangulation is the convex hull of the set of points.

4.3.2 Edge Flipping

Edge Flipping is an incremental construction algorithm for *Delaunay graph* proposed by (69). The algorithm can be exploited to define an incremental construction procedure for *Delaunay Networks*.

Given a *Delaunay Triangulation* D, a new site s is added to D by inserting it in a triangle T of D and by linking s to the three vertexes of T. Then the *flipping procedure* is exploited to correct triangles which do not satisfy the empty circle property.

Given a set of sites S=$\{s_1, ...s_n\}$, the algorithm starts from any triangulation T of S and then locally optimizes each edge in order to obtain a *Delaunay Triangulation*. Let e be an internal (non convex hull) edge of T and Q_e be the quadrilateral formed by the triangles sharing e. Q_e is reversed by flipping its diagonal if the two angles without the diagonal sum to more than $180°$ or equivalently if the circumcircles of the two triangles contain the opposite vertex. If Q_e is reversed, it is *flipped* by exchanging e for the other diagonal.

Figure 27 shows an example of the edge flipping procedure.

In (i) the new site p_i is inserted into the triangle sqr. In (ii) the new edge connecting p_i to q, r and s has been added, and the edge $\overline{qr}$ of the quadrilater p_i q, r t has already been flipped. Two more flips are necessary before the final state shown in (iii) is reached.

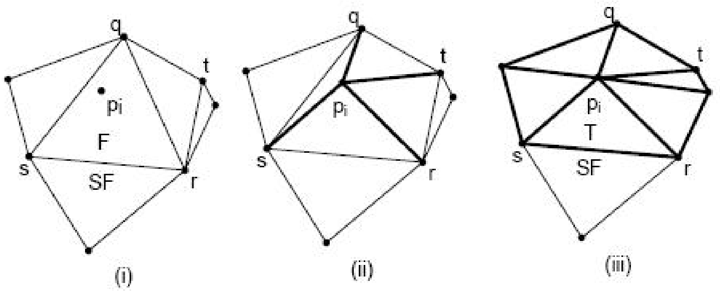


Figure 27: The Edge Flipping Procedure

4.4 Compass Routing on Delaunay Networks

As discussed in Section 4.2, a solution based on the routing of any event generated by a peer P to all the peers located in its AOI may be feasible when crowding does not occur. In a crowding scenario, direct connections between a peer and the other ones in its *AOI* should be preferred. Anyway, the number of direct connections may be too large for low bandwidth peer and an intermediate solution exploiting direct links with closer peers and routing to reach the farthest ones may be required.

Therefore, the definition of a proper *AOI-cast* mechanism defining a routing strategy to spread message within the *AOI* is *mandatory*. The definition of an efficient *AOI-cast* mechanism is fundamental in our case, especially un case of *crowding*. For instance. a solution based on *flooding* is not practical, because of the large amount of messages exchanged through the overlay.

This section describes *compass routing* (59)(60), a routing algorithm which exploits the properties of *Delaunay Triangulations* to minimize the information required at each routing step.

Compass routing has been introduced in (60) for generic geometric graphs and is based upon the following observation. Consider a connected graph G and assume of being located at a node n of G with the goal to reach a destination node d. (60) shows that the best strategy looks

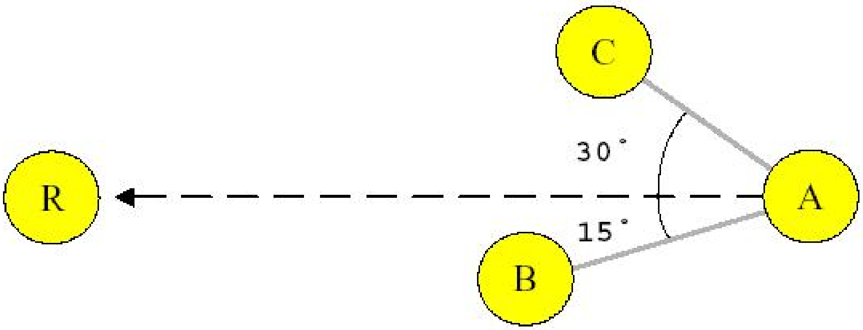


Figure 28: Compass Routing.

at the edges incident in n and chooses the edge whose slope is minimal with respect to the segment connecting n and d. Consider, for instance, Figure 28 and suppose to be located at A with target R. The best way to reach the target R is to pass through B, because the angle $\angle RAB$ is smaller than the $\angle RAC$.

(60) shows that while *compass routing* is not cycle free for general graphs, it can always find a finite path between two nodes of a *Delaunay Triangulation*.

The original formulation of *Compass Routing* makes it possible to discover a path from a node n *toward* a root node r. (69) suggests to exploit compass routing to implement a *multicast* routing algorithm on a Delaunay based Overlay. The basic idea is to define a *Spanning Tree* rooted at any node R of the overlay and spanning the Delaunay links by reversing the path which compass routing computes from any node to R. In our case, the root R of the multicast tree is the node which generates the heartbeat and the tree includes all the peers belonging to the *AOI* of P. It is worth noticing that no algorithm for the construction of the multi-

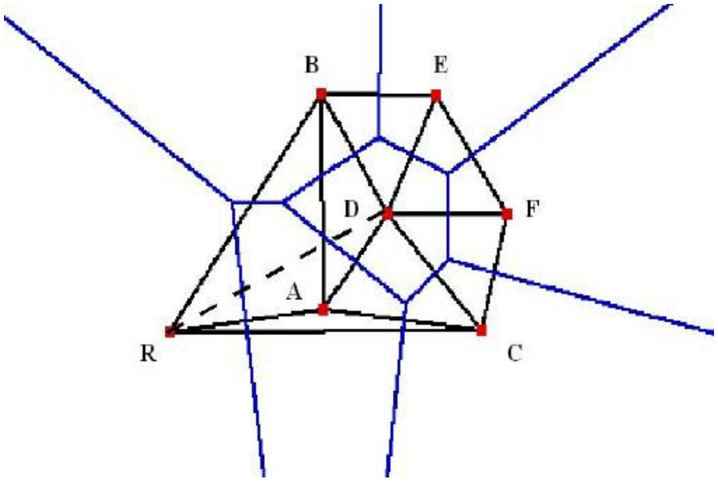


Figure 29: Compass Routing: Computation of the Spanning Tree

cast spanning tree is defined in (69). In this section we present a novel algorithm, based on compass routing, whose goal is the definition of a multicast spanning tree on Delaunay Networks. The algorithm dynamically builds the spanning tree, starting from the root and will be exploited to define a proper *AOI-cast* strategy to propagate events generated by a peer P to its *AOI-neighbors*.

A simple algorithm, based upon the original definition of compass routing, requires a node n to know, not only its Voronoi neighbors, but also those that are distant two Voronoi hops, i.e. the neighbors of its Voronoi neighbors. As a matter of fact, since n is the parent of a node v in the tree iff v chooses n as its parent by compass routing, n must be aware of the position of all the neighbors of v to compute the reverse path. The problem of building a spanning tree in a fully distributed P2P environment, is mainly due to the limited information available at each node. As a matter of fact, each node cannot make any assumption about the structure of the entire overlay and may rely on the knowledge of its *Voronoi neighbors* only.

This simple solution can be improved by considering a basic property of the *Delaunay triangulation*. We recall that any *Delaunay Triangulation* satisfy the *empty circle condition* which states that the circumcircle of each triangle belonging to the triangulation is empty, i.e. it does not contain vertexes besides those that define it. Consider now the *Voronoi Diagram* and the corresponding *Delaunay Triangulation* in Fig.29. Let us suppose that R is the peer generating the heartbeat, i.e. the root of the spanning tree and let us consider node A which receives the heartbeat directly from the root. A should decide whether it is the parent of node D in the spanning tree. A may apply *compass routing* by considering only the triangles ABD and ACD and by comparing only the slopes of the edges AD, BD, and DC with respect to the segment RD. As a matter of fact, the empty circle property guarantees that other edges incident in D cannot intersect these triangles, hence their slope with respect to the segment RD is larger. Hence, A is the parent of D iff the angle $\angle ADR$ is smaller than $\angle BDR$ and $\angle CDR$. Therefore our approach requires that each node knows its own coordinates, the coordinates of its Voronoi neighbors and those of the root of the spanning tree to determine its children in the tree. This minimizes the amount of information to implement *compass routing* and the number of messages exchanged through the overlay. For this reason, our approach is more scalable with respect to (61) that requires a larger amount of information. In Sect. 4.4.2 we will introduce a novel algorithm based on this approach.

4.4.1 Compass Routing in Constrained Regions

In the previous sections we have described a compass routing based strategy to compute a spanning tree covering all the nodes belonging to a given *Delaunay graph*. On the other hand, since our goal is the definition of an *AOI cast* mechanism, we should define a spanning tree including a subset of the *Delaunay nodes* of the DVE, i.e. the nodes corresponding to the peers belonging to the AOI of a peer. In the following we will use the term node and peer as synonymous.

Let us now consider R a $2D$ region including a set S of n sites and let $Dt(S)$ be a *Delaunay Triangulation* of S. Let us consider a subregion C of R. We consider the graph $G(C)$including the nodes of the $Dt(W)$ belonging to C and the subset of Delaunay links including only the links of $Dt(S)$ such that their end points both belong to C.

Note that compass routing should consider a set of nodes located outside $G(C)$ to define a spanning tree rooted at one of the nodes in C and covering all the nodes in C when:

- $G(C)$ is not connected
- $G(C)$ is not a Delaunay triangulation

Consider, for instance, Fig.30, where C is a rectangular region is defined upon a *Delaunay Triangulation*. Let us suppose that node a corresponds to a peer and that the rectangular region corresponds to the *Area of Interest* of the peer. No path between a and c including only nodes in the *G(C)* does exist. Therefore, compass routing should compute a path from a to c including node b which is located outside *G(C)*.

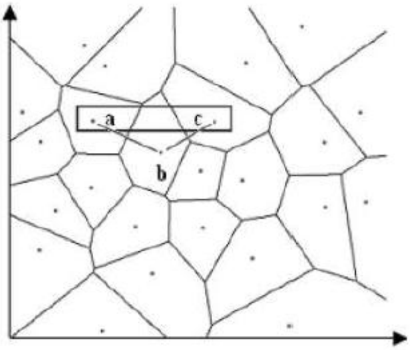


Figure 30: A Rectangular Region

Consider now Fig. 31 which shows a circular region C centered at the peer A. This region represents the *AOI* of A. $G(C)$ includes the nodes A, B, D, E, F, but not the node C. Hence the Delaunay edges represented

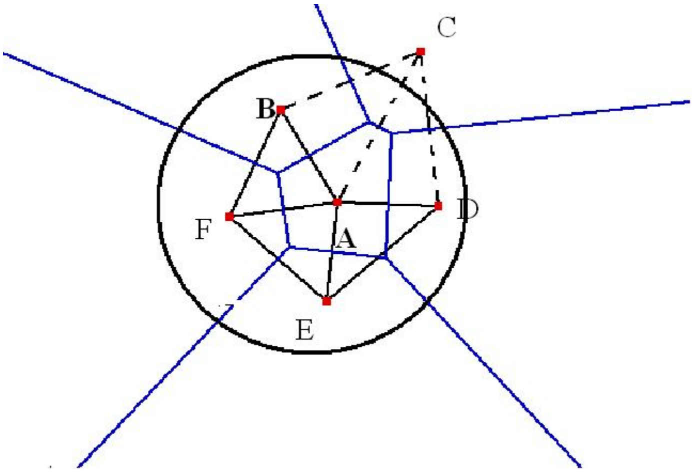


Figure 31: An AOI-graph

by dashed lines do not belong to $G(C)$, since one of their end points, i.e. C, does not belong to C. $G(C)$ is not a *Delaunay Triangulation* because it does not include the *convex hull* of its nodes.

In our case, it is interesting to evaluate the number of external Delaunay links which must be evaluated, because each of these links implies a routing hops so introducing a further latency in the delivery of an heartbeat.

The following results show that a spanning tree including all and only the nodes of the AOI may be computed by compass routing when the AOI has a *circular shape*. Hence, no hop outside the AOI is required in this case and latency is not increased.

We will first show that the graph $G(C)$ corresponding to a *circular region* is always connected.

Theorem 1 *Let $Dt(R)$ be a Delaunay triangulation defined on a set of nodes belonging to the 2-dimensional space R. If C is circular shaped subregion of R, then $G(C)$ is connected.*

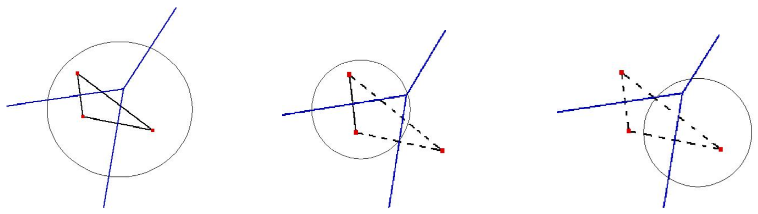


Figure 32: A Delaunay Triangulation Including a Single Triangle

Proof

We prove this property by induction. As base of the induction, consider a *Delaunay Triangulation* including a single triangle T such that $T \bigcap C \neq 0$. In this case, either $T \subset C$, or a single vertex/edge of $T \in C$. In both cases, $G(C)$ is connected, as shown in Fig.32.

First we show that *Edge Flipping* may not replace a link of $Dt(R)$ whose end points both belongs to C by another one whose end points are both located outside C. The proof is given by contradiction. Consider Fig. 33 and suppose that the edge(s_i,s_j) whose end points are both located in C is replaced by the link (s_k, s_t) whose end points are both located outside C. The secant $\overline{s_k, s_t}$ partitions C into two circle segments, C_i, which includes s_i and C_j, which includes s_j. Let us consider the circumcircle $Circ$ of the triangle s_k, s_i s_t. The intersection points between C and $Circ$ are A and B and, since two circumferences may intersect at most at two points, $C_j \subset Circ$. Hence $s_j \in Circ$ and the *empty circle property* does not hold which brings to the contradiction.

Let us now consider a Delaunay triangulation $Dt(R)$ including n triangles such that $G(C)$ is connected. Let us insert a further node P inside the triangle T of $Dt(R)$ and consider the Delaunay triangulation $Dt'(R)$ obtained by applying the *Edge Flipping procedure* to the new triangles obtained by connecting P to the vertexes of T. We have to prove that $G'(C)$, i.e. the graph including the sites of $Dt'(R)$ belonging to C, is connected.

We consider three cases. If $P \notin C$, the edge flipping procedure may flip some side of new triangles defined inside T, but previous argument shows that no links whose end points both belong to C is flipped by one

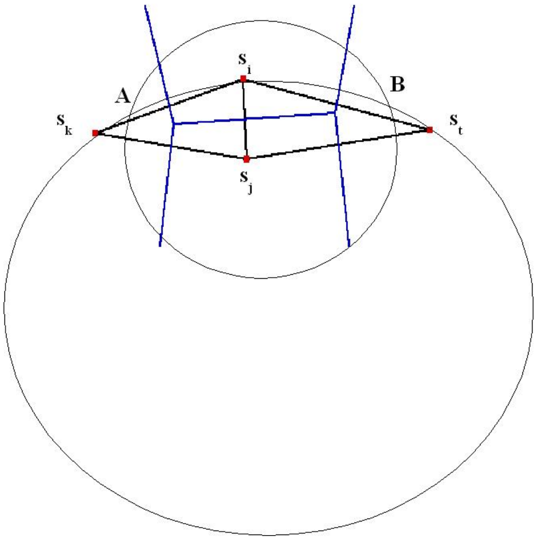


Figure 33: Intersection Between AOI and the Circumcircle of a Triangle

whose end points are both outside C: In the second case, $p \in C$ and T has at least an end point $n \in C$, then p is connected to n and this link cannot be replaced by edge flipping. The last case, which is shown in Fig.34, no vertex of $T \in C$. Since P is connected to the vertexes of T which are located outside C, no connection is initially defined between P and other vertexes belonging to C. In this case the union of the circumcircles of the new triangles defined within T and having a vertex in P cover C. Consider, for instance, Fig.34. P is the new vertex included in the triangle ABC. It is easy to prove that the circumcircles of the triangles APB, PAC, PBC cover C. Hence, if C includes at least a further vertex, the links connecting P to the vertexes of T will be flipped until P is connected to a vertex inside C. In conclusion, if the link n_i, n_j of $G(C)$ is flipped the new link will have at least one end point $e \in C$ and n_i, n_j

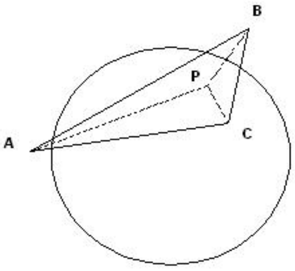


Figure 34: Inserting a Node in a triangle whose vertexes are external to the circumference

will be connected to e. This implies that any path passing through n_i, n_j in $G(C)$ may be replaced by the path n_i, e, n_j in $G'(C)$, hence $G'(C)$ is connected. $\diamond$

This result guarantees that it is possible to define a spanning tree covering *all and only* the nodes belonging to a circular shaped *AOI*. On the other way, it is still possible that the spanning tree computed by *compass routing* is not minimal, i.e. it includes some nodes located outside the AOI. Let us consider, for instance the rectangular *AOI* R shown in Fig. 30. While $G(R)$ is connected, compass routing chooses the node a which is located outside the AOI as parent of node b.

The following theorem shows that if we consider a circular region R centered at the node n, then compass routing may compute a spanning tree rooted at n and including all and only the nodes belonging to R.

Theorem 2 *Let $Dt(S)$ be a* Delaunay Triangulation *defined by a set S of sites belonging to a $2D$ space W. If R is a circular subregion of W centered at the node $s \in S$, compass routing is able to compute a spanning tree rooted at s and including* all and only the sites *of $S \in R$.*

Proof: (60) shows that compass routing decreases at each step the distance to the target node. Since the construction of the spanning tree reverses any path computed by compass routing, the distance from s increases at each step. Consider a site $p \notin R$. Since the distance of p

from the root s is larger than the radius of R, p cannot be the parent of a peer $q \in R$, otherwise the distance from the root s should decrease when passing from p to q. $\diamond$

On the other hand, the theorem may be not valid for different shaped area of interest. As shown in (58), if rectangular or squared areas are considered, some path of the spanning tree may zig zag around the borders of the considered region.

Previous results guarantee that the algorithm introduced in Sect.4.4 is valid when considering a circular region centered at the peer. Furthermore, the last theorem suggests that any peer belonging to the AOI of a peer P should consider, in the angle evaluation phase of the spanning tree construction, its *Voronoi neighbors* belonging to the AOI of P only. As a matter of fact, peers located outside the AOI cannot belong to the spanning tree and should not be considered.

4.4.2 A Distributed Algorithm for the Spanning Tree Construction

This section defines a distributed algorithm to compute a spanning tree within a circular region R which is a subset of a $2d$ region. The algorithm exploits the properties proved in the previous sections.

The AOI-cast strategy defined on the P2P overlay which enables a peer of the DVE to propagate an heartbeat within its AOI is based on this algorithm. This strategy will be presented in Section 4.5.

Let us suppose that a spanning tree rooted at r has to be computed. r first sends to its Voronoi neighbors through the Delaunay links a message including its coordinates. Each neighbor receiving the message should determine its children in the spanning tree with respect to r and forward them the message received form r. The procedure is recursively executed by each node receiving the message until the borders of R are reached.

Our goal is to define an algorithm to determine the children of a node in the spanning tree which requires a minimal knowledge of the structure of the overlay, i.e. the knowledge of the Voronoi neighbors of a given node only. Furthermore, the algorithm requires a minimal set of functionality to manage the Voronoi tessellation. In our case, we only require the existence of a support able to return the Voronoi neighbors of a given node. This function is present in any package supporting Voronoi Diagrams, while more complex functionality, for instance functions for the management of Delaunay triangulations are not always available.

The algorithm requires the sequential execution of the following steps:

- *Neighbors Sorting*
- *Children Detection*

In the *Neighbors Sorting* phase, n sorts its *Voronoi neighbors* according to a counter-clockwise ordering. This ordering is then exploited in the *Children Detection* phase when each neighbor v of n is considered to detect if it is a child of n in the spanning tree. This phase requires that the vertexes of the Delaunay triangles sharing the link l connecting n to v are detected. The nodes s and p corresponding to these vertexes are both neighbor of n and of v and should be considered in the angles evaluation phase. As a matter of fact, the compass based construction of the spanning tree introduced in Section 4.4 computes the vectors connecting v to n, s, p and compares the angle between each of these vector and the straight line connecting v to the root. The node corresponding to the smallest angle is the father of v in the spanning tree.

If l is an internal link of the Delaunay triangulation, i.e. if l do not belong to the convex hull or the nodes, p, r.s. s is the node that precedes, r.s. follows v in the counter clockwise ordering.

Consider, for instance, Fig.35 where the neighbors of node A are numbered according to the counter clockwise ordering, starting from the smallest one which is A_1. To detect if A_4 is a child of A, nodes A_1, rs. A_3, i.e. the predecessor, r.s. the successor of A_4 in the counter clockwise ordering are considered.

This procedure is not valid when l belongs to the convex hull of the set of nodes since l is an external link of the triangulation and it belongs to a single triangle. In this case a single node, i.e.the predecessor,r.s. the successor of v in the counter-clockwise ordering should be considered.

Consider for instance the Delaunay link l between nodes B and node B_1 in Fig.35. Since this link belongs to the convex hull, B should consider only node B_5 when deciding whether B_1 is its child. Note also that B_2 is not a Voronoi neighbor of B_1, even if it follows B_1 in the counter-clockwise ordering.

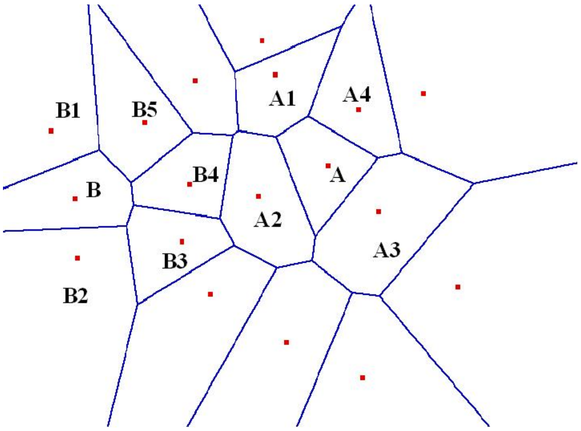


Figure 35: Sorting the Voronoi Neighbors

As a consequence, a triangle defined by n and a pair of consecutive nodes p and s in the counter clockwise ordering of its neighbor may not belongs to the Delaunay triangulation. Following conditions may be exploited to detect if the triangle pns does not belong to the Delaunay triangulation.

- the straight line which connects p and s intersects at least one of the links connecting n and the predecessor of p or n and the successor of s. This implies that the line connecting p and s cannot be a *Delaunay edge*.

- the triangle defined by vertexes p, n and s includes at least a node between the predecessor of p and the successor of t. This implies that the triangle does not belong to the Delaunay triangulation.

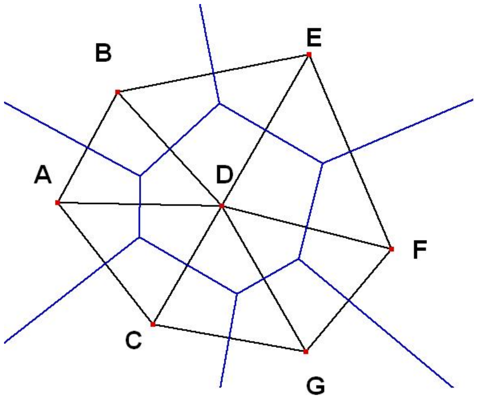


Figure 36: Delaunay Triangle Test: First Condition

Figure 36 shows that the line from node B to node C intersects at least a link from A to a a node that is predecessor of B and successor of A. Note that B and C are consecutive in the counter-clockwise ordering, but they are not Voronoi neighbors, because their Voronoi are intersects the border of the 2d region.

Figure 37 shows that the triangle BAC includes the node E and hence cannot be a Delaunay triangle.

It is worth noticing that a simple test checking if a pair of neighbor of n are Voronoi neighbors themselves cannot replace previous conditions. Consider for instance Fig.38. Even if node B and D are Voronoi neighbor, A should consider only node C when evaluating if node D is its child in the tree. Note also that this implies that two neighbors of a node n which are neighbors themselves are not necessarily contiguous in the counter-clockwise ordering.

Let us now define the algorithms exploited to implement the *Neighbors Sorting*, and the *Children Detection* phase.

To define the counter clockwise ordering, we define in the 2d space a coordinate system whose origin is at n with unit vectors $\overrightarrow{i}$ and $\overrightarrow{j}$. A counter-clockwise ordering of the neighbors of n is defined by consid-

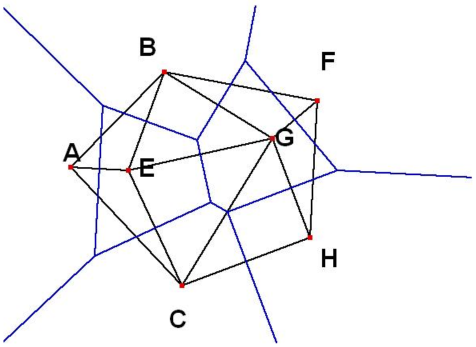


Figure 37: Delaunay Triangle Test: Second Condition

ering the angles between $\overrightarrow{j}$ and the vector connecting n to each of its Voronoi neighbors v. We consider the convex angle a between n and v if the x-coordinate of v is negative, otherwise the angle obtained by adding up π to the supplementary of a is considered.

The function $compareneighbors(n, a, b)$, shown in Fig.39, takes as input the coordinates of a pair of neighbors, a and b, of node n and returns the smallest one with respect to the ordering. The coordinates are referred to the coordinate system whose origin is at n.

The algorithm first checks if the neighbors have opposite x-coordinates and, in this case, the neighbor with the negative x-coordinate is the smallest one. Otherwise, the convex angles α, r.s. β between $\overrightarrow{j}$ and $\overrightarrow{na}$, r.s. $\overrightarrow{j}$ and $\overrightarrow{nb}$ are computed. If the x-coordinate of both a and b are negative the smallest neighbor is the one corresponding to the smallest angle, the other way round if both a and b have a positive x-coordinate.

The *Children Detection* phase determines the children of a node n in the spanning tree. As shown in Section 4.4, *compass routing* chooses the next node toward a target node d by detecting the neighbor with the minimal slope with respect to the straight line connecting to n to d. As shown in Section 4.4, the spanning tree may be computed by reversing this procedure.

The function $SpanningTreeChildren(r, n, i)$ described in Fig. 40 im-

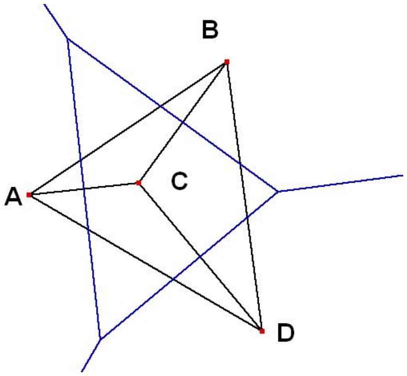


Figure 38: Delaunay Links and Voronoi Neighbors

plements the Children Detection phase. The function takes as input the coordinates of the root of the spanning tree, those of n and the index i, of the i-th Voronoi neighbor of n according to the counter-clockwise ordering of the neighbors. The function returns $true$ iff the i-th Voronoi Neighbor of n is a child of n in the spanning tree rooted at r.

Since (69) the distance from the target decreases at each step of compass routing, in our case the distance from the root should increase at each step, since compass routing is reversed. For this reason $SpanningTreeChildren\ (r, n, i)$ first checks if v_i is farthest from the root of the spanning tree with respect to n. If this is not true, v_i cannot be a children of n and the function returns a false value, otherwise the function executes the angle evaluation phase. The function $Delaunay\_triangle(n, v_i, v_j)$, where v_j is the predecessor or the successor of v_i in the counter-clockwise ordering, checks if the triangle defined by the three nodes belongs to the Delaunay triangulation. This check exploits the functions previously defined. For each valid triangle, the function executes the angle evaluation phase.

Figure 41 shows the results of the angle evaluation phase. In the figure the root R sends a message to all its neighbors. Among these, n receives the message and decides that v_3 is one of its child in the spanning

```
$compareneighbours(n, a, b)$ :
    **if** $(a_x \times b_x) < 0$
        **if** $a_x < 0$ **return** $a$ **else return** $b$
    **else**
        $\alpha = \arccos(a_y)/\sqrt{a_x^2 + a_y^2}$
        $\beta = \arccos(b_y)/\sqrt{b_x^2 + b_y^2}$
        **if** $a_x < 0$ **and** $b_x < 0$
            **if** $\alpha < \beta$ **return** $a$ **else return** $b$
        **else**
            **if** $\alpha < \beta$ **return** $b$ **else return** $a$
```

Figure 39: Neighbors Comparison

tree, because $\angle nv_3r$ is smaller of both $\angle v_2v_3r$ and $\angle v_4v_3r$.

The construction of the spanning tree stops when a node cannot find a child in the tree among its neighbors.

4.5 A P2P Overlay for Voronoi Based DVE

This thesis exploits a *Delaunay Triangulations* to define a $P2P$ overlay for Distributed Virtual Environments. According to this approach, the position of each peer is exploited to define a *Voronoi tessellation* of the DVE. Given *n sites* corresponding to the peers, a *Voronoi tessellation* partitions the virtual world into a set of n regions such that the region corresponding to a site s includes all the points of the DVE which are *nearer* to s with respect to any other site. The $P2P$ overlay is defined by the *Delaunay links* belonging to the *Delaunay Triangulation* corresponding to the *Voronoi tessellation*.

The choice of a Delaunay based overlay has several advantages:

- *Mapping of Passive Objects to the Peers*: Since each point of the DVE is mapped to a single *Voronoi region*, a straightforward *mapping of passive objects to the peers* may be defined. This mapping assigns each passive object to the peer which manages the Voroni region where the object is located.

```
SpanningTreeChildren(r,n,i):
    if $dist(v_i, r) < dist(n, r)$ return false
    else
        if $Delaunay\_Triangle(n, v_i, v_{i+1})$ and $Delaunay\_Triangle(n, v_i, v_{i-1})$
            if $\angle nv_ir < \angle v_{i+1}v_ir$ and $\angle nv_ir < \angle v_{i-1}v_ir$
                return true
            else return false
        else
            if $Delaunay\_Triangle(n, v_i, v_{i+1})$
                if $\angle nv_ir < \angle v_{i+1}v_ir$ return true
                else return false
            else if $Delaunay\_Triangle(n, v_i, v_{i-1})$
                if $\angle nv_ir < \angle v_{i-1}v_ir$ return true
                else return false
```

Figure 40: Angle Evaluation Phase

- *Bounded Connections*: Since, as shown in Fig.50, each site of a Voronoi tessellation has 6 neighbor node on the average, each peer should manage a limited number of connections with other peers, i.e. those corresponding to the Delaunay links.

- *Overlay Connectivity*: The connections corresponding to the Delaunay links guarantee that the overlay is connected. Even if a peer is located in a unhinabited region of the virtual world, it keeps in touch with the rest of the DVE through the connections with its Voronoi neighbors.

Note that the definition of a distributed algorithm for definition of the Voronoi overlay is a real challenge for a dynamic real time environment, like a *MMOG*, where the positions of the peers change continuously and no centralized coordination entity does exist.

As shown in Sect.4.2, further connections between a peer and other peers located in its AOI should be defined to guarantee scalability, especially in crowding scenarios. As discussed, the AOI structure may be implemented according to two different approaches. The first one is based on the definition of a set of *direct links* between a peer and any other one in its AOI. An alternative approach is that which includes in the

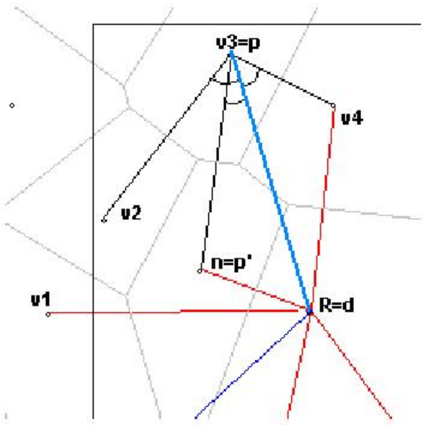


Figure 41: Angle Evaluation Phase

$P2P$ overlay Delaunay links only and defines an *AOI cast* mechanism to spread any event to all the peers in the AOI through the Delaunay links.

This thesis propose an *intermediate solution*, which defines direct connections between a peer P and *a subset* of the peers in its *AOI*, i.e. the peers nearer to P, and exploits *AOI-cast* to propagate an heartbeat to any other peer in the *AOI*. An approach similar mechanism is exploited to propagate further events, for instance updates of the state of the passive objects or events related to actions performed by the peers. These issues will be discussed in more details in Chapter 5.

According to our approach, the overlay includes a set of connections between a peer and its Voronoi neighbors and a set of further connections between a peer and a subset of the peers located in its *Area of Interest*.

Note that some *Voronoi neighbors* of a peer may not belong to its *Area of Interest*, but these connections are required to guarantee that the overlay is connected. Note also that connections among peers are based on their spatial relationship in the DVE, rather than on their physical network proximity.

Figure42 shows the *AOI* of the peer A. The overlay includes direct

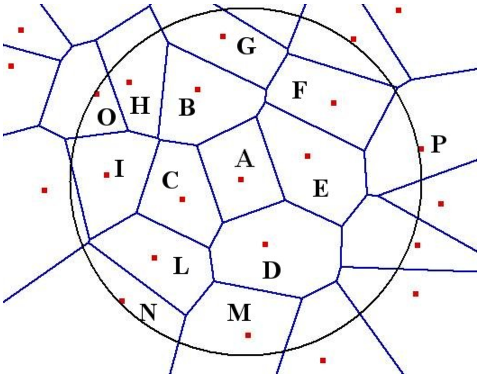


Figure 42: Area of Interest

links between A and its *Voronoi neighbors* B, C, D, E. Direct links between A and other peers in its AOI, like peer F, G, H, I, L, M , N and O may be defined according to the adopted forwarding model. No link is defined between A and P, because P neither belongs to the AOI of A nor it is a *Voronoi neighbor* of A.

In the following, we will consider circular areas centered at P. In this case all the properties introduced in the previous sections may be exploited. In any case, our approach can be generalized to different shaped areas.

According to our approach, the *Area of Interest* of each peer P is partitioned into two areas, the *Internal Area of Interest* of P, $IAOI(P)$ and the *Peripheral Area of Interests*, $PAOI(P)$, $PAOI(P) = AOI(P) - IAOI(P)$. The $P2P$ overlay includes, besides the *Voronoi links*, a link between P and any peer located in $IAOI(P)$. Each heartbeat is periodically sent by P through these links, afterward it is forwarded to any peer located in $PAOI(P)$.

Figure 43 shows the $IAOI$ of a peer a which is delimited by the internal circumference, while its $PAOI$ overlaps the *annulus* between the circumferences. The corresponding overlay includes direct links between a, and r.s. b,c,d,e,l, while *forwarding* is exploited to reach g, f and h. No-

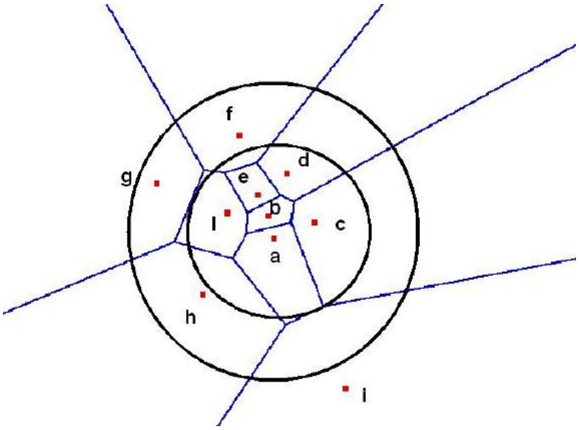


Figure 43: Internal and Peripheral Areas of Interest.

tice that since a and e are not *Voronoi neighbors*, the direct link connecting them is not a *Voronoi link*. Furthermore, it is worth noticing that a should send its heartbeats to i as well, even if it is located outside its area of interest because any *Voronoi Neighbor* act as 'beacon node' that inform a about peers approaching from farthest positions. Network connectivity is guaranteed by these nodes, *even when the AOI of a peer is empty*.

In the following, we will assume that the width of the annulus of $AOI(P)$ is larger than 0, i.e. the border of $IAOI(P)$ and of $PAOI(P)$ do not overlap. Furthermore, this width is chosen so that any peer Q entering $AOI(P)$ sends an heartbeat when it is located in $PAOI(P)$, before entering $IAOI(P)$. To compute the *minimal width* of the annulus which guarantees the previous condition, both the *maximum speed* of a peer and the *frequency of heartbeat notifications* should be considered.

The peer P which generates an heartbeat sends it to all its Voronoi neighbors and to any peer in its $IAOI$. The forwarding of the heartbeat is started by the peers located 'at the border' of its $IAOI$, i.e. the *boundary peers*, then, any peer located in $PAOI(P)$ carries on the forwarding. The forwarding phase exploits the *routing algorithm* based upon the construction of a *spanning tree* rooted at P which has been introduced in Sect

4.4.

The most challenging issue of our approach is the definition of proper *distributed algorithms* to guarantee that the structure of the Delaunay overlay is correctly preserved and no overlay partition occurs. We propose a *'pass the word'* approach, where peers becomes acquainted with each other through their Voronoi neighbors. A peer receiving an heartbeat from one of its neighbors N checks if any of its further neighbors Q, $Q \neq N$, is entering $AOI(N)$, and in this case it propagates N heartbeat to Q. In this way, each peer acts as a *beacon* for its neighbors by 'putting in touch' peers that are not acquainted with each other. A similar approach is adopted to acquire new objects located beyond the *AOI* of a peer.

In order to avoid network partitioning, a proper mechanism to guarantee that each peer correctly keeps its Voronoi neighbors is required. We can imagine two different scenarios. In a crowding scenario, a peer Q should become aware of another peer P when entering the area of Interest of P. As a matter of fact, several peers are located in $AOI(P)$ and these should notify Q any heartbeat produced by P. In the opposite scenario, $AOI(P)$ is empty. In this case, a new peer becoming a Voronoi neighbor of P should be notified of P by one of the previous Voronoi neighbors of P. To implement this, we assign a TTL to each heartbeat and we guarantee that each heartbeat is propagated at lest k hops away from its source. This should guarantee that a peer is notified about the presence of another peer before keeping in touch with it.

We pair a TTL with each heartbeat notification in order to guarantee that each heartbeat generated by a peer P is propagated at least n hops away from P.

Fig. 44 shows a specification of the distributed routing algorithm defined through the *Mobile Unity* formalism introduced in Sect.3. We suppose that each peer $P(i)$ is uniquely identified by the value of its index i. According to the *MobileUnity* formalism, a variable λ is paired with each peer $P(i)$ describing its current location. This variable is exploited to define *location aware interactions* among the peers.

Each $P(i)$ defines an array HB whose size equals the maximum number of peers in the system. $HB(i)\,[k]$ stores the most recent *heartbeat* no-

```
**System** *VoronoiOverlay*
**program** $P(i)$ **at** $\lambda$
    **declare**
        $\Box$ *HB*:array $[0 \ldots MaxPeer - 1]$ of
                    $(Pos : \lambda, TSS : integer, TSR : integer, TTL : integer)$
        $\Box$ *TSS,time,Gtime:integer*
    **initially**
        $\Box$ $\lambda$= *PeerLocation(i)*
        $\Box$ *time,Gtime,TSS* = 0
    **assign**
        $\Box$ *ClockTick::time :=time+1*
        $\Box$ *SendHB::* $\lambda, TTL, TSS, time, Gtime$:=
                                $NewLoc(\lambda), MaxTTL, Gtime, 0, Gtime + time$
                    **reacts-to** $time > HB\_frequency$
        $\Box$ *ResetHB:: HB* $[k]$:=$\bot, \bot, \bot, \bot$
                    **reacts-to** $Gtime - HB\,[k]\,.TSR > MaxDelay$
    **end**
**Components**
        $\Box$ i:$0 \le$i$<$*MaxPeer*:: *P(i)*
**Interactions**
        $\Box$ $P(i).HB\,[j] := (P(j).\lambda, P(j).TSS, Gtime, P(j).TTL - 1)$
                **reacts-to** $P(i).HB\,[j]\,.TSS \neq P(j).TSS$
                                        $\wedge$
                    $(P(i).\lambda \in IAOI\ (P(j)) \vee VoronoiNeighs\ (P(i),P(j)))$
        $\Box$ $P(i).HB\,[k]$:=
              $(P(j).HB\,[k]\,, P(j).HB\,[k]\,.TSS, Gtime, P(j).HB\,[k]\,.TTL - 1)$
              **reacts-to**
              $P(i).HB\,[k]\,.TSS \neq P(j).HB\,[k]\,.TSS$,
                                    $\wedge$
                    $VoronoiNeighs(P(i), P(j))$,
                                    $\wedge$
                    $Is\_Parent(P(j), P(i), P(k))$
                                    $\wedge$
        $((P(i).\lambda \in PAOI(P(k)) \vee (P(i).\lambda \notin AOI(P(k)) \wedge (P(j).HB\,[k]\,.TTL > 0))$
```

Figure 44: AOI-cast: A Mobile Unity-Based High Level Specification

tified by peer $P(k)$ to peer $P(i)$. For each heartbeat received from $P(k)$, $P(i)$ stores Pos, i.e. the last position of $P(k)$ notified by the heartbeat, TSS, the *Time Stamp* paired by $P(k)$ to the heartbeat, TSR, the timestamp paired by $P(i)$ to the heartbeat when it has been received, and TTL associated with the heartbeat. TSR is exploited by $P(i)$ to define an *heartbeat eviction* policy which deletes from HB the obsolete heartbeats, i.e. those which have been received more than $MaxDelay$ earlier. This minimizes the network traffic, because a peer which leaves the AOI of $P(i)$ should not notify this to $P(i)$, because its heartbeat will be simply deleted by $P(i)$ when it becomes obsolete.

The *assign* section of the *Mobile Unity* specification defines the behavior of each peer $P(i)$. The *Clock Tick* event corresponds to the periodic increment of the *time*, the variable which periodically updates *Gtime*, the local clock of $P(i)$. This variable is exploited to define the point of time when a new heartbeat should be generated and, in this case, to *fire* the *SendHB* event. The generation of the new heartbeat is modeled by the assignment of the new position of the peer, of the corresponding timestamp and of the TTL to the variables λ, TTS and TTL. This event should fire a set of reactions which propagate the heartbeat to any peer of the AOI. The peers belonging to $IAOI(P(i))$ copy the value of the heartbeat directly from $P(i)$, those belonging to the $PAOI(P(i))$ receives the heartbeat through a sequence of reactions which define the routing process. These reactions are defined in the *Interactions Section*. Note that the variable *time* is reset each time a new heartbeat is generated, while the variable *Gtime* which defines the local time of each peer, is never reset and is exploited to assign the timestamp to each heartbeat which is generated by $P(i)$.

The *ResetHB* event is fired when the interval of time elapsed from an heartbeat reception exceeds a predefined value *MaxDelay*.

The interactions between peers are modeled by two different clauses, defined in the interaction section. The first one describes a *direct interaction* between the peer $P(i)$ which generates the heartbeat and a peer $P(j)$ which is its Voronoi neighbor or belongs to $IAOI(P(i)$. This interaction is modeled as a reaction fired by the generation of a new heartbeat

h by $P(i)$, and is defined by a copy of h in the i-th position of the array HB of $P(j)$. A real implementation should implement this reaction by a message sent from $P(i)$ to $P(j)$

The second clause is fired by the reception from $P(j)$, which is the neighbor of a peer $P(i)$, of a new heartbeat generated by $P(k)$. $P(i)$ realizes that a new heartbeat generated by $P(k)$ has been received by $P(j)$ by comparing the value stored in the k-th position of its vector with that stored in the corresponding position of the vector store by $P(k)$. If the timestamp corresponding to these entries do not correspond, the heartbeat is copied from $P(j)$ to $P(i)$.

$P(j)$ propagates the heartbeat to $P(i)$, if $P(i)$ is both its Voronoi neighbor and its child in the spanning tree rooted at $P(k)$ and one of the following conditions holds:

- Both $P(i)$ belongs to $PAOI(P(k))$
- $P(i)$ is located outside $AOI(P(k))$, but the value of the TTL of the heartbeat is larger than 0.

A peer Q is a *boundary peer* for peer P iff one of its Voronoi neighbors is located in $PAOI(P)$. We exploit $Is\_Parent(P, Q, R)$ to check if Q is a son of P in the spanning tree rooted at P. This function is based on the compass-based spanning tree construction defined in Sect.4.4. It is worth noticing that the spanning tree built by the algorithm includes a set of *'shortcuts'* connecting the root to its boundary peers. The results proved in Sect4.4 are still valid in this case, because any peer belonging to the *PAOI* should be a child of a boundary peer or of another peer in the *PAOI*.

It is worth noticing that the reaction is fired only for peer belonging to the $PAOI$ of the source of the heartbeat. For instance when peer b in Figure 43 receives a heartbeat h from peer a, it does not propagate h to its neighbors because they are all located in $IAOI(a)$ and they receive the heartbeat directly from a.

Note that a peer Q located outside $AOI(P)$ notifies the heartbeat to each peer V which is its Voronoi neighbor that does not belong to

$AOI(P)$, if the timestamp $\neq 0$. Notice that V may not know P because it is approaching P from a further position with respect to that of Q. In this case, Q 'puts in touch' P and V. For instance, a heartbeat generated by P and received by a peer Q located outside $AOI(P)$, is propagated to all the Voronoi neighbors of Q, even to those that are not neighbors of P. In this case, an abrupt crash of Q will not disconnect the network because a larger set of peers knows the position of P. These notifications are critical, because they guarantee network connectivity when the area of interest of a peer is empty. The robustness of the application may be increased by introducing further redundancy.

It is worth noticing that in our approach the heartbeat notification mechanism naturally supports the discovery of new neighbors. Each peer Q approaching P, must first keep in touch with a Voronoi neighbor V of P or with a peer located in $AOI(P)$ (recall the width of the annulus is different from 0). V simply propagates an heartbeat received from P to Q. In this way, Q becomes acquainted with P. Notice that this differs from the approach proposed in (1), where each peer acquires knowledge of new approaching peers by *explicitly querying* its boundary neighbors.

Finally, it is worth noticing that *AOI-cast* can be further optimized by exploiting a set of *aggregation techniques*. As a matter of fact, if Q is the parent of V with respect to a set of spanning trees routed at different peers, it may *aggregate* the heartbeats sent by these peers in a single message which is periodically sent.

It is important to note that in a distributed environment the local views of the virtual world at each peer may be inconsistent, due to network delay, packet loss. For this reason, different notifications of the same heartbeat may reach a peer, while some notification may not reach a peer at all. This issue will be discussed in Sect 4.7.4.

4.6 Avoiding Overlay Partition

Overlay partition is a situation where the overlay graph is divided into two or more subgraph, or partitions, and the nodes belonging to different partitions do not know each other. In this case, each partition corre-

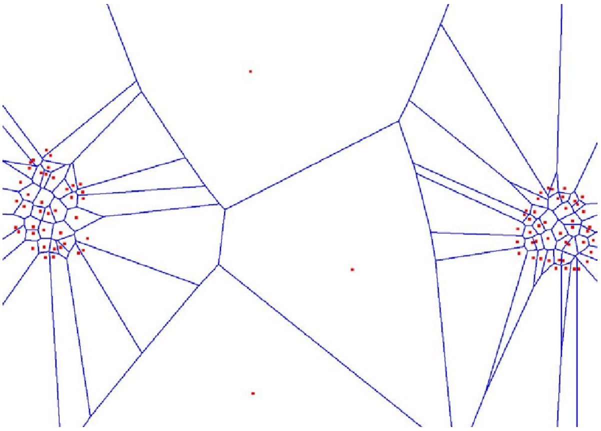


Figure 45: Partition at risk in a Voronoi based overlay.

sponds to an isolated sub world and when a peer P moves close to the peers belonging to this partition, P and these nodes are not able to keep in touch each other.

In a Voronoi based overlay, a partition may occur when nodes are crowded in distinct areas, as shown in Figure 45. In this situation, a few nodes may be located in between the crowded areas. Note that a few nodes are located in the central area of the virtual world shown in the figure. Since the node in the two crowds are able to communicate only through them, if they fail simultaneously a partition does occur.

In general, the distribution of the nodes in a DVE is not uniform, because nodes cluster in some regions where interesting events occur or virtual cities do exist.

This situation may be faced by introducing a certain degree of *redundancy* in the heartbeat transmission protocol. The experimental results shown in the following sections show that this strategy avoid overlay partition in most case.

A node can find out whether its crash may produce an overlay partition, by exploiting only local information. If the node receives heartbeats from peers located outside its AOI, it has a view of the distances that separate itself from the others and can cope with the risk of overlay partitioning by adopting a set of countermeasures.

We can chose for instance *a threshold node distance*, beyond which a countermeasure must be exploited to reduce the risk of overlay partition.

For instance, when a node finds out that its failure can compromise the connectivity of the overlay it can modify the protocol of heartbeat transmission to improve the overlay stability. For instance the node located in the central region of Figure 45 can propagate an heartbeat received from its neighbors some steps farther. In this way the node at risk become a bridge for two or more possible overlay partitions and even if it fail the border nodes of the partitions know each other.

Each heartbeat used to improve the overlay stability could be properly marked to stress the risk of a partition and to inform the receiver that it should pay attention to any delay on messages from the sender.

We can note that, even if this solution increases the number of heartbeats, the node at risk in Figure 45 has a few neighbors, hence they can support a larger amount of traffic. As a matter of fact, this situation is orthogonal to crowding where a node should support a large heartbeatstraffic.

4.7 Experimental Results

4.7.1 Real time Constraints

DVE applications are characterized by severe real-time constraints, since state consistency must be maintained as soon as possible and time thresholds must usually be respected by events dispatching. In our approach, each peer represents its local view of the DVE through a *Voronoi Diagram*. Each time a peer receives an heartbeat from a connections, it updates its Voronoi Diagram. It is possible to reduce the frequency under which each peer performs the updates of its local view, but a frequency too low introduces a high degree of inconsistency in it. For this reasons, *the computation of the local view of the overlay must be very efficient*. As far as concerns the frequency of the event propagation, we define two kind of events. Each peer receives any event occurring within its AOI, both heartbeats and other events generated by actions executed by the peers. While the latter are notified only when they occur, each peer sends an heartbeat to the others peers *with an high temporal frequency, up to 5 times per second*. This implies that, in order to maintain a consistent local view, it must be updated up to some times per second, in the worst case each time a peer receive an heartbeat. The AOI increases the scalability, but as discussed in Sec.4.2 crowding often occur in a DVE, and, in these cases, scalability is at risk even if AOI are exploited.

The definition of an highly efficient support for the management of Voronoi Diagrams is therefore mandatory for our approach.

The Voronoi based overlay has been implemented by a subset of the *VAST Library* (68). This library supports a set of functionalities to support the construction and management of *Voronoi Diagrams*. The functions exploited by our simulations are the insertion/update of a node in the overlay, the discovery of the neighbors of a given node, and the discovery of the node that is the owner of a point of the DVE.

We have tested the performance of different functions of the $VAST$ library in order to understand if they may be exploited to support the management of the Local Voronoi Diagrams of the peers such that real time constraints are respected.

In Figure 46 is showed the temporal overhead introduced by the most used VAST functions to implement and maintain the overlay, Insertion/Update and GetEnclosingNeighbor operations. The results show that VAST may support a Voronoi-based overlay topology without loss off scalability up to 800 1000 nodes within the same AOI.

4.7.2 Peersim

Peersim(55) is a $P2P$ simulator designed to support highly scalable simulations of very large scale P2P systems, overlay and protocols. It supports the modular implementation of new overlay topologies and protocols. It handles peers which dynamically join and leave the network, but the notion of dinamicity has to be redefined for our purpose. It supports the

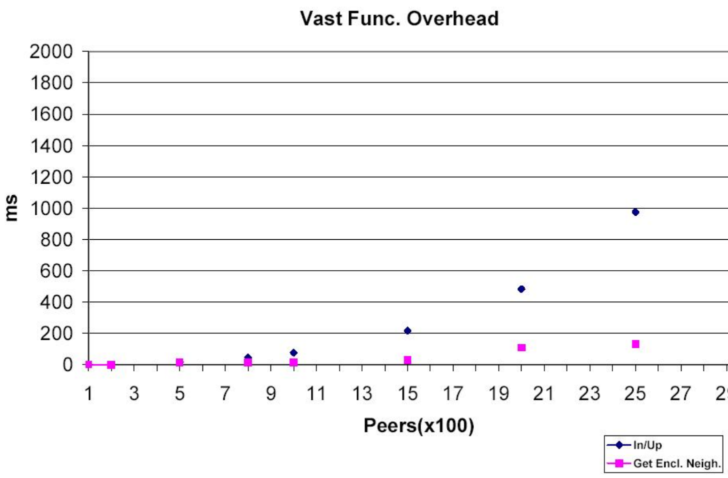


Figure 46: VAST functions overhead.

simulation of a high number of peers in order to test the scalability of new protocols.

Peersim supports two different types of simulations:

- **Cycle Based:** This is the simplest type of simulation as it does not take into account delays and issues relate to the *Transport layer*. It makes it possible to achieve extreme scalability and performance and it is suitable to test, for instance, a dynamic overlay topology in order to find out, the average degree of each peer, the number of links created/destroyed by a peer at every simulation cycle, the number of messages sent/received by each peer.

- **Event Based:** It introduces the *notion of time* within the simulation. It supports the simulation of message sending/receiving and the simulation can be configured in order to define both the delay and the probability of messages loss. Each message is inserted into a heap, afterward it is scheduled by the *Event Scheduler*. The simulation stops when the event queue is empty. In our case this model of simulation can be useful, for instance, to simulate delays in the propagation of heartbeat messages.

The core simulator has been extended to make it possible the dynamic removal of the links of the simulated overlay. In fact, in *PeerSim* a node can join or leave the overlay, but it cannot modify its connections within the overlay during the simulation. In our case, since each peer dynamically moves within the DVE, the topology should be modified at each movement of the peer.

We have exploited the cycle based version of *Peersim* in order to execute several experiments to analyze the topology of the Voronoi based overlay under different scenarios which have been generated by defining a set of simulation parameters, like the numbers of the peers of the DVE, their speed and the type of links of the overlay. For instance, it is possible to define an overlay including only links between Voronoi neighbors, or to links between a peer and other peers located within its *Internal Area of Interest*. In this case, the range of the *Internal Area of Interest* is a parameter of the simulation as well.

4.7.3 Overlay Evaluation

As discussed in the previous sections, it is possible to define two orthogonal solutions for the definition of a Voronoi based overlay topology. As a matter of fact, it is possible to define direct links between Voronoi neighbors only, or between a peer and other peers located in a given range. We recall that both solutions may present problems especially when crowding scenarios occur. In this case, when a lot of of peers are close each other in the same subregion, a large amount of AOI overlaps and the former solution suffers of the problem related to the latency of heartbeats propagation, because of the multiple hops required to deliver an event. Since we consider *real time applications*, the issue related to latency is very important, because a high latency may compromise the interaction of the users with the DVE.

The latter solution, instead, may require a high communication bandwidth. As a matter of fact, a peer should send the same heartbeat to each peer within the given range. Since some peer may have a limited communication bandwidth, such problem can compromise the stability of the application.

The purpose of the first set of tests is to evaluate the scalability of a Voronoi based overlay when crowding occurs. We evaluate the average number of links of each peer in different scenarios, i.e. varying the number of peers, the type of the links, the speed of peers. Our purpose is also to evaluate the frequency which characterizes the definition/deletion of the overlay links.

The solution introduced in Sect.4.5 which exploits *Internal Area of Interest* and *Peripheral Area of Interest* has been evaluated as well. The behavior of the overlay has been evaluated in different scenarios to understand how the range of the *Internal Area of Interest* influences the average number of links, when different number of peers are considered.

Configuration of the Experiments

We exploit the cycle based simulation model of *Peersim*, in order to relate the length of a Peersim cycle with the interval of time elapsed between the transmission of two successive heartbeats. Since the frequency of heartbeat notification is up to 5 per second, to cope 5 minutes of a real time execution, we should plan 1500 Peersim cycles. It is worth noticing that a lower frequency can be chosen using a dynamic approach, where the frequency of the heartbeat notification is dynamically defined. In this case, a set of *Dead Reckoning* (36; 38; 39) techniques are required.

In our experiments, the peers move within a two dimensional *800x600 grid*. At each simulation cycle, each peer changes its direction at random, afterward it moves and covers a distance which is proportional to its speed. When a peer hits the border of the grid, it rebounds to in order to stay inside the grid. The values of the speed span *from 1 to 12 pixels*.

At the beginning of each test, the peers are positioned inside the grid at random, according to a *uniform distribution*.

The parameters of our simulations, which may be defined through the configuration file of $Peersim$ are the following.

- *Peer Number:* This parameter makes it possible to measure scalability and to simulate crowding scenarios as well.

- *Peer Speed:* This parameter is introduced to measure the frequency of the changes of the topology of the overlay and to model particular types of crowding as well.

- *Probability of Change Direction:* This parameter may model particular types of DVE scenarios. For instance, in a *RPG game* the direction of the avatars change more frequently than in a *cars's race*, where each avatar follows a given direction, like other avatars close to them.

- *Type of Links:* This parameter defines whether a peer is connected to its Voronoi neighbors only, or to any peer located in a given range.

We have performed three different kind of tests.

- The purpose of the first set of experiments is the analysis of the number of connections defined by a single peer during 1500 cycles of simulation, when different type of links are defined. We consider an *Only Neighbors* solution, where each peer defines connection with its Voronoi neighbors only, and the solution where each peer defines connections with each peer located within its Internal Area of Interest. In this case, the radius of the *IAOI* is incremented by 10 cells for each experiment. In this set of test the speed of each peer is kept constant.

- The second set of experiments test the scalability of the overlay, by varying the number of peer from 100 to 1500 and observing the average number of link in any linking solutions.

- The third set of experiments has the purpose to test the frequency of the topology changes, when the speed of the peers changes from 1 to 12, and the different type of links are considered.

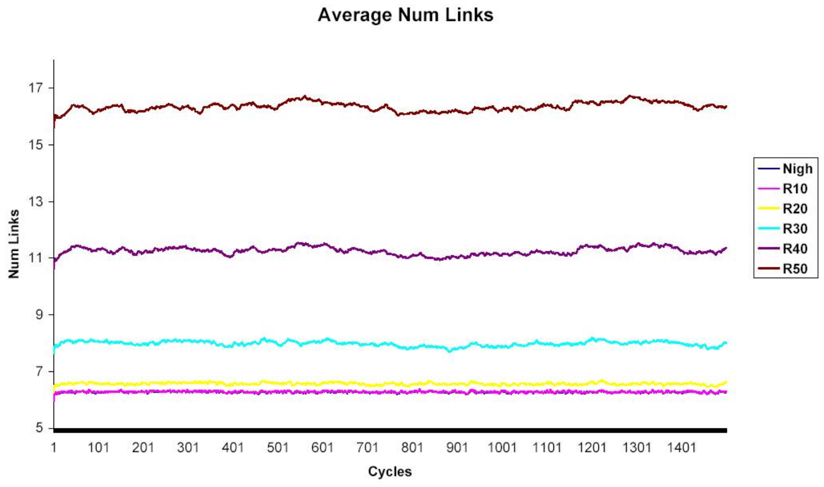


Figure 47: Average number of links during 1500 cycles.

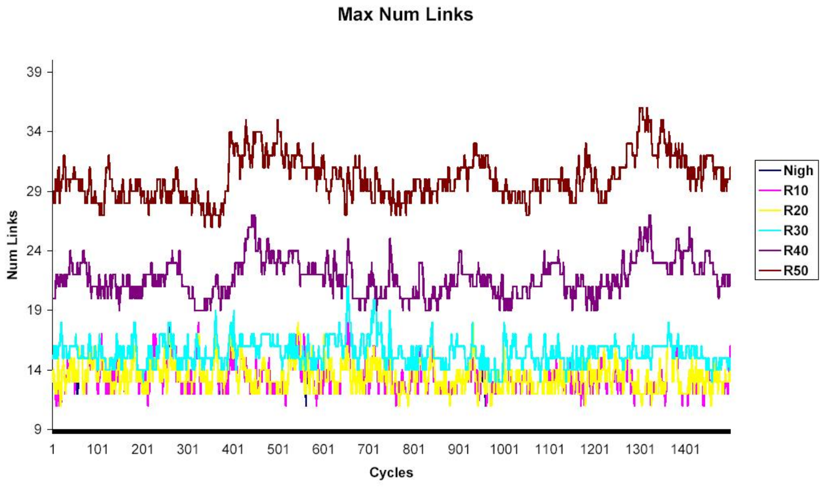


Figure 48: Max number of links during 1500 cycles.

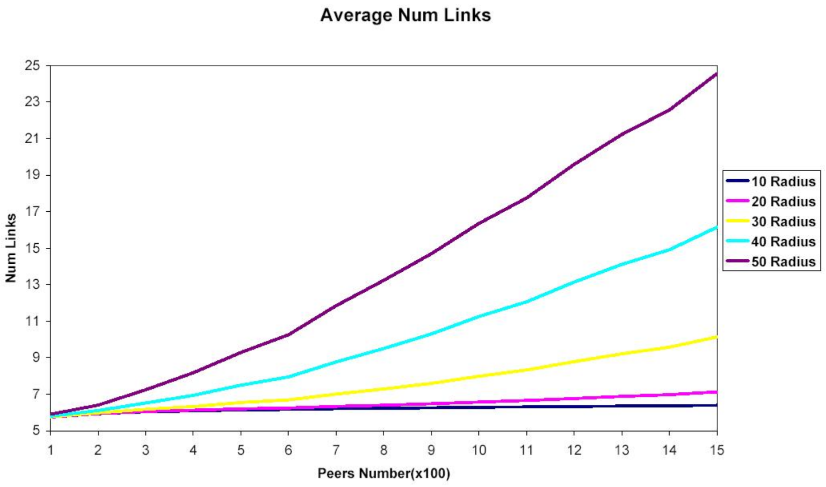


Figure 49: Average numbers of links with increasing number of peers from r10 to r50.

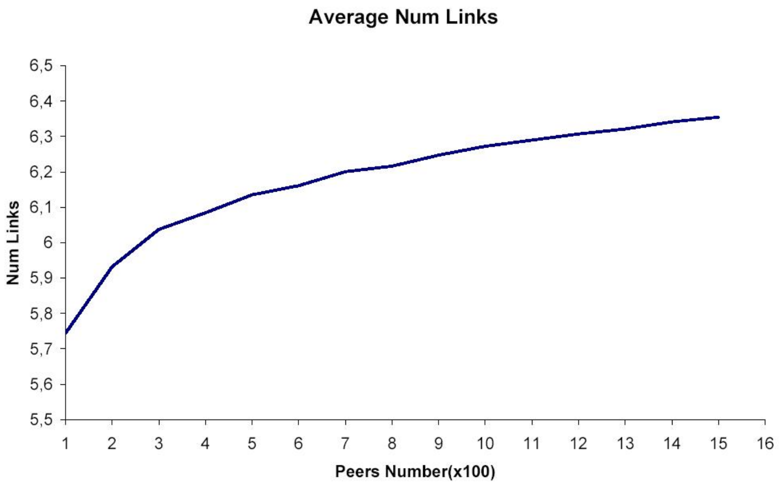


Figure 50: Average numbers of links with increasing number of peers on nigh.

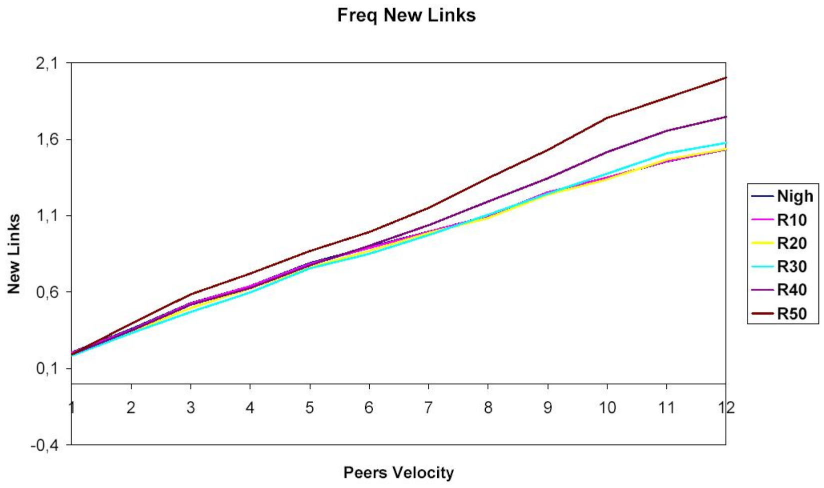


Figure 51: New links frequency with increasing velocity.

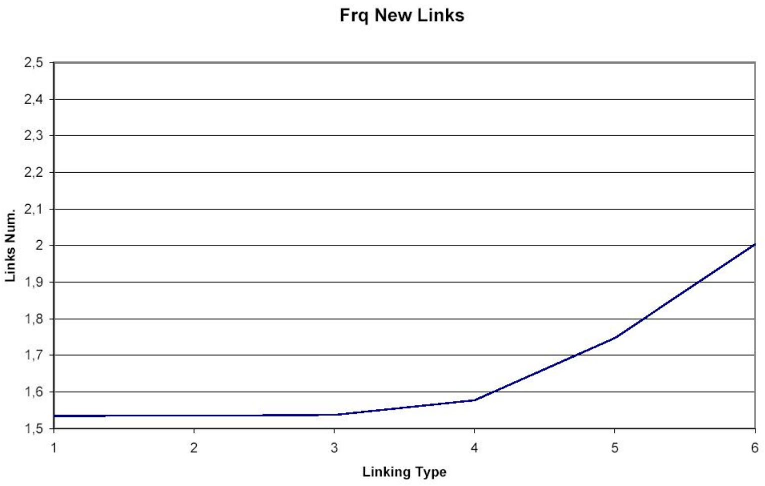


Figure 52: New links frequency with increasing velocity varying linking type.

Evaluation of the Results

Figure 47 shows the average number of links defined by each peer at each cycle of simulation, when different type of connections are considered, i.e. from *only neighbors* to *50 radius*. We consider 1500 simulation cycles and we keep constant both the number of peers and their speed during the experiment. Figure 48 shows the maximum number of links at each simulation cycle in the same conditions.

Figure 49 shows the average number of links for each type of connections, when the number of peers range from 100 to 1500. In Figure 50 the *only neighbors* linking type is considered. The scalability of the Voronoi based overlay in crowding situations is confirmed by the results shown in Figure 49 and Figure 50. As we can see, in the *Only Neighbors* solution an upper limit to the average number of links does exist. Even if in those solutions exploiting a varying radius for the *Internal Area of Interest*, the number of links increases with the number of peers, Figure 49 shows that a good scalability may be obtained by considering a small radius. This results can be useful to exploit a trade off solution which use *Only Neighbors linking* type in general situations and a small radius in crowding situation, to limit the number of retransmissions to reach all the peers inside the area of interest of the sender when the crowding occur.

Figure 51 shows the frequency on overlay changes when the speed of the peers changes from 1 to 12. Figure 52 shows, instead, the trend on new links creation in the same test of Figure 51, from the point of view of linking type, from *Only Neighbors* to a solution exploiting a 50 radius.

The experimental results confirm the theoretical one which states that the average number of neighbors is 6, when only Voronoi links are considered. Since we consider a *uniform distribution*, the average number of links is roughly constant across different simulation cycles.

The results show the feasibility of a trade off solution exploiting only Voronoi links in the general case and an *Internal Area of Interest* characterized by a small radius in crowding scenarios. As a matter of fact, while a small radius corresponds to a reasonable number of links, it may constrain the number of routing steps with respect to the solution including

Voronoi links only. The results also show that the length of the radius has a modest impact on the overlay variance.

4.7.4 Routing Evaluation

We have performed several experiments in order to analyze the *routing strategy* described in Sect.4.4. We recall that this strategy is exploited to propagate the heartbeats to the peers located in the AOI, if only links connecting with Voronoi neighbors are exploited, or to the peers of the *Peripheral AOI*, if links with neighbors located within a given range are defined.

The purpose of these experiments is to evaluate the degree of inconsistency introduced by the movement of the peers. As a matter of fact, inconsistencies may arise because the location of peers change continuously and two peers may have a different perception of the position of a common neighbor, due to the delay of heartbeat notifications. This implies that these peers perceive a *positional drift* with respect to the real position of the neighbor. As a consequence, in a dynamic scenario, B and C, two peers sharing a common neighbor R, which is the source of the heartbeat, may both neglect the propagation of an heartbeat to R because of their different view of the virtual world. Due to the *positional drift*, each peer may suppose that the other peer should propagate the heartbeat to R. On the other way round, the *positional drift* may also generate *redundant notifications*, because B and C may decide to propagate the same heartbeat to their common neighbor. As we can see in Figure 53, if the upper graph represent the local view of B and the lower graph represent the local view of C, the first scenario occurs and both B and C propagate the heartbeat to A. Otherwise, in the opposite situation, the second occurs where A does not receive the heartbeat at all.

It is worth noticing that these problems are introduced by the dynamic nature of the application we consider, and, at the best of our knowledge, they have not been investigated in the literature.

The first set of experiments evaluates the number of peers located within the *AOI* of a peer P which does not receive an heartbeat generated

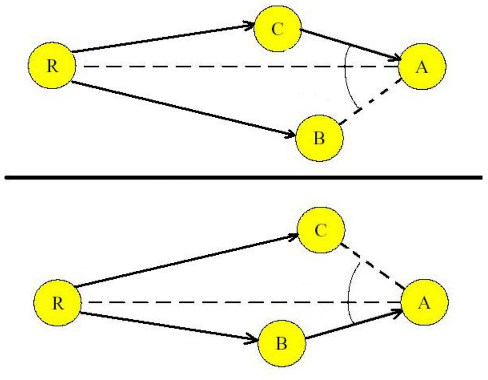


Figure 53: Spatial drift inconsistencies.

by P.

To simplify the execution of this test, each peer P is created at a random position *inside* the AOI of the peer G which generates the heartbeat, afterward it can move freely inside the AOI, but it cannot exit $AOI(G)$, i.e. it bounces on the border of the AOI. In this way, it is possible to evaluate, for each heartbeat h generated by G, the number of peers which have not received h, due to the *positional drift*. As a matter of fact, in this scenario, h should be received by any peer located in its AOI. In a more general scenario, it is more difficult to detect the peers which should receive h, because of new peers entering the AOI and of peers leaving the AOI.

To set up the simplified scenario, we have done a set preliminary tests to find out the average number of peers belonging to the AOI of a predefined peer P as a function of the number of peers present in the DVE and of their speed. Each peer is generated at a random position of the DVE and it can move inside the DVE and enter/exit an $AOI(P)$. The preliminary test exploits 1000 cycles, at each cycle a Peersim control counts the number of peers inside $AOI(P)$. The average number of peers present

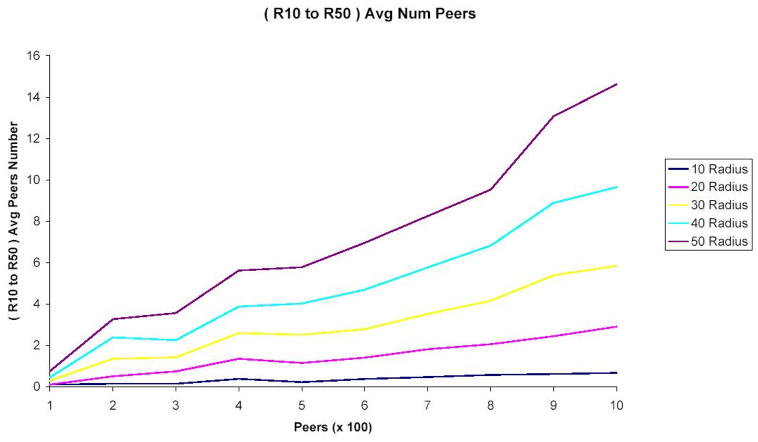


Figure 54: Average number of peer .

inside $AOI(P)$, as a function of the whole number of peers inside the DVE, is the referential parameter of the real tests. The results of the analysis find out in the preliminary test are illustrated in Figure 54, where we can find the average number of peers present inside a $AOI(P)$ when the radius of the AOI ranges, from 10 to 50, and when the number of peers varies from 100 to 1000.

The number of peers generated inside the AOI is proportional to the density found out in the preliminary test. The upper part of Figure 55 shows a snapshot of the preliminary experiment, where the AOI of the predefined peer is highlighted, while the lower part of the Figure shows a snapshot of the following experiments which is based upon the previous measures.

In each test, a particular peer **A** generates a stream of heartbeats. Figure 56 to Figure 60 illustrates the measures of interest measured by this sequence of tests, which are:

- Total number of messages sent for the propagation of an heartbeat.
- Number of redundant messages received for the propagation of the

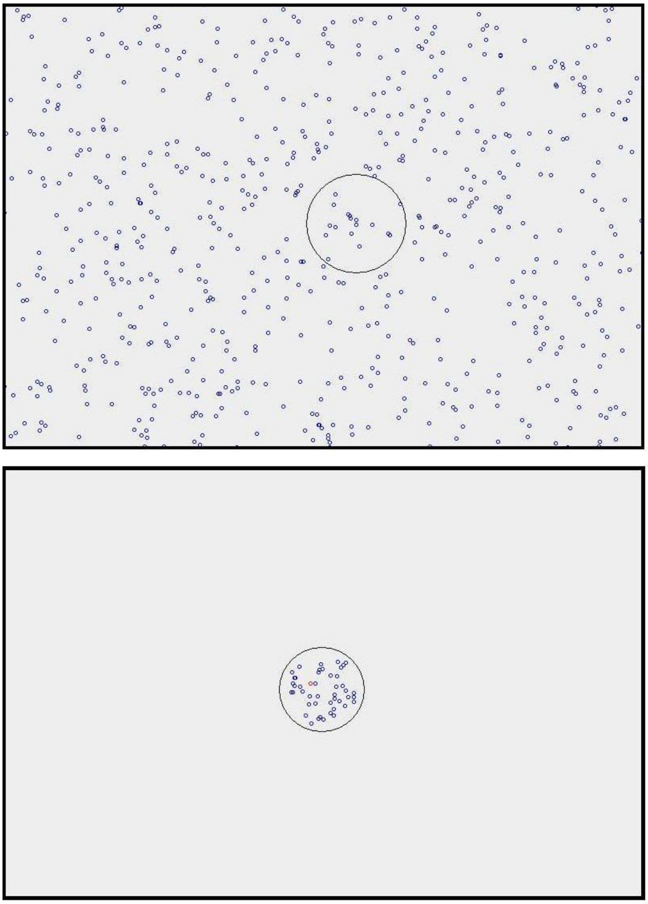


Figure 55: Tests examples.

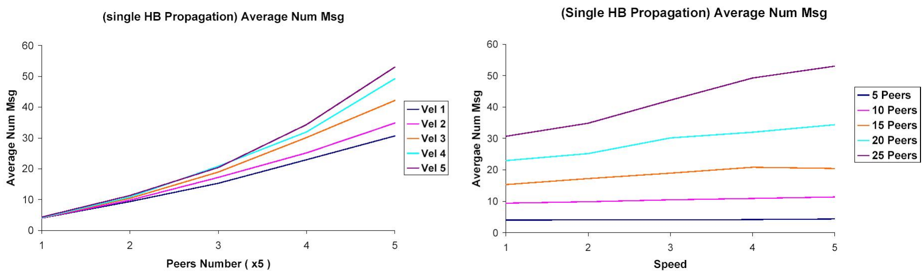


Figure 56: Average number of messages.

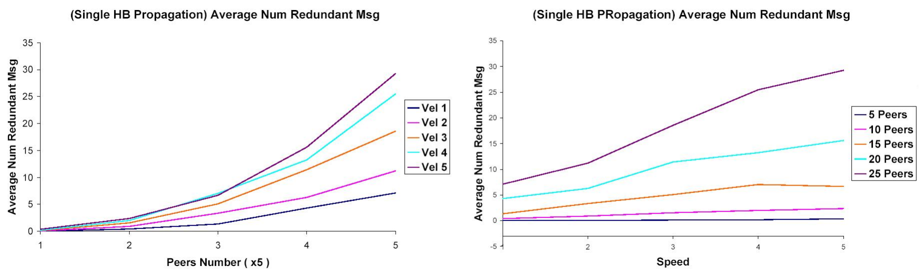


Figure 57: Average number of redundant messages.

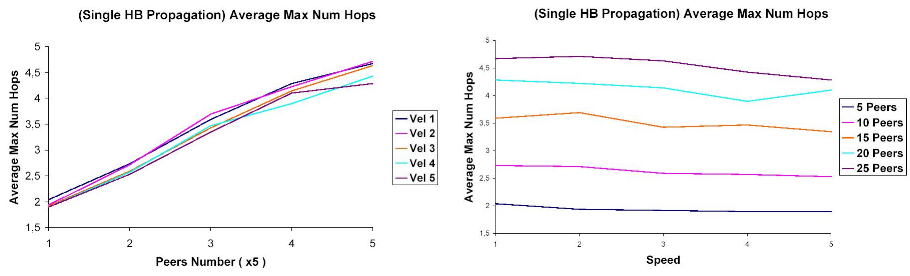


Figure 58: Average max number of hops for each HB.

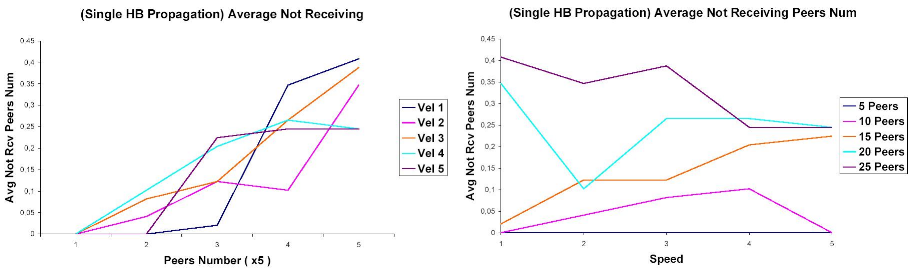


Figure 59: Average number of not receiving peers.

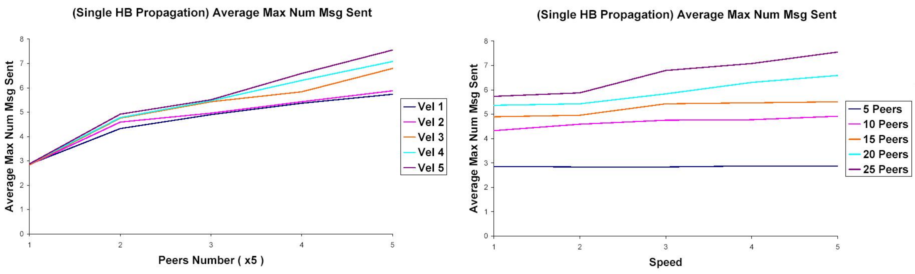


Figure 60: Average number of messages sent for each peer.

heartbeat.

- Maximum path to propagate an heartbeat
- Number of peers which have not received the heartbeat.
- Maximum number of messages sent from a peer to propagate an heartbeat.

We consider a *stream of heartbeats* generated by the observed peer. Each measure is computed by computing the average on the number of the heartbeats belonging to the stream. The peer moves in the DVE according to the mobility model previously described.

The left side of each Figure illustrates each one of the previous measures as a function of number of peers, from 5 to 25 peers, with a step of 5 peers while the right side of each figure shows the same measure as a function of the speed of the peers.

As expected (Figure 56 left side) the average number of messages required for the propagation of the heartbeat increases with the number of peers in the AOI, but the speed of the peers (Figure 56 right side) has a low impact unless crowding occurs.

Redundant messages (Figure 57) also have the same behavior.

The length of the maximum path (Figure 58), is proportional to the number of peers inside the AOI and the speed of the peer does not affect it.

The average number of peers which have not received the heartbeat, see (Figure 59), increases with the number of the peers.

Finally, as expected, the maximum number of messages sent from a peer to propagate an heartbeat (Figure 60) is, on the average, equivalent to the average number of link in a Voronoi based overlay, i.e. 6 links.

To face the phenomenon of peers which do not receive the heartbeat, we have modified the algorithm defined in Sect4.4.1, in order to reduce the number of peers which do not receive the heartbeat at the price of a greater number of redundant messages.

In the original algorithm based on reversing compass routing, each peer P chooses its children in the spanning tree rooted at R by consider-

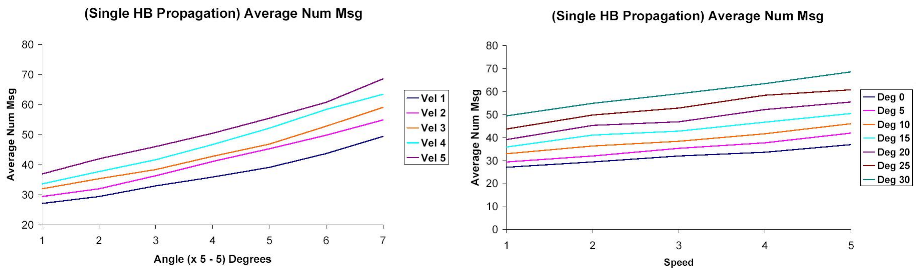


Figure 61: Number of messages.

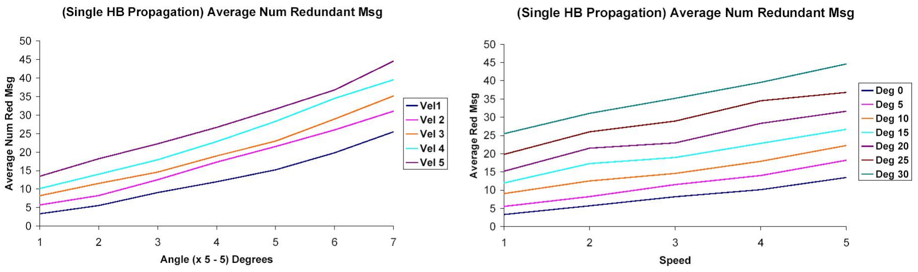


Figure 62: Number of redundant messages.

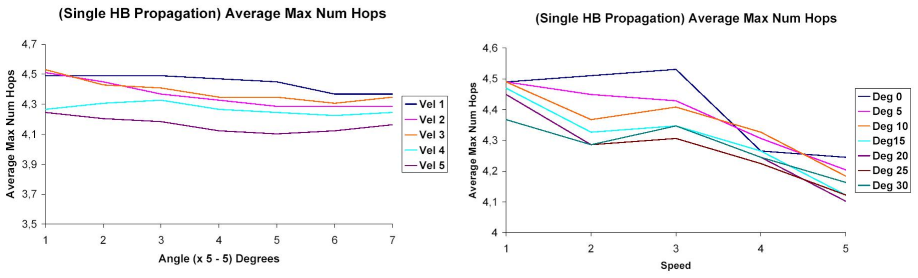


Figure 63: Maximum number of hops for each heartbeat.

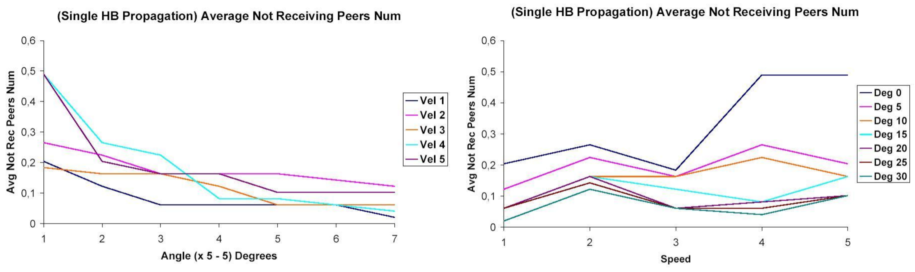


Figure 64: Number of peers not receiving the heartbeat.

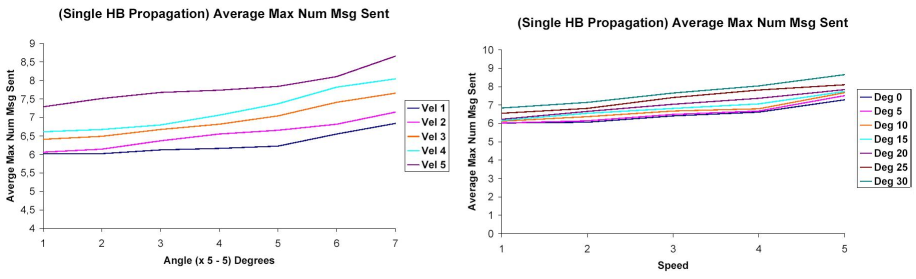


Figure 65: Number of messages sent for each peer.

ing the angle a between R, its child C and itself. This angle is compared against the angles between R, C and each common Voronoi neighbor of P and C. C is a child of P if a is the smallest angle.

We modify the algorithm introduced in Sect.4.5 in order to consider C as a child iff the difference between a and the neighbors angles is smaller than a given threshold. In this case the same an heartbeat may be notified to a peer P by more than one peer. It is worth noticing that in our case it is better that a greater number of messages is sent, instead that some peers do not receive the heartbeat at all. In general, however, a compromise must be chosen, in relation to the type of information sent with the heartbeat. For instance, we could choose to not increase the number of redundant messages so introducing a number of peers not receiving the heartbeat and to exploit some dead reckoning strategy as a counter measure.

The threshold chosen for our experiments spans from an angle of 5 degrees to one of 30 degrees with step of 5 degrees.

Unlike previous experiments, we have fixed the number of peers to 25, ranging the degree of the threshold from 0 (which corresponds to the previous test) to 30, and speed of the peers from 1 to 5.

The measures of interest are the same of the previous experiments. The organization of the figures, from Figure 61to Figure 65, is the same as well, but in this Figures, the left side shows the measures of interest as a function of the degree threshold, while the right side shows the same measure as a function of the speed of the peers.

We can notice that (see Figure 64 left side) the number of peers not receiving the heartbeat is decreased by the introduction of the angle threshold. On the other side, a larger number of redundant messages is generated, as shown in Figure 62).

4.7.5 Mobility Models

To highlight the behavior of the network consistency in two quite separate situations, different *models of mobility* have been defined and implemented. A mobility model defines the strategy exploited by each peer to decide the direction of its next step. Even if for each peer the choice is defined on the basis of local information only, in some situation an amount of global information may be required to define the behavior of the peers in order to simulate a group coordination.

Beside the *Random Walking* model, that define a complete random movement of the peers toward distinct targets, we have implemented a model, the *Complex Battle*, to simulate a virtual battle.

The characteristics of the *Complex Battle* model are the following:

- *Target acquisition*: Each peer is directed toward a *target*, where it is going to fight.
- *Path calculation*: To reach the target.
- *Release of the target*: After having reached the target, a peer returns to its home base.

While *Target acquisition* and *Release of the target* have an explicit semantic, the *Path calculation* requires further evaluations. Since the purpose is to simulate the behavior of a set of avatars moving towards a particular area from different directions, we have implemented a non deterministic path manager. While a peer is going toward the target, it may modify its direction to the target with a low probability, while maintaining with high probability the target direction, This simulates the behavior of an avatar in a battlefield where it can met a set of obstacles.

When the simulation starts, all the peers are arranged randomly and partitioned into two different teams. Each team is initially positioned at an home base, that is a particular area on the map afterwards each peer moves to the same target, which is the center of the battle. Once the target will be reached by all the peers, a Battle Crowding scenario (see Sect.4.2) will be generated. The peers inside a particular range from the target will move completely at random. Each time the protocol computes the new direction, each peer rolls a dice and decides whether possibly leave the battle and go toward the home base. If it happens, that peer choices its home base as a new target and starts to move toward it.

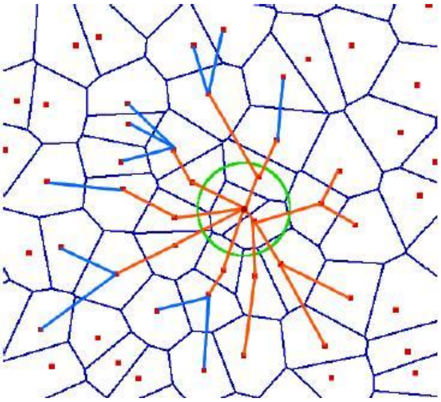


Figure 66: Time to live value = 3.

When peers are back at the home base another particular scenario is generated,the *City Crowding* one (see Sect.4.2) where the peers slow down their speed and change their behavior to Random Walking. After a while, a new cycle starts and a new target is generated.

Results

Each test described in this section concerns 200 peers. This amount of peers is really significative to evaluate crowding scenarios, since the size of areas of interest and areas of crowding have been calibrated so that in crowding battle scenarios the average density of peers into the AOIs is about 80% of the peers included in the whole DVE.

In this set of evaluations, a time to live (TTL) for the heartbeats has been introduced. A TTL=3, for instance, guarantees that the heartbeat, sent by its source, does at least a 3 hops path from the source, as shown in Figure 66. Instead, if the AOI of the source is very crowded by a great number of peers, then the heartbeat will be sent only inside the AOI. The TTL then could be a measure to avoid the overlay partition, as expressed in Section 4.6.

The first experiment, executed to measure the network consistency,

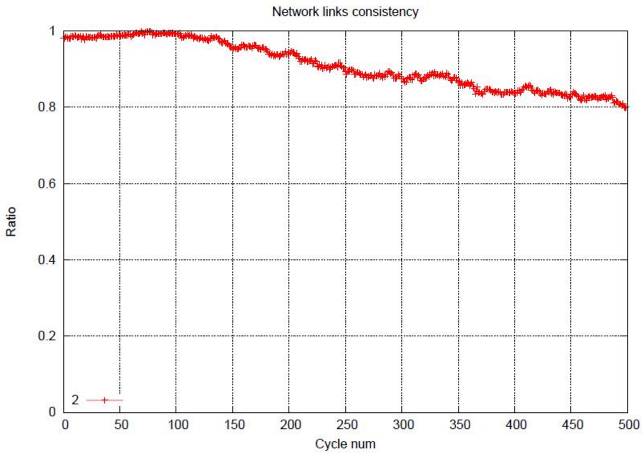


Figure 67: Network links consistency: Random Walking, no Compass tolerance.

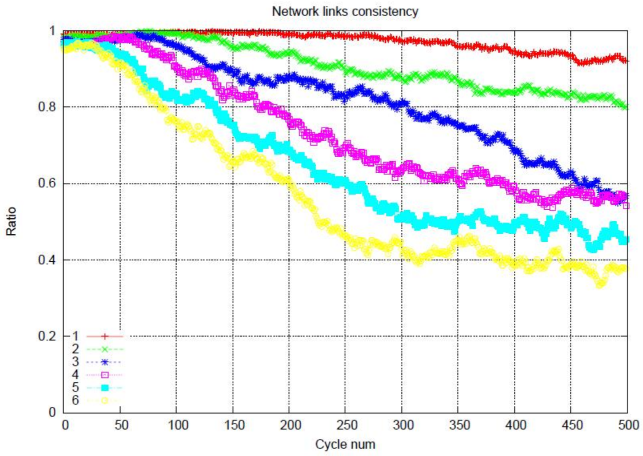


Figure 68: Network links consistency: Random Walking, Variable velocity, no Compass tolerance.

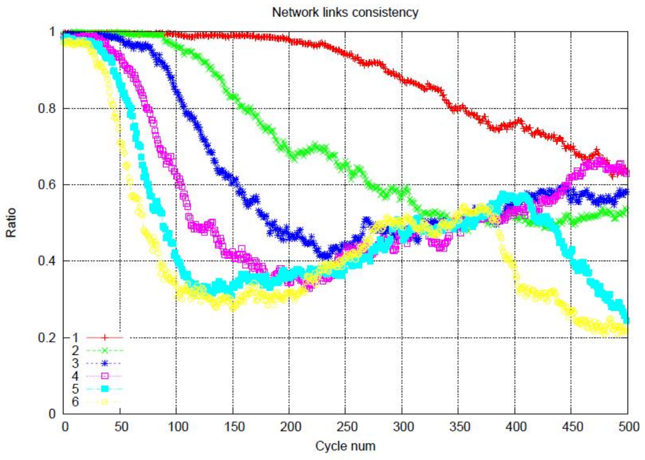


Figure 69: Network links consistency: Complex Battle, Variable velocity, no Compass tolerance.

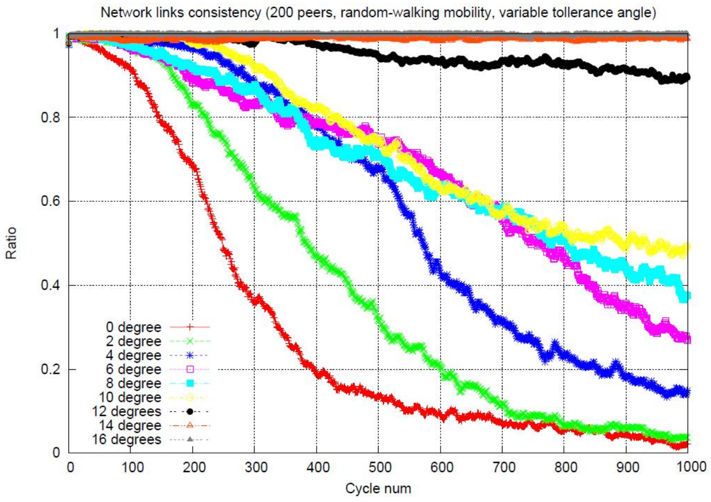


Figure 70: Network links consistency: Random Walking, Variable velocity, Compass tolerance.

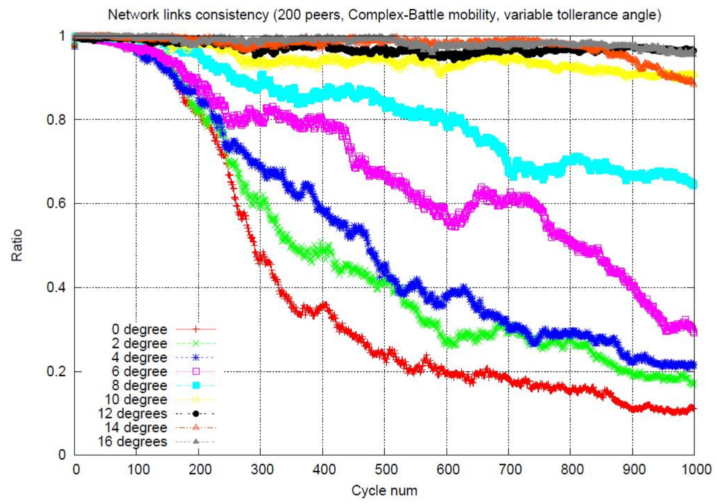


Figure 71: Network links consistency: Complex Battle, Variable velocity, Compass tolerance.

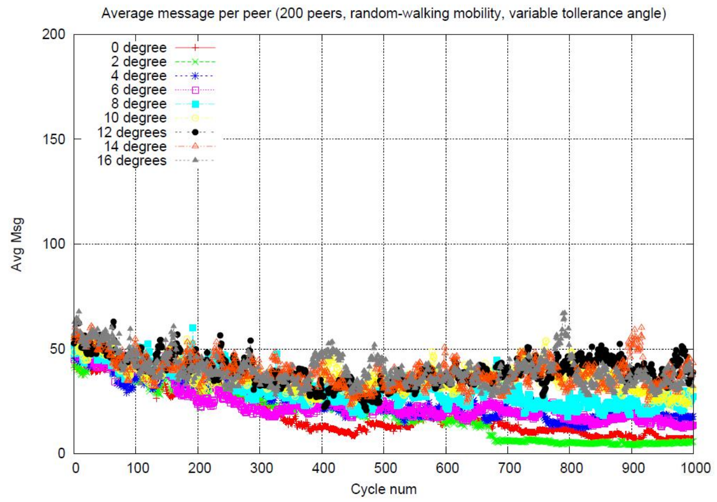


Figure 72: Average messages for each peer: Random Walking, Compass tolerance.

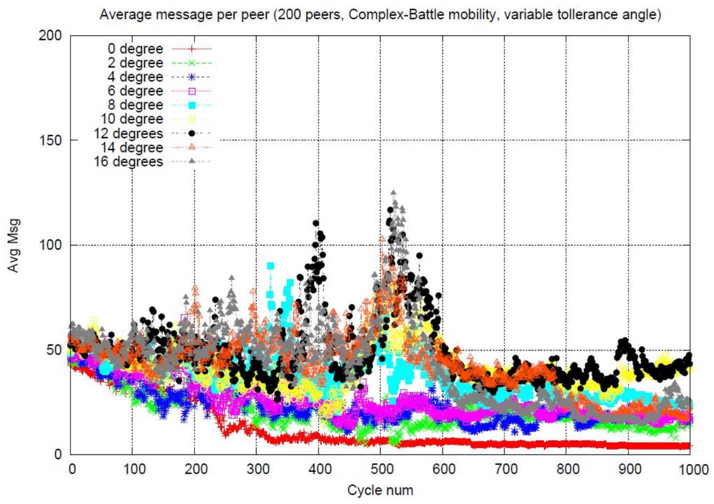


Figure 73: Average messages for each peer: Complex Battle, Compass tolerance.

has been executed without introducing any tolerance in the Compass Routing based spanning tree construction, with a TTL equal to 3, speed equal to 2 and with the *Random Walking* mobility model. In the following tests, a *cycle* is the interval between the sending and the delivery, by each peers, of the heartbeats. The network consistency has been calculated after each cycle, comparing the local view of each peer, with the global overlay. If a peer, in its local view, presents a different number of links in respect to the number of link which that peer had in the global overlay, of the previous cycle, then that peer is considered not consistent in the test. The result shown in Figure 67, prove the effectiveness of the protocol. The network topology through the distributed knowledge over various peers with local (and then limited) vision, always keeps a consistency degree more than 80%. Note that the test, thanks to the mobility model used, starts from a relaxed state where the peers are spread all over the map, and then create a really strong crowding situation.

The next tests are characterized by a *variable speed* of the peers, i.e. by a variable length of the step of each peer at each movement.

Figure 68 shows the network consistency with the *Random Walking* mobility model, $TTL = 3$, for different speeds.

Figure 69 shows the network consistency in the same conditions of the previous one, but the mobility model is the *Complex Battle*.

Both figures show that the higher is the speed the lower is the network consistency.

Till now no Compass tolerance has been used. The next test shows how the network consistency and the number of message sent by each peer, for different tolerance angles of the Compass. The angle spreads from 0 to 16 degrees.

Figure 70 shows the network consistency with the Random Walking mobility model, $TTL = 3$ and $speed = 2$. Instead, Figure 71 shows the network consistency with the *Complex Battle* mobility model, TTL=3 and speed=2.

Previous tests enables to conclude that the Compass tolerance improves network consistency, as shown in Figure 70 and in Figure 71, where the higher is the compass tolerance the higher is the network consistency in both the *Random Walking* and the *Complex Battle* mobility models.

Next figures show the average number of messages received by a peer in the tests characterized by the previous configuration. Figure 72 shows as in the Random Walking mobility model the number of messages increases very little when increasing the angle of Compass tolerance.

Figure 73, instead, shows how the tradeoff of an higher network consistency is paid with an higher number of messages when the angle of Compass tolerance increases in a crowding scenario.

4.7.6 Grid 5k: Experimental Results

Grid 5000 is a project of the French ACI Grid (Concertee Action Initiative) which provides a solution to a problem revealed by many discussions on the methodologies used by scientific research.

In addition to theory in research there is a strong need of large scale experiments and to compare experimental results with those provided by theoretical analysis. The size of *Grid'5000* in terms of number of sites and number of processors per site, was established according to the scale of the experiments and the number of researchers involved.

Grid'5000 main objective aim to serve as experimental bench and then to be used to conduct tests on a grid computing, and requires a great effort on a large scale, involving a large amount of resources and means to be able to have a good infrastructure for research on the grid. Indeed it involves 17 laboratories at national level, with the aim of providing to the researchers community an ad-hoc grid for all types of tests that can cover all layers of software and network protocols.

Tests Configuration

Each tests have been executed upon Rennes GRID5000 frontend exploiting 220 nodes. At the start of each test, 1500 peers are positioned within a $800 * 600$ map at random, exploiting a uniform distribution.

As in the previous tests we have implemented 2 type of linking to model the overlay, both of them are based on VAST functionalities.

Each peer move within the map every 200 ms.

Figure 74 shows the average number of links varying the type and range of the linking type.

Figure 75 shows the maximum number of links in the same situations of the previous figure.

Figure 76 shows the behavior of the average number of links for each connection type, varying the number of peers from 100 to 1500.

The results obtained by the implementation on a real platform confirm the simulation results obtained by Peersim.

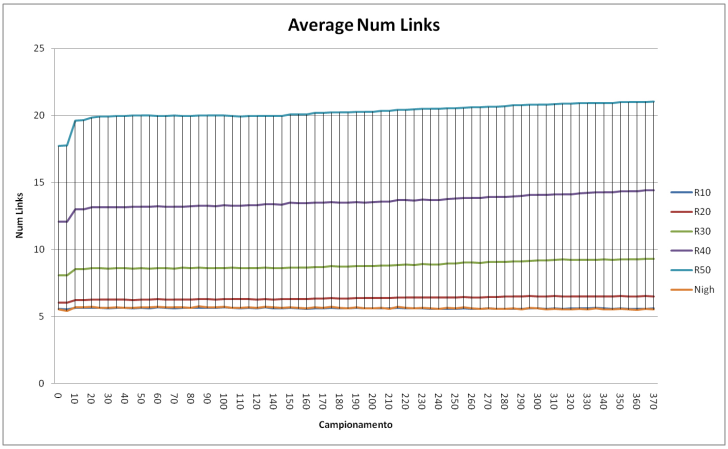


Figure 74: Average number of links.

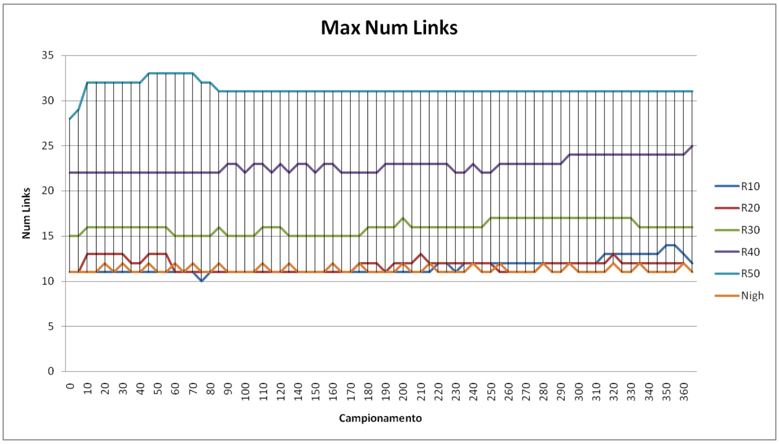


Figure 75: Maximum number of links.

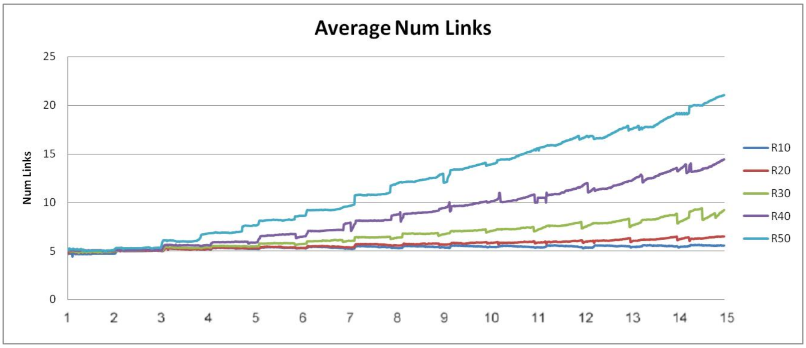


Figure 76: Average number of links varying the peers number.

4.8 Conclusions

This chapter defines a *scalable approach* for the development of a P2P support for *Distributed Virtual Environments*. Our approach is based on a *Voronoi Tessellation* of the DVE which is exploited to define the P2P overlay.

After a more theoretical analysis of Delaunay characteristics, which has lead to some theoretical results, we have defined a protocol for heartbeat propagation which requires a minimal amount of information at each peer. As a matter of fact, each peer should be aware only of its Voronoi neighbors to forward a heartbeat within the AOI, while the routing algorithm presented in (52) requires the knowledge of the two hops away neighbors. For these reason our solution considerably reduces the number of messages exchanged through the overlay and the bandwidth requirement at each peer especially when crowding occurs.

The algorithm is based on compass routing (60). Even if the general idea of exploiting compass routing to build a multicast tree has been presented in (59), at the best of our knowledge, our proposal is the first one where an algorithm is defined, implemented and evaluated.

The proposal exploits the concept of *Area of Interest* which models the locality of interactions typical of the DVE. According to this approach, the maximum degree of consistency is obtained, for each peer, within the portion of the DVE within the AOI of the peer, while the consistency decreases when the distance from the peers increases.

A further important characteristic of the proposal is that when a peer moves within the DVE, the state of the areas not belonging but close to its AOI can be easily accessed in comparison to those far away from its AOI. As a matter of fact, our approach is based on a "pass the word" strategy, where the peers on the border of the AOI of a peer p "put in touch" p with new peers entering its AOI from outside. This mechanism is simply based on the propagation of the heartbeats and does not require a further request reply mechanism like that introduced in (1) which introduces further traffic on the overlay.

The experimental results show that the Voronoi based overlay has got good scalability properties and the routing strategy, based on a modified version of the Compass algorithm, has got a good behavior even in crowding scenarios.

Chapter 5

Passive Objects Management in Voronoi based DVE

5.1 Introduction

This chapter introduces the problem of the management of the passive objects in a Voronoi based DVE and illustrates our solution. The proposed solution manages both the *consistency* and the *persistency* of the state by exploiting the properties of the Voronoi tessellations. A specification of the proposed protocols is defined by *Mobile Unity*. The last section shows the experimental results obtained by a set of simulations implemented through *Peersim*.

5.2 Managing Passive Objects in DVE

As already discussed in chapter 4, the *scalability* of the DVE may be improved through the concept of *Area of Interest*, so that an avatar receives only notifications of events occurring in its *AOI*. The *AOI* should be exploited for the management of the passive objects as well. As a matter of fact, the event corresponding to the *creation* or to the *update* of a passive object, should be propagated to any peer which is located in the surroundings of the passive object. In this way, each peer should keep a consistent state of any passive object in its surroundings and a lower amount of messages should be required.

The mechanisms introduced in the previous section are still valid. For instance each peer may *dynamically acquire knowledge* of the objects in its surroundings as it moves within the *DVE* by exploiting a 'pass the world approach', like that discussed in the previous chapter. On the other way, the problem of the management of the passive objects poses novel problems. For instance, a set of mechanisms to guarantee the *persistence* of the objects of the inhabited regions of the *DVE* have to be defined. Another challenging issue is the management of passive objects in a fully distributed environment. A solution based on the *replication* of the state of the objects onto a set of peers, i.e. the peers located in its surroundings, improves the scalability of the *DVE*, but requires the definition of proper mechanisms to guarantee the *consistency* of the objects in spite of the *concurrent updates* of the peers. Finally, we should define proper mechanisms to guarantee a good balance of the load related to the management of the passive objects among the peers of the *DVE*.

5.3 A Voronoi Based Approach

In this section we propose to dynamically exploit the subdivision of the *DVE* into regions defined by the *Voronoi Tessellation* for the management of the passive objects.

Definition 3 *The* coordinator, *or* owner *of a passive object* ***A****, is the peer* P *whose Voronoi region* Vor(P) *includes* ***A****, i.e. the coordinates of* ***A*** *belong to* Vor(P).

Note that, due to the Voronoi properties, the *owner* of an object O is always the peer closer to O. The owner of an object O may be modified due to the movement of the peers. It is worth noticing that the ownership of an object O is *always delegated* by the owner of O. When the owner OW of O realizes that O no longer belongs to its *Voronoi Area*, OW detects its

Voronoi neighbor V such that O belongs to *Vor(V)* and sends the current state of O to V. V thus becomes the new *owner* of the object. A drawback of this solution is that the ownership of an object O may be continuously transferred between close peers in crowding scenarios. Section 5.6 will introduce a strategy to avoid this situation.

Each passive object is characterized by the following *Accessibility Areas*

- *Visibility Area of Interest(ViAOI)*. This is a circular area centered at the object, whose radius must be equal to that of the *AOI* of the peer. The *Visibility Area* of an object determines the boundary of the *DVE* where it may be perceived. The state of the object must be replicated and kept consistent within its *ViAOI*.

- *Interaction Area of Interest(IAOI)*. This is a circular area centered at the object, whose radius is smaller than that of the *Visibility Area*. The *Interaction Area* of an object O determines the boundary of the region of the *DVE* where a peer must be located to be able to *modify* the state of O.

The following issues justify the assumption that the radius of the *AOI* of a peer is equal to the radius of the *ViAOI* of an object:

- Let us suppose that the radius of the *ViAOI* of an object O is smaller than the radius of the *AOI* of a peer P. Consider an object O belonging to *AOI(P)* and a peer Q located within *AOI(P)* close to O. In this scenario, P should receive any event generated by Q while it could not be aware of O. This implies fuzzy scenarios, for instance Q picks up a powerful weapon and increases its power, P receives from Q a notification of this event, but it is not able to understand its reason that the event has generated.

- If the radius of the *ViAOI* of an object O is larger than the radius of the *AOI* of a peer P, an opposite situation may occur. P should perceive the updates perfomed upon O, while it is not aware of the peer performing those updates. Fuzzy scenarios similar to the previous ones may occur.

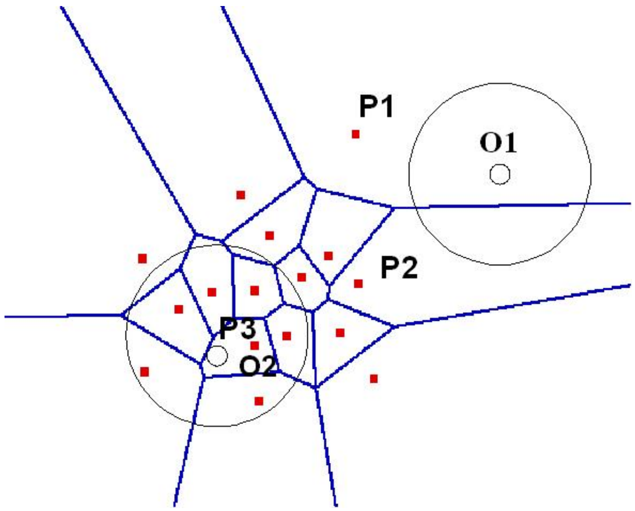


Figure 77: Visibility Areas and Voronoi Areas.

The following property holds for any assignment of objects based on a Voronoi tessellation.

$$\textit{If an object } O \in Vor(P) \textit{ and } P \notin ViAOI(O) \textit{ then } ViAOI(O) = \emptyset$$

This property follows from the structure of the Voronoi tessellation which guarantees that any point belonging to a Voronoi region *Vor(P)* is closer to P than to any other point of the DVE. For instance, Fig.77 shows a set of peers, shown by the red points and a set of passive objects shown by circles. Object O_1 belongs to the *Voronoi Region* of P_1, but P_1 does not belong to $ViAOI(O_1)$. Even if $ViAOI(O_1)$ intersects the *Voronoi Region* of P_2, $ViAOI(O_1)$ cannot include P_2, otherwise it should include P_1 as well. On the other hand, O_2 belongs to $Vor(P_3)$ and its $ViAOI$ includes a set of peer besides P_3.

The owner *OW* of an object *O* is responsible for notifying any update to the state of *O* to any peer belonging to *ViAOI(O)*. Furthermore, some *DVE* objects may be dynamically created as well, for instance a new weapon or a magic potion may pop up in any position of the *DVE*.

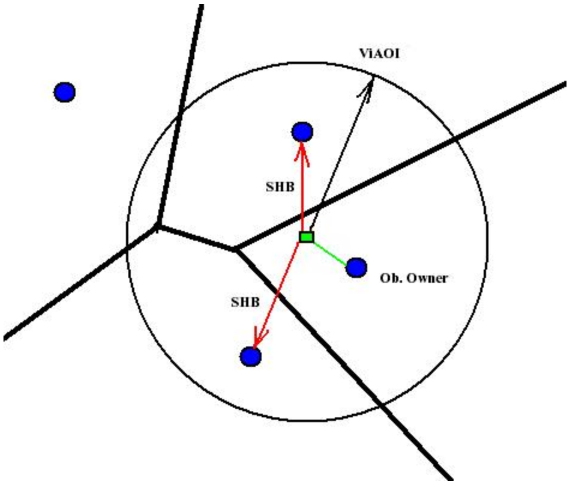


Figure 78: Static Heartbeat.

Even in this case, the *OW* should notify the presence of the new object to any peer in *ViAOI(O)*.

In the following the notification of the update of the state of a passive object or of the creation of a new object will be referred as *Static Heartbeat, SHB,* to distinguish it from the heartbeat notifying the position of the peer. Note that, while a heartbeat is characterized by a constant frequency of transmission, static heartbeats are not periodically generated.

As illustrate in figure 78, the *ViAOI* is centered on the (green) passive object. The owner of such object must notify to all peers, inside the *ViAOI*, every change upon the state of the object and manage each modification request from the peers in the *IAOI* of the object. In this way, the consistency of the state of the object is fully maintained within the boundaries of its *ViAOI* and it fades beyond them. As a consequence each peer perceives a consistent state of the portion of DVE close to it.

Even if compass Routing may be exploited to route static heartbeat, some modifications are required. First of all, the peer which generates the static heartbeat, i.e. the *owner* of the object, is no more located at the center of the area where the heartbeat should be propagated, i.e. the *Visibility Area*of the object. This implies that some hops outside the *ViAOI* of the object may be required to reach any peer belonging to it.

The second issue is related to the acquisition of new objects by a peer P entering their Visibility areas. Since the static heartbeats are no longer sent periodically, the basic mechanism of static heartbeat propagation can be no more exploited as a knowledge acquirement mechanism. These issues will be discussed more deeply in Section 5.7.

Finally, it is worth noticing that, when a set of peers are present within the *Interaction Area* of an object O, concurrent updates may be required by different peers on the same object. As a consequence, proper mechanisms should be taken into account to maintain the consistency of the DVE. The solution introduced in Section 5.5 exploits the object owner as a coordinator of the concurrent accesses. In this way the owner acts as a *temporary server* for the passive object.

5.4 Object Replication by Ownership Forecasting

As discussed in the previous section, an object **O**, initially present in a Voronoi region of a peer OW can find itself in a Voronoi region of another peer P due to the movement of the peers. In order to avoid inconsistencies, the ownership must be delegated by OW to P.

An important problem to be faced when defining a $P2P$ support for DVE is that of the state loss due to the crash of a peer or to its voluntary leave of the DVE. These scenarios often occur in a P2P environment, where peers may autonomously enter and leave the DVE.

Any solution to this problem should introduce a certain degree of *replication* for the passive objects of the DVE. Note that the semantics of these application naturally introduces a certain degree of replication, because each object must be replicated and kept consistent on any peer belonging to its Visibility Area of Interest.

We propose to increase the reliability of the DVE by exploiting the properties of Voronoi diagrams.

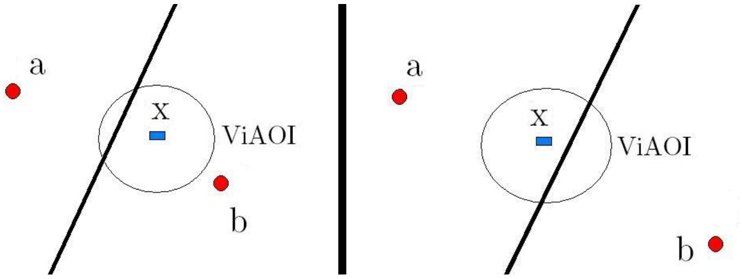


Figure 79: SHB propagation outside ViAOI.

First of all, when $ViAOI(O)$ is empty, the owner OW of O should send the state of O to each one of its Voronoi neighbors even if they are located outside $ViAOI(O)$. As shown in the previous section, since OW does not belong to $ViAOI(O)$, no other peer of the DVE is aware of O and the crash of OW implies the loss of the state of O.

This situation is shown in Fig. 79 where peer b, in the left part of the Figure, rs. peer a in the right part of the Figure, must send to peer a, rs. peer b, any change of state of the object x, even if $ViAOI(x)$ does not include a, r.s b.

In this way, if a peer P crashes, the state of the objects it owns will not be lost and the portion of the DVE owned by P will be divided among its neighbors, according to the Voronoi tessellation resulting by the new partition of the DVE.

This strategy may be optimized in the following way. Each peer P, which is the owner of at least an object, computes a Voronoi diagram including all its neighbor, but not itself. In this way, it can detect, for each object O it owns, the neighbor N which will include O in its Voronoi Area, if it crashes. P then sends a static heartbeat to N, even if N does not belong to $ViAOI(O)$.

Fig. 80 illustrates this strategy. The object owner P, shown in the left side of the figure, deletes itself from the Voronoi tessellation and finds out that in the case of its departure from the DVE, the new owner for x is N, as shown in the right part of the figure. P then sends a static

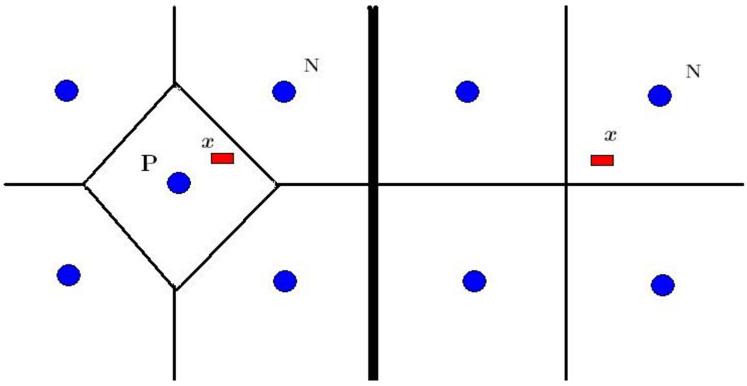


Figure 80: Ownership forecast.

heartbeat to N notifying the state of x.

This basic solution can be improved to increase the degree of replication of each object in order to decrease the probability of loss of the of critical information of the DVE. For instance, different Voronoi tessellations may be considered, each one corresponding to a different *degree of replication*. While the minimum degree of replication is computed by the owner OW of an object O, by deleting itself from the Voronoi diagram, an higher degree of replication is defined by deleting itself and its Voronoi neighbors and so on, by applying a recursive procedure. OW then considers the sequence of tessellations obtained by the recursive procedure, detects, for each tessellation, the peer whose Voronoi region includes O and sends it a static heartbeat notifying the state of O.

5.5 Object Consistency

A critical issue for the definition of a *P2P DVE* support is the definition of a proper mechanism to guarantee the consistency of the objects when concurrent updates occur.

In a DVE each peer may update an object only if it is close to it. For instance, a peer should be close to a magic potion to drink it. If the peer is far from the potion, it may throw a stone to break the bottle containing the potion. In any case, for each object **O**, we can define an *Interaction Area of Interest* or *IAOI*, that is the portion of the DVE where a peer must be located in order to modify **O**. This area can be different for different kind of objects and its exact radius is part of the object's state.

Note that if *IAOI(O)* is not empty, then at least the owner of O is located in it, because the owner is the peer which is closer to O with respect to any other peer.

Our solution is based on the *dynamic definition of a server* for each object which guarantee the consistency of the object by serializing its updates. In our model, these servers are the owners of the objects which act as *transitory servers* for them and resolve the conflicts due to the concurrent updates on the same object. Each object may be managed by different servers during its lifetime, because the ownership of the object is delegated when the peers move, but each object is owned by at most a server at each instant of time.

When a peer P is going to modify an object O, P requires to the owner OW the state of O, modifies O, then it sends back the updated state of O to OW. OW guarantees that no further request is accepted while it is serving the request from P.

The same conflict resolution strategies defined for the client/server computational paradigm may be exploited to solve conflicts due to concurrent updates. For instance, the update requests may be served according to a $FIFO$ policy or the timestamp of the request may be considered, for instance when the application exploits a NTP protocol (44; 45; 46).

The major drawback of this solution is the overhead introduced by the request-reply protocol. To decrease this overhead, an *optimistic approach* may be considered. A peer P may locally update the state of an object O by exploiting its local copy LS of the state of O. On the other way round, since the update must be committed by the owner of the object, P neither renders the updated state US of the object on the user interface nor exploits the US to update the local game state. P then sends both LS and US to the owner of the object.

When the owner receives the request, it compares LS with its copy of the state of O and if they are equal, the update is committed and a *positive acknowledgement* is sent to P. When P receives the acknowledgement, it may render the new state and update the state of the game. If the two state are not equal, a negative acknowledgement is sent to P which resets the update. P may retry the update later. Note that while this solution reduces the overhead, it *is not fair*, because a peer may not succeed in updating an object.

Different solutions may be exploited for different kind of objects, for instance, a particular type of object can be very important for the semantic of the application, instead another type could be less important. These objects could be managed through a different approach. The proper consistency strategy may be choisen according to DVE application.

Consider, for instance, a $MMORG$ scenario where all the players are involved in a time consuming virtual battle whose goal is the control of a fortified city. In this case, the consistency of state of the big doors of the fortified walls is crucial for the semantic of the application. When the player tries to destroy the doors, or tries to enter into the city, he asks for the state of the doors to their owner before any action. If the state of the doors state is not consistent, some players can enter into the city even if the doors are closed, the game is no more fair and a faction of players may win without merit.

On the other way round, consider a battle including players and non-players characters (NPC), which are avatars managed by artificial intelligence.Like for passive objects, it is possible to assign a owner to each NPC, managing its behaviour. Suppose now that a group of players try to attack a group of NPC. To maintain the dinamicity of the application each player may use its local vision of the DVE, and autonomously interact with the NPC. Then, the player sends its action to the owner of the NPC which receives the local actions performed by the player, decides a meaningful order of the events and sends the new state of the NPC through a static heartbeat.

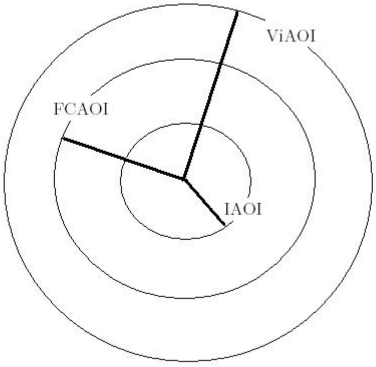


Figure 81: AOI's ranges.

5.6 Forced Coordination

The ownership delegation protocol defined in the previous section exploits the Voronoi tessellation of the DVE to define the assignment of the objects to the peers. The main drawback of the protocol is the 'ping pong' effect which may occur in *crowding scenarios*, where a pair of neighbor peers continually exchange an object O, because their common Voronoi edge steps over O, due to their movement.

To cope with this problem we define, for each object O, a third area of interest, the *Forced Coordination Area of Interest*, *FCAOI*, centered upon O as well. As shown in FIg. 81, the range of the $FCAOI$ is larger than that of the $IAOI$ and smaller than that of the $ViAOI$.

If the owner OW of an object O is located inside $FCAOI(O)$, then OW should not delegate the coordination of O to its Voronoi neighbor P, even if the peers move and, as a consequence, P includes O within its Voronoi region. In this way, in a crowding scenario, where many avatars and objects are very close within the same area, communications overhead due to the bounce of the object backward and forward from a peer

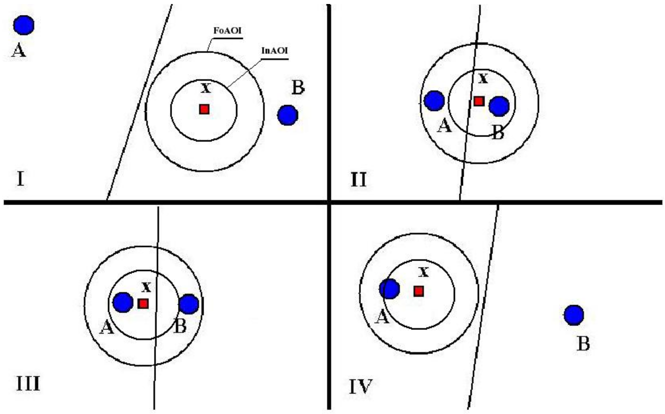


Figure 82: Forced Coordination Area of Interest

to a neighbor may be avoided.

Figure 82 shows the exploitation of the *Forced Coordinated Area of Interest* in a crowding scenario.

- ***I:*** B is the owner of x.
- ***II:*** A and B are approaching and B is inside the $FCAOI$ of x.
- ***III:*** Now x is located inside the Voronoi area of A, but since B is still located inside the $FCAOI$ of x it does not delegate the ownership of x to A.
- ***IV:*** Since B is outside $FCAOI$ of x and x is outside the Voronoi area of B, then B delegates the ownership of x to A.

5.7 SHB Notification by Compass Routing

The compass routing protocol introduced in the previous chapter has been exploited to propagate the static heartbeats within the *Visibility Area of Interest* of an object. On the other way round, it is worth noticing that unlike heartbeat messages, the static heartbeat is sent by the owner of a

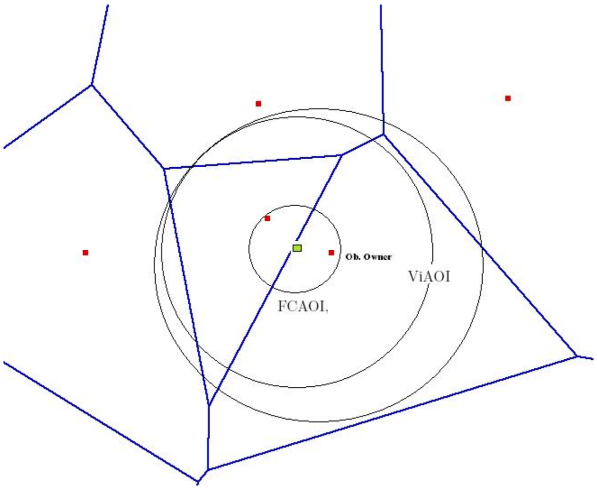


Figure 83: Extended Propagation Area

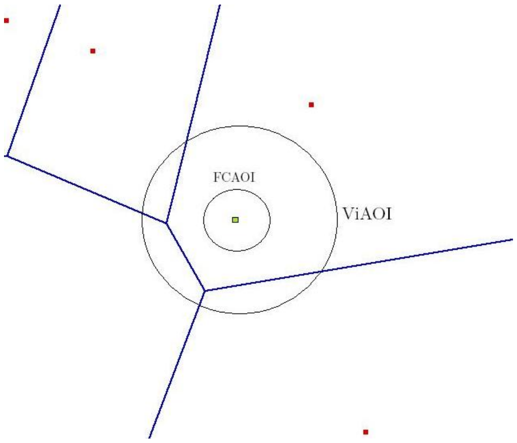


Figure 84: Object far from the owner.

passive object, which is not located at the center of the *Visibility Area of Interest* of the object, i.e. the area where the static heartbeat should be propagated.

The theorems proved in the previous chapters are valid if the heartbeat is propagated in a circular area and the source of the heartbeat is centered in this area. If these conditions hold, only the peers included in the circular area should be considered.

To guarantee that all the peers inside the *ViAOI* of the object receive the static heartbeat notifying the update, we consider an area centered at the owner, whose range equals the sum of the radius of the *ViAOI* with the radius of the *FCAOI*, as shown in Figure 83. Since this area includes the *ViAOI* centered at the object and the peer generating the static heartbeat is located at the center of this area, previous results show that any peer located in this area will receive the static heartbeat.

Note that if the owner is located outside the *ViAOI* of the object no change to the object state need to be propagated because the owner is, by definition, the peer which is closer to the object. This situation is shown in Figure 84.

Finally, it is worth noticing that the static heartbeat notification mechanism cannot be exploited for the acquirement of new objects, because of the not periodic transmission of the static heartbeats. On the other way round, the "pass the word" approach outlined in the previous sections is still valid. When a peer P belonging to the *Visibility Area* of an object O detects that a new peer P' enters the *Visibility Area* of O, P may send to P' a static heartbeat notifying the state of O. In this way, P' acquires knowledge of new objects while moving within the DVE.

5.8 Mobile Unity Specification

The figure 85 shows an high level specification of the protocols for the management of the passive objects described in the previous sections.

To simplify the specification, the maximum number $MaxObj$ of passive objects of the DVE is statically defined and no object may be dinamically created.

Each peer stores the state, the owner and the position of each object it is aware of in the array Obj where each position corresponds to a different object. The elements of Obj corresponding to objects which are not perceived by P contains the value $\perp$. The array $ObjReq$ stores the pending request for object updates required by P.

The *Move* clause in the *Assign* section models the movement of the peer.

The *GiveOwnership* clause is executed when P_i realizes that one of its objects O currently belongs to the Voronoi region of a Voronoi neighbor P_j and P_i is located outside the *Forced Area of Interest* of P_i. In this case, the owner of the object is set to P_j. This event fires a set of reactions in the Interactions Section which corresponds to the routing of a static heartbeat notifying the new Owner of the object.

The *Reset* clause is executed when a peer comes out of the *Visibility Area of Interest* of an object and resets the state of the object in the vector Obj to $\perp$.

The *RequestModify* is executed when a peer P belongs to the *Interaction Area* of an object O and stores in the array $ObjReq$ the request of update of P . This request fires one of the first two reactions in the Interactions Section. These reactions model the reception of the request by the owner of the object which checks if the update may be accepted (the first clause) or rejected (the second clause). In any case a notification is sent to P by assigning to the value true/false to the committ field of the object descriptor.

The *ModifyMyObject* clause models an update of an object executed by the owner of the object. In any case, the update of the state of the object fires a set of reactions, in the Interactions Section, modelling the compass routing based routing. These reactions are defined by the last Interaction clause.

Finally, the third Interaction clause, models the situation where a peer enters the Visibility Area of an object O . In this case, the peer receives the state of O by one of its Voronoi neighbors which is already aware of O.

```
**System** *VoronoiPassiveObjects*
**program** $P(i)$ **at** $\lambda$
    **declare**
        □ *Obj*:array $[0 \ldots MaxObj - 1]$ of
                    $(State : integer, Owner : [0 \ldots MaxPeer - 1], Pos : \lambda)$
        □ *ObjReq*:array $[0 \ldots MaxObj - 1]$ of
                    $(Req : Boolean, Newstate : integer, Committ : Booolean)$
    **initially**
        □ $Obj[k] = (\lambda_k, val_k, i)$ when $\lambda_k \in Voronoi(\lambda)$
        □ $Obj[k] = \perp$ when $\lambda_k \notin Voronoi(\lambda)$
        □ $ObjReq[k] = false, \perp, false$
    **assign**
        □ *Move*:: $\lambda := NewLoc(\lambda)$
        □ *GiveOwnership*:: $Obj[k].Owner = j$
            **reacts-to** $Obj[k].Owner = i, Voronoi(Obj[k].Pos, Peer[j].\lambda),$
                          $dist(\lambda, Obj[k].Pos) > FAOI$
        □ *Reset*:: $Obj[k] = \perp$ **reacts-to** $\lambda \notin ViAOI(Obj[k].Pos)$
        □ *RequestModify*:: $< f = Random();$
                          $ObjReq[f] := true, GenNewState(Obj[f].State), false >$
            **when** $(\lambda \in (IAOI(Obj[f].Pos))$
        □ *ModifyMyObject*:: $< f = Random();$
                          $Obj[f].State := GenNewState(Obj[f].State) >$
            **when** $(Obj[f].Owner = i)$
    **end**
**Components**
        □ i:0≤i<*MaxPeer*:: *P(i)*
**Interactions**
        □ $P(i).Obj[k], P(j).ObjReq[k].committ := (P(j).ObjReq[k].NewState, true$
                **reacts-to** $P(j).ObjReq[k].Req = true$
                                    $\wedge$
                          $P(i).Obj[k].Owner = i$
                                    $\wedge$
                $P(i).Obj[k].State = P(j).Obj[k].State$
        □ $P(i).Obj[k], P(j).ObjReq[k].committ := (P(j).ObjReq[k].NewState, false$
                **reacts-to** $P(j).ObjReq[k].Req = true$
                                    $\wedge$
                          $P(i).Obj[k].Owner = i$
                                    $\wedge$
                $P(i).Obj[k].State \neq P(j).Obj[k].State$
        □ $P(i).Obj[k] := P(j).Obj[k]$
                **reacts-to** $P(i).Obj[k] = \perp \wedge P(j).Obj[k] \neq \perp \wedge VoronoiNeigh(P(i), P(j)), \wedge$
                          $P(i).\lambda \in ViAOI(P(j).Obj[k])$
        □ $P(i).Obj[k] := P(j).Obj[k]$
                **reacts-to** $P(i).Obj[k] \neq P(j).Obj[k],$
                                $\wedge$
                          $VoronoiNeighs(P(i), P(j)),$
                                $\wedge$
                          $Is\_Parent(P(j), P(i), P(j).Obj[k].Owner)$
                                $\wedge$
                          $P(i).\lambda \in ViAOI(P(j).Obj[k])$
```

Figure 85: Passive Objects Protocol: A Mobile Unity Specification

5.9 Experimental Results

This section presents a set of experimental results whose goal is to analyze the number of ownership changes as a function of the radius of the *Forced Coordination Area of Interest* and of the speed of the peers.

The simulations have been implemented by the cycle-based version of *Peersim*. All the experiments consider 1000 peers and 3000 passive objects, the position of both the peers and the objects are *uniformly distributed* within a 800x600 grid. At each simulation cycle, the peers can move and change their direction at random and turn back when they hit the border of the grid.

Figure 86 shows the number of ownership changes when the radius of the *Forced Coordination Area of Interest* ranges from 0 to 50 pixels during 200 simulation cycles. As we expected, the number of changes of ownership decrease as the radius of the $FCAOI$ is increases.

The goal of the following tests is to investigate the maximum number of ownership changes as a function of the *speed* of the peer and of the range of the *Forced Coordination Area of Interest*, during a 1000 cycles Peersim simulation.

The maximum number of ownership changes have been computed as follows:

- a counter has been paired with each object. At each simulation cycle, the counter is incremented if the object change its owner. Note that an object can change its owner at most one time for each cycle.

- at the end of each cycle, we compute the maximum number of ownership changes in that cycle. Note that this value could not belong at the same peer of the previous cycle.

- At the end of the simulation we compute the average maximum number of ownership changes for each cycle and then we divide this number for the number of cycles. In such case, we obtain the probability, for each cycle, that an object has got to change owner in the worst case.

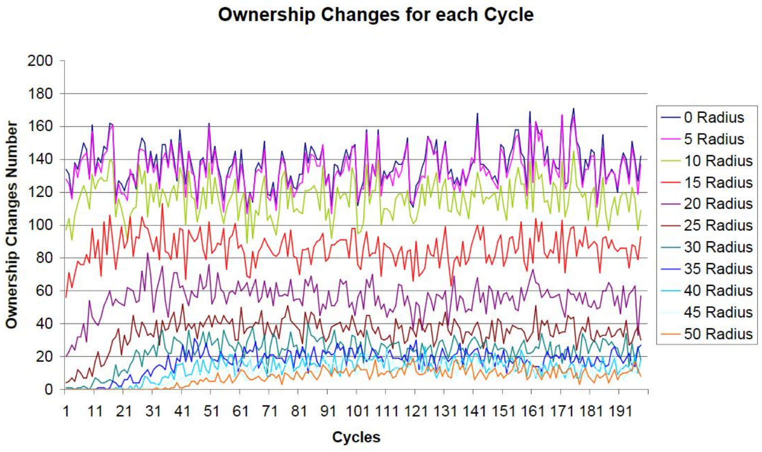


Figure 86: Number of ownership changes for each cycle.

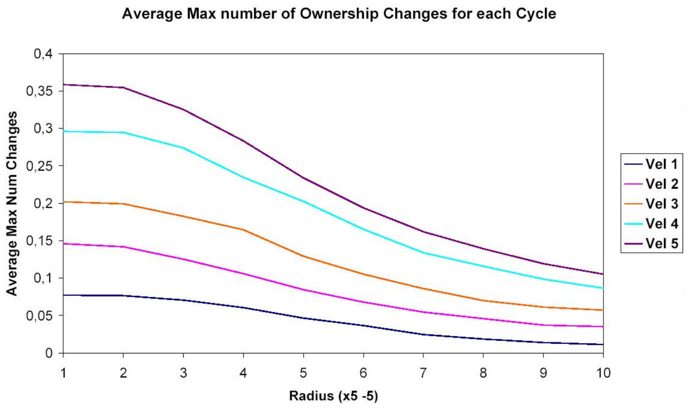


Figure 87: Maximum number of ownership changes as a function of the forced coordination radius.

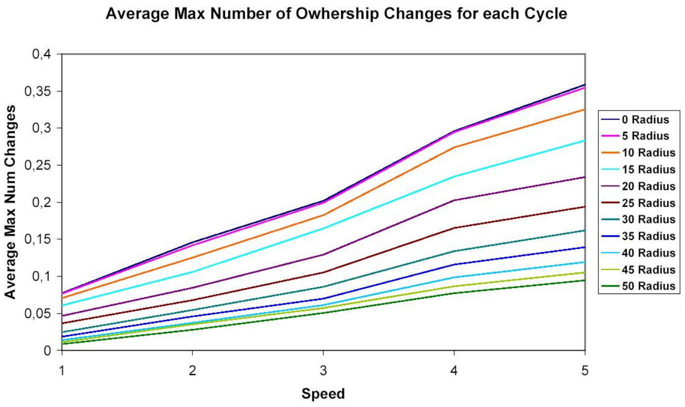


Figure 88: Maximum number of ownership changes as a function of the speed

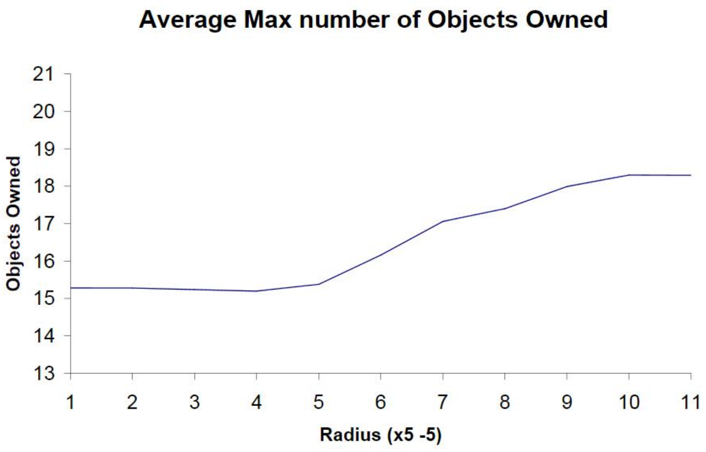


Figure 89: Maximum number of objects owned by a Peer

Figures 87 and 88 illustrate how the forced coordination radius influences, and decreases, the maximum number of changes even in presence of a large avatar mobility. That is, the probability of ownership change, for each object, is greatly lowered even when the speed of the peer is high, as shown in Figure 87 by the Vel 5 curve, with greater forced coordination range.

Note that, in general, a larger forced coordination range is paired with a greater number of objects owned by a peer. On the other way round, Figure 89 shows that the increase of the maximum number of objects owned by a peer at each cycle increases slowly when the range of the *Forced Coordination Area of Interest* increases.

Chapter 6

Hierarchical Voronoi Based Overlay

6.1 Introduction

This chapter presents a hierarchical Voronoi based overlay which is defined by pairing each peer with a *weight* which is proportional to its *bandwidth*. Peers characterized by higher weights act as *superpeers* by offering a set of services to peers characterized by lower bandwidth. *Additive Weighted Voronoi Diagrams*(56) are exploited to define a partition of the DVE that assigns to each peer a region whose size is dependent to the weight of the peer. Superpeers are modeled by sites of the tessellation that have absorbed at least the Voronoi area of another site. A simple strategy to *balance the load of passive objects management* is defined. A set of experimental results show that this approach can be a load balancing mechanism for P2P networks, that does not impair usual properties of Voronoi-based P2P networks.

6.2 Additive Weighted Voronoi Diagrams

The classical Voronoi tessellation, and hence the classical Delaunay triangulation, is defined by considering the standard L^2 metric for distance:

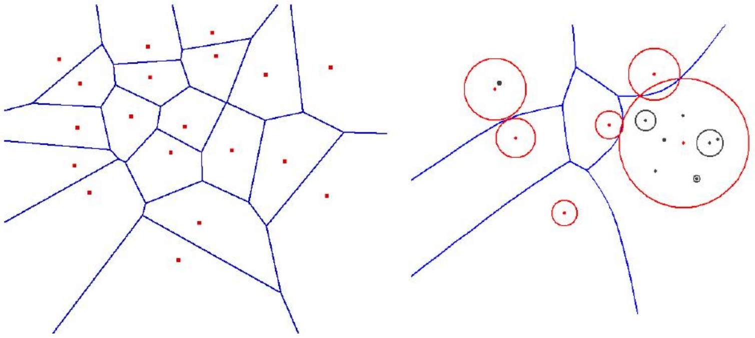


Figure 90: Classical (left) and Weighted (right) Voronoi tessellation.

$$d = ||s_i, s_j|| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

where (x_i, y_i) are the coordinates of the site s_i, and (x_j, y_j) are the coordinates of the site s_j.

Distances different from the one induced by the standard L^2 metric, can be exploited to build a Voronoi Tessellation from a set of sites. For instance, it is possible to assign a *weight* w_i to each site s_i, to tune the size of its Voronoi region $Voro(s_i)$. The resulting tessellation is referred as *Weighted Voronoi Diagram*.

The most common weighted distances are the *multiplicative weighted* one:

$$d^m = ||s_i, s_j||/w_i = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}/w_i$$

and the *additive weighted* one:

$$d^a = ||s_i, s_j|| - w_i = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - w_i$$

that lead respectively to the *Multiplicatively Weighted Voronoi Graph* and to the *Additively Weighted Voronoi Graph*, shown in right side of Figure 90.

Moreover, the *Additively Weighted Voronoi Graph, AWV*, also referred as *Apollonius Graph*, partitions the plane into a set of connected regions, where the sides of the regions are *hyperbole arcs*. In general, sites with larger weights own larger regions.

As shown in right side of Figure 90, *AWV diagrams* include both *visible* and *hidden sites*. The former own a *Voronoi Area* which may include a set of hidden sites and are connected to their *Voronoi neighbors*, i.e. the peer whose Voronoi region overlaps their region. The latter do not own any area and are only connected to the peer owning the Voronoi region where they are located. As a matter of fact, a site s_i with an high weight may 'absorb' the region of a close site s_j with a low weight. Note that the weights of the peers are shown in right side of 90 by circles centered at the peer whose radius is proportional to the weight of the peer.

If at each site is assigned the same weight the *Apollonius graph* degenerates into a standard *Voronoi graph*.

Notice that the extent of the *Voronoi Area* of a peer, and the number of hidden peers assigned to it, depends both on the ratio between its weight and the weight of the peers located in the neighborhood, and on the distance from those peers.

6.3 Modeling Hierarchical Overlays by AWV Tessellations

A *Hierarchical P2P Network* can be modeled by a *Weighted Voronoi Tessellation* where each site corresponds to a peer and whose *weight* is proportional to the bandwidth of the corresponding peer. Even if several resources may be considered when defining the weight, we focus our attention on the communication bandwidth only.

As discussed above, *AWV diagrams* include both *visible* and *hidden sites*. We recall that the *Voronoi Region* associated to a *visible site* may include hidden sites, while *hidden sites* do not own any Voronoi area. Furthermore, the overlay is defined only among the visible peers.

Visible sites model peers that do not rely on the support of a *superpeer* to forward/receive their notifications. These peers may be characterized

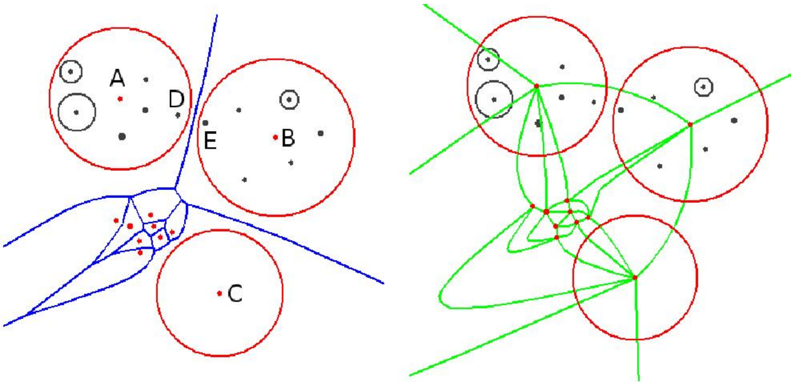


Figure 91: AWV Tessellation (left) and the Corresponding Overlay

by a high weight, or they have low bandwidth and are not able to find out a *superpeer* in their surroundings which is able to support them. In the latter case, they should directly connect to their neighboring peers to exchange notifications with them. These peers may act as servers for the peers corresponding to sites that have been hidden.

Hidden sites model peers that do rely on the support of a *superpeer*, that is the peer that has 'absorbed' their Voronoi region. The *superpeer* acts as a *proxy* for the hidden peer, so that any notification is sent/received through it.

Figure 91 shows an *AWV tessellation* and the corresponding overlay. Both peer A and peer B have absorbed some neighbors, that are shown within their Voronoi regions. Left side of Figure 91 also shows a crowd of peers (A, B, C) characterized by low weights, that have not been absorbed by any *superpeer*, because they are far away from the closest one. As we will discuss, this situation may be handled by *dynamically adapting* the weights of the peers.

The *AWV Tessellation* based approach presents several advantages. First of all, each peer may autonomously decide its current role in the *DVE*, i.e. visible or hidden, by considering both the ratio between its weight and those of its neighbors, and the relative position with respect to its neighbors.

As expressed in section 5.3, each object is dynamically associated to the peer owning the Voronoi region where the object is currently located. This peer is the *owner* of the object, stores its state and manages concurrent updates. The ownership of an object may be delegated to another peer, when the owner moves away and the object is included in the Voronoi area of another peer. Our approach naturally supports a *load balancing* strategy, because peers characterized by larger weights are associated to larger Voronoi regions, hence they manage a larger number of passive objects.

An important issue in our approach is the definition of proper weights for the peers. A simple solution *statically assigns* weights to peers, according to their bandwidth. For instance, it is possible to define two classes of peers, those which may act as *superpeers*, and the *ordinary peers*. A high weight is assigned to the former ones, while a weight equal to zero may be assigned to the other ones. In this way, an *ordinary peer* never hides other peers, while the high weight assigned to a *superpeer* allows it to support close *ordinary peers* in crowding scenarios. The weight assigned to *superpeers* must be carefully chosen with respect to size of the *DVE*. As a matter of fact, if it is too high, a few *superpeers* would serve all the other ones, so reverting to a classical client server architecture. If the weight is too small, a *superpeer* would not be able to support a large number of peers belonging to the crowd.

A more sophisticated solution is based on the *dynamic adaptation* of the weights. For instance, a *superpeer* should *reduce its weight* when its bandwidth is saturated by active connections, while it should *increase* its weight when its bandwidth is not fully exploited. Consider for instance the peer C in left side of Fig.91 which is characterized by a high weight represented by the circle centered at C. Since it has not absorbed any peer and the number of its neighbors in the Delaunay overlay is small, its bandwidth is not fully exploited. Therefore it should dynamically increase its weight in order to give support to some peers of the crowd.

6.4 AWV Based Overlays Management: Distributed Approaches

This section briefly reviews the most important concepts and describes the additions required to support *AWV* extension.

The first step performed by a new peer N joining an existing Voronoi overlay is the discovery of its *Voronoi neighbors* in order to define a set of initial connections with them. N notifies its initial position I to a *bootstrap peer*, which forwards the request by *greedy routing* to the peer P owning the *Voronoi region* including the point I. In this way, P puts N in touch with its neighbors, and the initial connections can be defined. These connections are going to change during the permanence of N in the *DVE*, because N moves around the world and hence new Voronoi neighbors might be defined.

A challenging issue is the definition of a *fully distributed strategy* which guarantees that each peer is always correctly connected to its *Voronoi neighbors* such that no disconnections occur in the overlay.

As in the previous chapters, we propose an approach based on a *'pass the word'* strategy among peers, where peers get acquainted with each other through their Voronoi neighbors. In a *DVE* each peer periodically sends a *heartbeat*, i.e. a message notifying its position, to all the peers in its *Area of Interest*. These messages may be forwarded through the Voronoi links by proper routing strategy which exploit the properties of the Voronoi tessellation to minimize the amount of messages exchanged. (65) proposes *compass routing* (64) to dynamically define a *spanning tree* covering the peers in the *AoI*. To reduce the notification delay a set of *direct connections* between a peer and a subset of peers belonging to its *AoI* may be defined.

When a peer P receives an heartbeat from Q, it checks if one of its neighbors, say N, is entering the Area of Interest of Q, and in this case it notifies to N the position of Q. In this way, each peer acts as a *beacon* for each neighbor by putting it in touch with new peers approaching its Area of Interest from far away locations.

A similar approach may be adopted to notify the updates of passive objects. In this case, the notification is sent to each peer belonging to the *visibility area* of the object, which is generally a circular area centered in the object. These notifications may be forwarded via the same routing mechanism defined for heartbeats.

An important issue to be considered when adopting a fully distributed and dynamic approach is that it needs a set of mechanisms to face the inconsistencies which may rise due to network delays, loss of messages, or abrupt crashes of peers. To avoid network partitioning, the definition of a set of mechanisms is therefore mandatory.

Let us now consider the *Additive Weighted Voronoi* approach. The routing algorithms proposed in the previous chapters must be adapted to cope with the hierarchical structure of the network. A different strategy must be defined to route the notifications generated by a peer which are hidden by the Voronoi regions of other peers. These notifications should be sent to the *superpeer* SP, i.e. the peer which has 'absorbed' the hidden peer. SP, in turn, dispatches these notifications to its hidden peers which are interested in these notifications, and to its visible neighbors. It is worth noticing that peers which are physically close in the *DVE* may be hidden by different *superpeers*, like D and E in Figure 91. For this reason each notification received by a *superpeer* must be forwarded to its visible neighbors which, in turn, may propagate the notification to interested peers hidden in their regions. This basic mechanism can be improved by propagating on the overlay only notifications of hidden peers whose *AoI* intersect the border of the region owned by their *superpeer*.

It is worth noticing that each peer may turn its state from hidden to visible and the other way around, when moving within the *DVE*. Each peer is able to autonomously decide if it has been 'absorbed' by a *superpeer* by comparing its weight with those of its Voronoi neighbors and by evaluating the euclidean distance from them. When a peer is absorbed, it resets all its connections to visible peers and establishes a single connection with its *superpeer*.

On the other hand, a hidden peer P cannot autonomously decide if it has become visible, since it does not know any peer of the *DVE*, except its *superpeer* SP. In this case, when SP hides no more P, it notifies to P the change of its status. P receives from SP the list of its visible neighbors as well. Note that this operation is equivalent to a new join of the hidden peer to the Voronoi overlay.

6.5 Experimental Results

The model introduced in this chapter has been analyzed by means of simulations. The simulator was composed by stable and mainstream components. A mature library that provides Apollonius graph's creation and management is $CGAL$ (Computational Geometry Algorithms Library) (62), written in C++ and developed by a large consortium of European research institutions. The simulation infrastructure is Peersim (55). The softwares were linked together by means of a compatibility layer created using SWIG (Simplified Wrapper and Interface Generator) (63).

The experiments were aimed at the analysis of the topology of peer-to-peer networks based on Apollonius graphs, under two different scenarios. In the first one, that we called "constant weights scenario", the peers are divided into two groups, the weightless peers, with a weight equal to 0, and the weighty peers, having all the same weight p, that has been kept constant during each run of the simulation. The second scenario, the "dynamic weights scenario", also manages weightless and weighty peers, but the weight p has been modified from 25 to 100 during the simulation, that is composed of only one run.

In the rest of the chapter, we use the following definitions:

- **Weightless peer:** a peer that has a weight of 0
- **Weighty peer:** a peer that can have a weight different from 0

6.5.1 Constant weights scenario

The goal of the experiment was to measure the parameters that characterize peer-to-peer networks composed by active peers with different weights. The peers were 900 and were divided into two logical groups. The first group features 800 weightless peers, while the second group

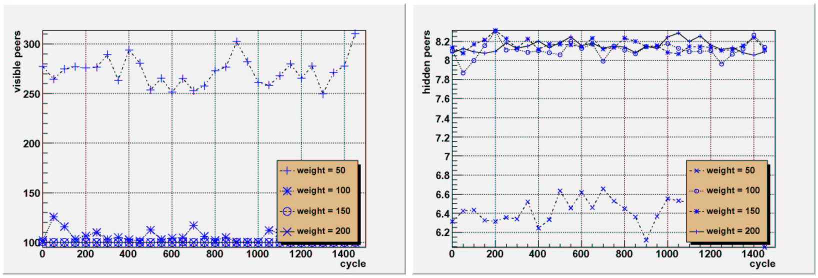


Figure 92: Number of visible peers (left) and mean number of hidden peers (right) for different weights.

is composed by 100 weighty peers characterized by the same positive weight p, that is assigned a different value in the different runs of the experiment. In particular, each experiment is composed by 4 runs, one with $p = 50$, then one with $p = 100$, then one with $p = 150$, and finally one run with $p = 200$. Peersim simulator performed 1500 cycles for each weight assignment, meaning that the experiment analyzes 1500 networks for each weight. The coordinates of the peers and of the passive objects are distributed uniformly at random at the beginning of each cycle.

Left side of Figure 92 shows the number of visible peers against the cycle number. Each line represents the outcome of the experiment for a different weight p. The figure shows that, when $p \geq 100$, only 100 peers (the weighty ones) are visible. This result is also confirmed by the right side of Figure 92, that shows the mean number of peers that are hidden by each weighty peer, against the cycle number. Each line represents the outcome of the experiment for a different weight p. The figure shows that, when $p \geq 100$, the mean gets very close to 8, meaning all the weightless peers got hidden by weighty peers.

6.5.2 Different numbers of weighty peers

This set of experiments analyzes the effects of having a small number of weighty peers in the peer-to-peer network. The simulated networks

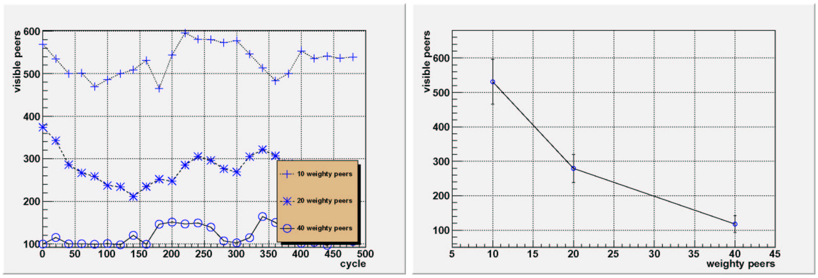


Figure 93: Number of visible peers (per cycle and mean value) for 10, 20 and 40 weighty peers.

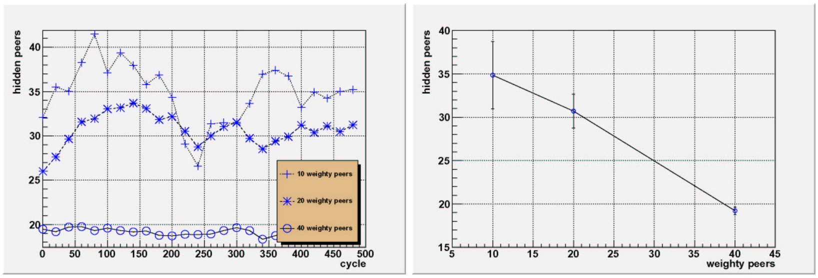


Figure 94: Number of hidden peers (per cycle and mean value) for 10, 20 and 40 weighty peers.

featured 900 peer, and the weight p of the weighty peers is 100, the number weighty peers dependeds on the run and is $n \in 10, 20, 40$, and 500 networks were simulated for each different number of weighty peers.

Left side of Figure 93 shows the number of visible peers during the simulation for different numbers of weighty peers, and it is summarized by the right side of Figure 93, that shows the mean number of visible peers against the number of weighty peers. The result is that, when there are 10, 20 and 40 peers, there are respectively $500 - 600$, $280 - 380$ and $100 - 150$ visible peers, hence the weighty peers hid respectively $300 - 400$ peers, $500 - 600$ peers, and $750 - 800$ peers.

Left side of Figure 94 shows the mean number of peers hidden by each weighty peer during the simulation for different numbers of weighty

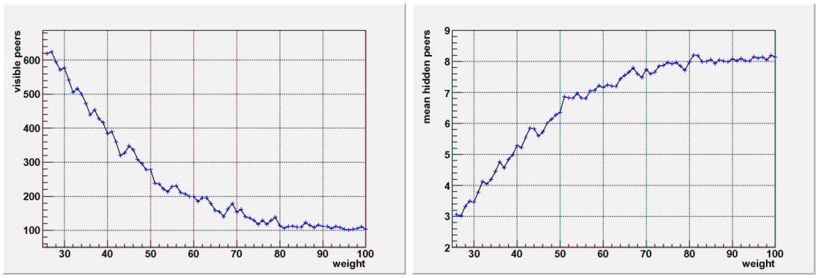


Figure 95: Number of visible peers (left) and number of peers hidden by each visible peer (right) against the weight of weighty peers.

peers, and it is summarized by the right side of Figure 93, that shows the mean number of peers hidden by each weighty peer against the number of weighty peers. The result is that, when there are 10, 20 and 40 peers, each weighty peer hids respectively $31-39$, $29-33$ and $19-20$ weightless peers.

6.5.3 Dynamic weigths scenarios

The goal of this part of the experiment has been to measure the parameters that characterize peer-to-peer networks composed by active peers that are divided into two logical groups. The peers were 900, the first logical group featured 800 weightless peers, while the second group was composed by 100 peers characterized by the same positive weight p, that is modified during the experiment. There are also 4000 passive objects in the area, to be managed by the active entities that owned the locations where the object resided. Each experiment involved 1500 cycles (say, 1500 different networks), and $p = 100$ for the first 20 cycles, then $p = 99$ for cycles $21 - 40$ and so on, down to $p = 26$ for cycles $1481 - 1500$. The coordinates of the peers and of the passive objects are distributed uniformly at random for each generated network, i.e. for each cycle.

Left side of Figure 95 shows the number of visible peers against the weight assigned to the weighty ones. It shows that, when $p \geq 80$, only the 100 weighty peers are still visible. This result is confirmed by the right side of Figure 95, that shows the mean number of peers that were hidden by each visible peer, against the weight assigned to the weighty peers. It shows that, when $p \geq 80$, the 800 weightless peers are hidden by the 100 weighty peers, and hence each visible (weighty) peer hids a mean of 8 hidden (weightless) peers.

Left side of Figure 96 shows the number of peers that own passive objects against the weight p of the weighty peers. It shows that, when $p \geq 80$, the 800 weightless peers are hidden by the 100 weighty peers, and hence only the 100 weighty peers own objects. On the other hand, when p is small, a larger number of peers, both weightless and weighty, own passive objects. The dual view of this result is provided in the right side of Figure 96, that shows the mean number of objects that are managed by each owner, against the weight p of the weighty peers. When $p \geq 80$, the 100 weighty peers are the only visible ones, hence they own all the 4000 passive objects, and the mean number of passive objects for each owner is $4000/100 = 40$. On the other hand, when p was small, a larger number of peers, both weightless and weighty ones, own passive objects, and hence the mean number of objects for each of them drops towards 5.

6.5.4 Number of links

Left side of Figure 97 shows the mean number of links between visible peers. The number of links does not vary when the weight is p differed.

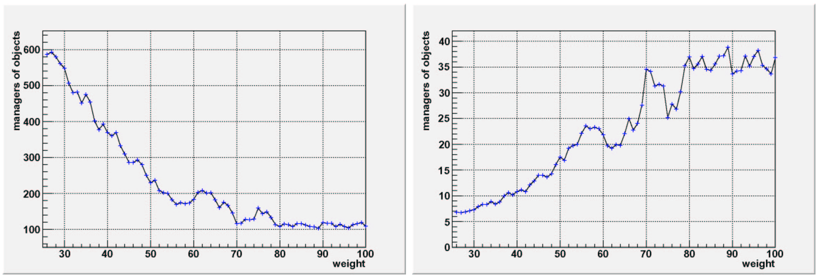


Figure 96: Number of peers that own passive objects (left) and mean number of passive objects per owner (right).

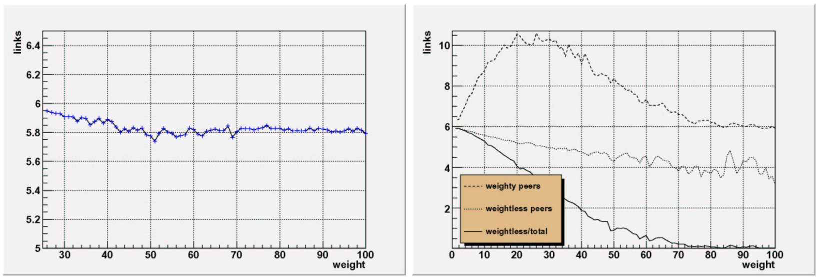


Figure 97: Number of links between visible peers: mean value (left) and discerning weightless and weighty peers (right).

More information about the number of links of visible peers is given by the right side of Figure 97, that discerns between weighty and weightless peers. For this Figure only, the experiment was slightly different from the previous ones. This time the number of weightless peers is 800, there are 100 weighty peers, the weight is $p \in [1, 100]$, and we simulated 20 networks for each different weight. The lines represent the number of links of a visible peer to different visible peers. In particular, the higher line shows the mean number of links from *weighty* peers to visible peers. The middle and bottom line both show the number of links from *weightless* visible peers to visible peers. The middle line shows the mean number of links from weightless peer to visible peers, normalizing it against the number of visible weightless peers, completely *ignoring* hidden weightless peers. On the other hand, the bottom line shows the mean number of links between weightless peers and visible peers, normalizing it against the number of all the weightless peers (both visible and hidden), *counting* hidden peers as having 0 links.

This experiment highlights a number of facts. First of all, the behaviors of the weightless and weighty peers converged when the weight p of the weighty peers got to 0, and in particular the number of links converged to 6, as it is the standard behavior for Voronoi graphs50. Moreover, the behavior of the weighty peers goes back to standard Voronoi when $p \geq 80$, meaning that the weightless peers were totally hidden by the weighty ones, and hence the remaining weighty peers created a

Voronoi network between them. An unexpected yet interesting behavior is the intermediate one, when the number of links of weighty peers goes over 10. Some weightless peers, for $p \in [10, 50]$, are still visible into the network, but have small areas assigned to them, hence the weighty peers still have their Voronoi links between them, but they have also some links to the weightless peers.

Chapter 7

Conclusions and Future Works

7.1 Conclusions

This thesis defines a *scalable approach* for the development of a P2P support for *Distributed Virtual Environments*. Our approach is based on a *Voronoi Tessellation* of the DVE which is exploited to define both the P2P overlay and the support for the management of the passive objects.

Our proposal differs from (1) in several directions. First of all, we have defined a protocol for heartbeat propagation which requires a minimal amount of information at each peer. As a matter of fact, each peer should be aware only of its Voronoi neighbors to forward a heartbeat within the AOI, while the routing algorithm presented in (52) requires the knowledge of the two hops away neighbors. For these reason our solution considerably reduces the number of messages exchanged through the overlay and the bandwidth requirement at each peer especially when crowding occurs.

Our routing algorithm is based on compass routing (60). Even if the general idea of exploiting compass routing to build a multicast tree has been presented in (59), at the best of our knowledge, our proposal is the first one where an algorithm is defined, implemented and evaluated.

Our proposal exploits the concept of *Area of Interest* which models the locality of interactions typical of the DVE. According to this approach, the maximum degree of consistency is obtained, for each peer, within the portion of the DVE within the AOI of the peer, while the consistency decreases when the distance from the peers increases.

A further important characteristic of our proposal is that when a peer moves within the DVE, the state of the areas not belonging but close to its AOI can be easily accessed in comparison to those far away from its AOI. As a matter of fact, our approach is based on a "pass the word" strategy, where the peers on the border of the AOI of a peer p "put in touch" p with new peers entering its AOI from outside. This mechanism is simply based on the propagation of the heartbeats and does not require a further request reply mechanism like that introduced in (1) which introduces further traffic on the overlay.

Furthermore the thesis presents a set of strategies for the management of the passive objects exploiting the properties of the *Voronoi Tessellation* to guarantee the *consistency* and the *persistency* of the state of the DVE. A few proposals (20; 23) for the management of the passive objects of the DVE have been presented in the literature and most of them exploit a DHT to assign the management of the passive objects to the peers. We believe that the cost of dynamically querying a DHT for objects retrieval is not tolerable in a DVE.

Both the heartbeat propagation protocol and that for the management of the passive objects are specified by *Mobile Unity*.

The thesis also proposes a *hybrid P2P architectures based* on *Additively Weighted Voronoi* graphs. In this architecture, each peer may have a different role in the P2P network according to its communication bandwidth, CPU power and memory capacity. Moreover, passive objects may be assigned to peers such that the load for their management *is balanced* among the peers.

The experimental results have shown the feasibility of our approach. First of all, we have checked the cost of maintaining a distinct Voronoi diagram at each peer including its AOI neighbors. We have tested the $VAST$ library and shown that the cost of rebuilding a Voronoi diagram at each heartbeat reception is tolerable with respect to the realtime constraints typical of this kind of applications.

Each proposed protocol has been tested and evaluated by *Peersim*, a scalable simulator for P2P networks. A set of preliminary evaluation have been also performed on the *GRID 5K platform*.

7.2 Future Work

Even if the definition of a P2P network for DVE is a challenging alternative to the classical *client server* solution, we are aware that several problems should be still be solved for the definition of a comprehensive solution. The main problems which have still to be solved concern, for instance, the authentication of the peers joining the DVE and the management of the DVE state when the number of peers is very low or zero. The main problem when a low number of peers belong to the DVE is related to the high load assigned to each peer. A further problem is the maintenance of the state of the DVE when all the peers have left it, because this state should be restored later when some peer joins the DVE.

To manage the problem of state persistency and of peer authentication, we have started to investigate an *hybrid solution* including a small number of "classical" servers controlling the state of the DVE and a huge amount of interacting peers. Note that this solution differs from that proposed in Chapter 6, because the set of servers is *statically defined*, while in the hybrid overlay proposed in Chapter 6, the *Superpeer* are dynamically elected, they participate to the DVE as normal peers and support the further task of routing the event notification for the peers they manage.

This section briefly sketches the resulting architecture which we plan to develop and test as future work.

In this solution, the server controls the join of the peer to the DVE, their authentication and manages a portion of the DVE state.

For the sake of simplicity, we consider a system where a single server S is defined. The server is a supervisor which does not belong to the P2P overlay and is connected to all the peers. When a peer enters the DVE or updates its position, it notify this event to the server so that

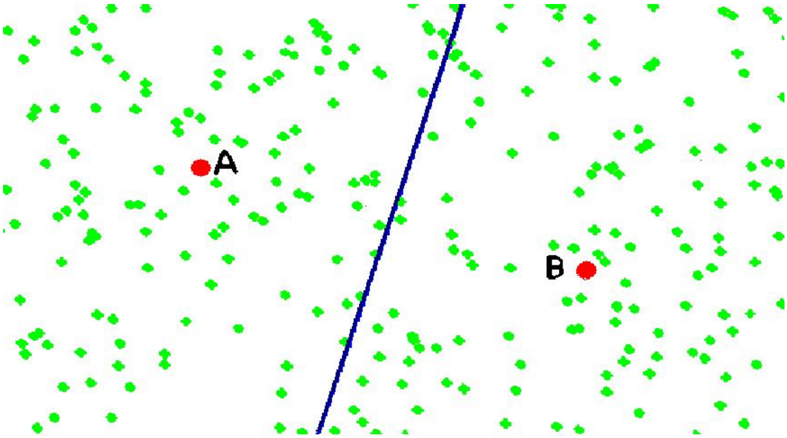


Figure 98: A few peer managing the DVE.

the server continuously has a vision of the whole DVE and is able to compute a Voronoi tessellation including all the peers of the DVE. In this way, it is able to distribute portions of the state of the DVE to the joining peers. On the other way round, the server does not forward the received notification to the interested peers, since these are directly exchanged between the peers. This avoid that the server becomes, like in classical client server solutions, a bottleneck for the entire system.

As shown in Chapter 5 a *Voronoi based tessellation* enables a natural mapping of the passive objects of the DVE to the peers where each object is mapped to the peer whose Voronoi region includes it. In this way, each object is managed by a single peer of the DVE. The problem of this approach is that, when the DVE includes a small number of peers, a large number of objects may be associated to a single peer which may be not able to manage all them. As a matter of fact, when a few peers are present in the DVE all objects are partitioned between them and a single peer may *result overloaded*, as shown in Figure 98 where only two peers manage the whole DVE:

To avoid peer overloading, we associate with each peer a new circular

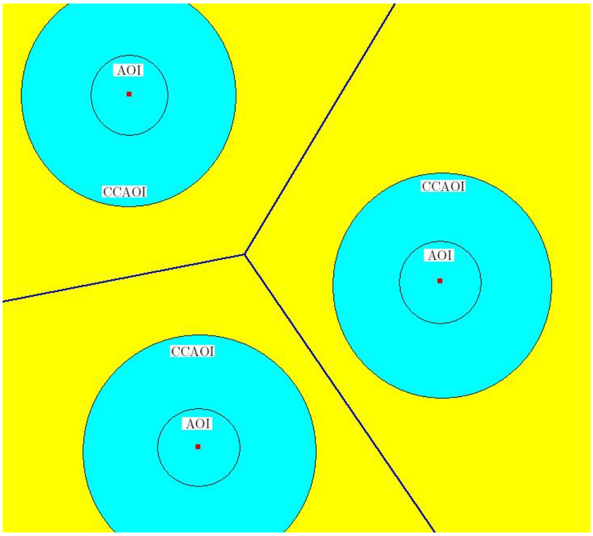


Figure 99: AOI and CCAOI.

area, the *CCAOI, Chain Coordination AOI*, centered at the peer and whose radius is larger than of the AOI. This area changes dynamically when the peer moves like the other areas introduced in the preceding sections. The goal of the $CCAOI$ is to reduce the number of objects assigned to a a peer when its Voronoi Area is too large. As a matter of fact, in this solution each peer manages the objects located inside the *Intersection Area, IA*, i.e. the area corresponding to the *intersection* between *Voronoi Area* and its $CCAOI$, while the objects located outside the *Interaction Area* of any peer are assigned to the server.

As Figure 99 shows, the area managed by the peer is only the blue one while the yellow area does not belong to any *Interaction Area* and is managed by the server.

The server initially owns the state of the whole DVE. When a peer joins the DVE, it first contacts S for the *authentication*, then it receives from S, and/or form its Voronoi neighbors the set of objects belonging to its Intersection Area.

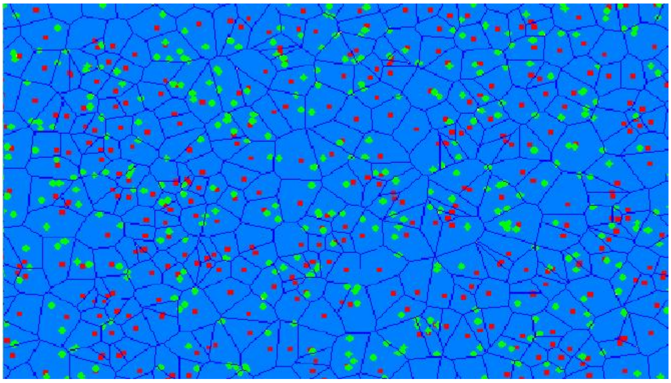


Figure 100: Each object is managed by a peer.

It is important to note that the IA overlaps the $CCAOI$ when the the $CCAOI$ is totally included in the Voronoi area of the peer. This corresponds to a scenario where a very small number of peers are present in the DVE. Note that in this case the *Voronoi Area* is much larger than the *Interaction Area*. In this scenario, the introduction of the *Interaction Area* enables each peer to take care only of the coordination of the closer objects, i.e. the objects located inside its IA while the server manage the objects located within its Voronoi Area, but not belonging to its Intersection Area, i.e. the objects located far from it.

Note that in this scenario the server does not become a bottleneck for the system, even if a large number of objects are mapped to it because of the presence of a few peers. As a matter of fact, the probability that the peers update the objects mapped to the regions managed by the server, i.e. the yellow regions in Fig. 99, is low, because these objects are located far away the peers. Note that as the number of peer decreases, the objects are assigned back to the server that, in a natural way, acquires the total control of the system, when the last peer leaves the DVE. In this case the server acts as a *backup server* for the DVE state, and when each peer exits the DVE, the state of the DVE will be stored by the server to be restored later. Furthermore, in this way the load of the peers is reduced.

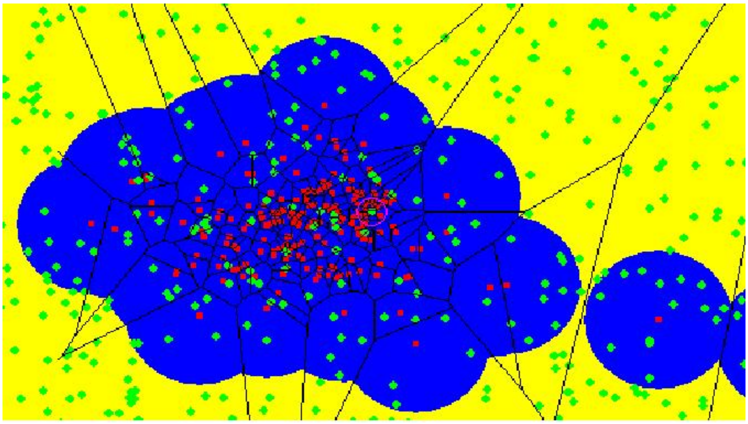


Figure 101: A Crowding Scenario

When the number of peers increases, the number of objects owned by the server decreases, because it delegates the management of the objects to the joining peers. In this scenario, as in a crowding situation, the *Interaction Area* of each peer may overlap its Voronoi region and the management of the state of the DVE may be delegated entirely to the peers, as shown in Figure 100, where the management of the DVE is totally delegated to the peers and the server owns no object. In this scenario the only task of the server is to control and authenticate the peer joining the network since all the objects are managed by the peers.

Consider now a crowding scenario, for instance one where peers fight against each other and a large number of peers is concentrated in a small portion of the virtual space, as showed in Figure 101. In this case, the Voronoi Area of each peer is included in its $CCAOI$, hence the IA of the peer overlaps its Voronoi Area and, despite the large number of peers, the area that the server must manage is very large. Even in this situation, the server does not become a bottleneck, because it does not receive updates for the objects it owns since they are located far away from the peers. Again its task is to store the state of the objects and to decrease the load of the peer.

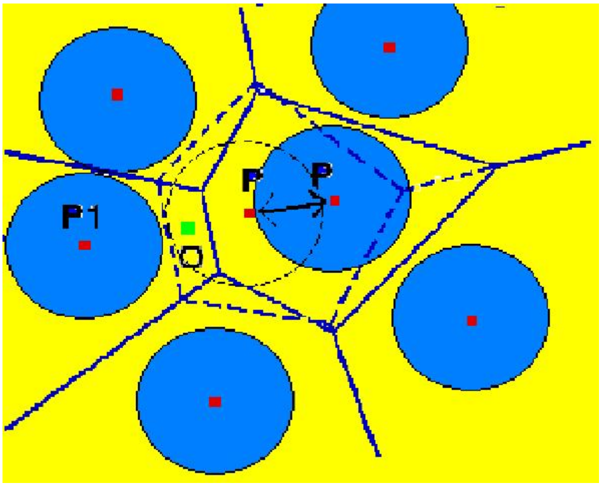


Figure 102: Object delegation between peer and the server.

In our solution, the server itself becomes, compared to the classical client server model, both a *backup* and a *load distribution* mechanism.

When a new peer enters the DVE or the overlay is modified due to the movement of the peers, the server checks if some object it owns falls within the IA of a peer and in this case it sends the object to this peer.

The peers instead have a limited knowledge of the DVE because of the local information obtained by the direct connections with peers that fall in their AOI and with their Voronoi neighbors. When a peer P receives information from a neighbor V about either a position update or a new neighbor notification, P rebuilds its local Voronoi diagram with the new neighbors positions and check whether any of the objects owned fall in the IA of its neighbors or in the server area. In both cases, P is no longer the owner of the object and sends the the object to the new owner.

For instance, in Figure 102, the peer **P** moves from left to right and the object **O**, first included in the IA(**P**) managed by **P**, because of the shift of **P**, enters the yellow area under server competence. If we observe the movement of **P**, from right to left, the object **O** would be assigned by the

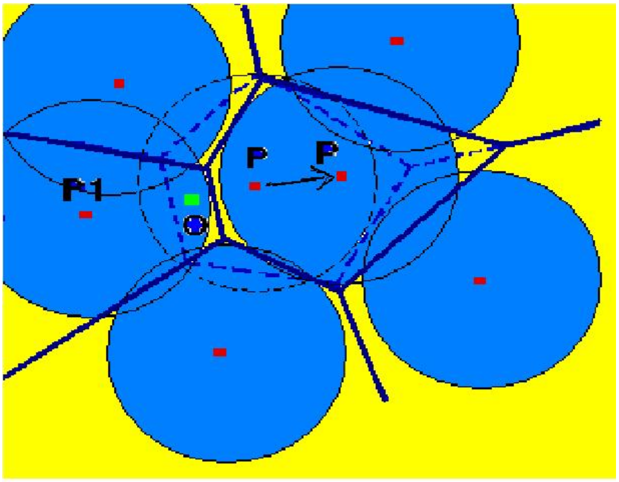


Figure 103: Object delegation between peers.

server to **P**.

In Figure 103 we see the exchange of an object between peer **P** and P_1. Dotted lines show the $CCAOI$ and the Voronoi region of **P** before its movement, when the object **O** just falls in $IA(P)$. When **P** moves following the arrow, then **O** enters $IA(P_1)$ and P_1 becomes the new owner of **O** by receiving from **P** all the information.

Further future improvements of our work concern the evaluation of our protocols on a real platform to evaluate their effectiveness in a real setting. Another open issue is the investigation of alternative Voronoi models, for instance the *Multiplicatively Weighted Voronoi graphs*.

Bibliography

[1] Shun-Yun Hu, Jui-Fa Chen, Tsu-Han Chen, *VON: A Scalable Peer-to-Peer Network for Virtual Environments*, IEEE Network, July-Aug. 2006 1, 23, 57, 58, 60, 89, 121, 156, 157

[2] Joaquin KELLER, Gwendal SIMON, *SOLIPSIS: A Massively Multi-Participant Virtual World*, Proc. Int. Conf. Parallel and Distributed Techniques and Applications (PDPTA 2003), CSREA Press, 2003 17, 19

[3] Joaquin KELLER, Gwendal SIMON, *Toward a Peer-to-Peer Shared Virtual Reality*, Proceedings of the 22nd International Conference on Distributed Computing Systems table of contents, 2002 2, 17, 19

[4] Joerg Eberspaecher, Ruediger Schollmeier, Stefan Zoels, Gerald Kunzmann, *Structured P2P Networks in Mobile and Fixed Environments* , AEÜ - International Journal of Electronics and Communications January, 2006 2, 12

[5] Ozalp Babaoglu, Hein Meling, Alberto Montresor, *Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems*, Proceedings: International Conference on Distributed Computing Systems 2

[6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, LNCS, Vol: 2009/2001 2, 9

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, Proceedings of ACM SIGCOMM, San Deigo, CA, August 2001. 10, 12

[8] Shun-Yun Hu, Guan-Ming Liao, *Scalable Peer-to-Peer Networked Virtual Environment*, Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games, Portland, Oregon, USA, SESSION: Novel techniques and cheat detection, 2004 23, 25

[9] Shun-Yun Hu, Jui-Fa Chen, Tsu-Han Chen, *A Forwarding Model for Voronoi-based Overlay Network*, VAST Technical Report (VAST-TR-2005-01), 2005 2, 23, 25

[10] David R. Karger, Matthias Ruhl, *Diminished Chord: A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks* , Congrs Peer-to-peer systems III, La Jolla, 26-27 February, 2004 2, 10, 12

[11] Ozalp Babaoglu, Hein Meling, Alberto Montresor, *Peer-to-Peer Document Sharing using the Ant Paradigm*, Proceedings of Norsk Informatikkonferanse (NIK), Troms, Norway, November 2001.

[12] Venkita Subramonian, Liang-Jui Shen, Christopher Gill Nanbor Wang, *The Design and Performance of Configurable Component Middleware for Distributed Real-Time and Embedded Systems*, Proceedings of the 25th IEEE International Real-Time Systems Symposium, 2004 2

[13] Ralf Steinmetz, Klaus Wehrle, *Peer-to-Peer Systems and applications*, LNCS, N.3485, State of the art survey. 8, 9, 12

[14] Shang-Wen Cheng, David Garlan, Bradley Schmerl, Joo Pedro Sousa, Bridget Spitznagel, Peter Steenkiste, Ningning Hu, *Software Architecture-based Adaptation for Pervasive Systems*, International Conference on Architecture of Computing Systems (ARCS'02): Trends in Network and Pervasive Computing, April 8-11, 2002

[15] David Garlan, Bradley Schmerl, and Jichuan Chang, *Using Gauges for Architecture-Based Monitoring and Adaptation*, In the Working Con-

ference on Complex and Dynamic Systems Architecture, Brisbane, Australia, 12-14 December, 2001.

[16] Shang-Wen Cheng, David Garlan, Bradley Schmerl, Peter Steenkiste, Ningning Hu, *Software Architecture-based Adaptation for Grid Computing*, Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02), 2002 2

[17] Stefan Saroiu, Krishna P. Gummadi, Steven D. Gribble, *Measuring and analyzing the characteristics of Napster and Gnutella hosts*, Multimedia Systems archive Volume 9, Issue 2, August, 2003 2, 8, 9

[18] Antony Rowstron, Peter Druschel2, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November 2001 10, 12

[19] A. Bonotti, L. Genovali, L. Ricci, *DiVES: A Distributed Support for Networked Virtual Environments*, Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1, AINA, 2006 28

[20] B. Knutsson et al. *Peer-to-peer support for massively multiplayer games*, Proc. INFOCOM, Mar. 2004 157

[21] 28] T. Iimura, H. Hazeyama, and Y. Kadobayashi, *Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games*, Proc. ACM SIGCOMM Wksp. on NetGames, Aug. 2004

[22] Y. Kawahara, T. Aoyama, and H. Morikawa, *A peer-to-peer message exchange scheme for large-scale networked virtual environments*, Telecomm Sys. vol.25, 2004.

[23] A. Yu and S. T. Vuong, *MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games*, Proc. NOSSDAV, Jun. 2005 14, 157

[24] A. Goldin and C. Gotsman, *Geometric Message-Filtering Protocols for Distributed Multiagent Environments*, Presence, 2004

[25] A. Steed and C. Angus, *Supporting Scalable Peer to Peer Virtual Environments Using Frontier Sets*, Proc. IEEE Virtual Reality, Mar. 2005

[26] S.Rooney, D. Bauer, and R. Deydier *A federated peer-to-peer network game architecture*, IEEE Commun. Mag. 2004

[27] Samuel Madden, Michael J.Franklin, and Joseph M.Hellerstein Wei Hong, *TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks*, Appearing in5th Annual Symposium on Operating Systems Design and Implementation (OSDI). December, 2002

[28] Ivan Vaghi, Chris Greenhalgh, Steve Benford, *Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments*, Proceedings of the ACM symposium on Virtual reality software and technology, London, United Kingdom, 1999

[29] Indranil Gupta, Robbert van Renesse, Kenneth P.Birman, *Scalable Fault-Tolerant Aggregation in Large Process Groups*, Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS), 2001

[30] Nicolas Bouillot, Eric Gressier-Soudan, *Consistency models for distributed interactive multimedia applications*, ACM SIGOPS Operating Systems Review archive Volume 38, October, 2004 37

[31] Leslie Lamport, *Time, clocks, and the ordering of events in a distributed system*, Communications of the ACM, Volume 21 , July, 1978 37

[32] Francisco J. Torres-Rojas, Mustaque Ahamad, Michel Raynal, *Timed consistency for shared distributed objects*, Annual ACM Symposium on Principles of Distributed Computing archive Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, 1999

[33] Francisco J. Torres-Rojas, Esteban Meneses, *Convergence Through a Weak Consistency Model: Timed Causal Consistency*, 30ma Conferencia Latinoamericana de Informtica, CLEI, 2004

[34] Michel Raynal, *From Causal Consistency to Sequential Consistency in Shared Memory Systems*, LNCS, Vol. 1026, Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science, 1995 37

[35] Suiping Zhou, Wentong Cai, Bu-Sung Lee, and Stephen J. Turner, *Time-Space Consistency In Large-Scale Distributed Virtual Environments*, ACM Transactions on Modeling and Computer Simulation (TOMACS), Volume 14, January 2004

[36] Eric Cronin, Burton Filstrup, Sugih Jamin, *Cheat-Proofing Dead Reckoned Multiplayer Games (Extended Abstract)*, Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04: Network and system support for games, Portland, Oregon, USA, SESSION: Novel techniques and cheat detection, 2004 95

[37] Laurent Gautier, Christophe Diot, Jim Kurose, *End to end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet*, Proceedings of the Conference on Computer Communications (IEEE Infocom), (New York), Mar. 1999

[38] Bu-sung Lee Wentong Cai Stephen J. Turner L. Chen, *Adaptive Dead Reckoning Algorithms For Distributed Interactive Simulation*, Proceedings of the thirteenth workshop on Parallel and distributed simulation, Atlanta, Georgia, United States, 1999 95

[39] Lothar Pantel, *On the Suitability of Dead Reckoning Schemes for Games*, Network and System Support for Games, Proceedings of the 1st workshop on Network and system support for games, Bruanschweig, Germany, 2002 95

[40] Ivan Vaghi, Chris Greenhalgh, Steve Benford, *Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments*, Pro-

ceedings of the ACM symposium on Virtual reality software and technology, London, United Kingdom, 1999

[41] Frank Adelstein, and Mukesh Singhal, *Real-time causal message ordering in multimedia systems*, Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95), May 30-June 02, 1995

[42] Takayuky Tachikawa, Makoto Takizawa, Δ-*Causality in Wide-Area Group Communications*, International Conference on Parallel and Distributed Systems (ICPADS '97), IEEE Computer Society, Seoul, Korea, 11-13 December 1997

[43] R. Baldoni, R. Prakash, M. Raynal, and M. Singhal, *Efficient Delta-Causal Broadcasting*, International Journal of Computer Systems Science and Engineering, September, 1998.

[44] David L. Mills, *Network Time Protocol (Version 3) Specification, Implementation and Analysis, Request for Comments: 1305*, RFC 1305, University of Delaware, March, 1992 130

[45] David L. Mills, *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI, Request for Comments: 2030*, RFC 2030, University of Delaware, October, 1996 130

[46] V. Krishnaswamy, M. Raynal, D. Bakken, and M. Ahamad, *CShared State Consistency for Time-Sensitive Distributed Applications*, Proc. of the Intl. Conference on Distributed Computing Systems, April, 2001 130

[47] F.Baiardi, A.Bonotti, L.Genovali, L.Ricci,*A publish subscribe support for networked multiplayer games*, Internet and Multimedia Systems and Applications (EuroIMSA 2007) Chamonix, France, IASTED Press, March 14-16, 2007

[48] Bjorn Knutsson, Honghui Lu, Wei Xu, Bryan Hopkins, *Peer-to-Peer Support for Massively Multiplayer Games*, INFOCOM 2004, Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, 7-11 March, 2004 15

[49] R. Baldoni, R. Prakash, M. Raynal, M. Singhal, *Efficient Δ-Causal Broadcasting*, Journal of Computer System Science and Engineering, 1998.

[50] http://www.cs.utah.edu/flux/papers/bees-openarch03-base.html

[51] https://jmephysics.dev.java.net/

[52] Jehn-Ruey Jiang, Yu-Li Huang, and Shun-Yun Hu, *Scalable AOI-Cast for Peer-to-Peer Networked Virtual Environments*,28th Int. Conference on Distributed Computing Systems Workshops(ICDCSW), June, 2008 25, 26, 58, 60, 120, 156

[53] Jui-Fa Chen, Wei-Chuan Lin, Tsu-Han Chen, and Shun-Yun Hu,*A Forwarding Model for Voronoi-based Overlay Network*, 13th ICPADS 2007, P2P-NVE workshop, December, 2007 25, 58

[54] J.Lee et al. *APOLO:Ad hoc Peer to Peer Overlay Network for Massively Multi-player Online Games,* Technical Report, KAIST, 2006 21

[55] The Peersim Simulator http://peersim.sourceforge.net/ 92, 149

[56] F.Aurenhammer, *Voronoi Diagrams-A Survey of a Fundamental Geometric Data Structure*. ACM Computing Surveys, Vol 23, September, 1991 3, 5, 23, 42, 61, 142

[57] L.Ricci, A.Salvadori, *Nomad: Virtual Environments on P2P Voronoi Overlays*, 1th Int. Work. on Peer to Peer Networks (PPN 2007), Vilamoura, LNCS, Vol. 4803, November, 2007 1, 57, 58

[58] M.Albano, M.Baldanzi, R.Baraglia, L. Ricci, *VoRaQue: Range Queries on Voronoi Overlays* 13th IEEE ISCC, Marrakesh, July, 2008 74

[59] J.Liebeherr, M.Nahas, *Application Layer Multicast with Delaunay Triangulations* IEEE Journal on Selected Areas in Communications 40(8), October, 2002 65, 120, 156

[60] E.Kranakis, H.Singh, J.Urrutia *Compass Routing on Geometric Networks* 11th Can. Conf. on Computational Geometry, CCCG 99, August, 1999 65, 66, 73, 120, 156

[61] J.Jiang, Y.Huang, S.Hu, *Scalable AOI-Cast for Peer-to-Peer Networked Virtual Environments* 28th ICDCSW, June, 2008 68

[62] Computational Geometry Algorithms Library, http://www.cgal.org/ 149

[63] Simplified Wrapper and Interface Generator, http://www.swig.org/ 149

[64] J.Urrutia. *Routing with guaranteed delivery in geometric and wireless networks Handbook of Wireless Networks and Mobile Computing*, 2002 4, 147

[65] L.Genovali, L.Ricci, *AOI-Cast Strategies for P2P Massively Multiplayer Online Games.*, 5th IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME'09) Las Vegas, Nevada, USA, January, 2009 1, 147

[66] L.Cardelli and Andrew D.Gordon. *Mobile Ambients.* Theoretical Computer Science, Special Issue on Coordination, D. Le Mtayer Editor. Vol 240/1, June, 2000 43

[67] Peter J.McCann, Gruia-Catalin Roman *Compositional programming abstractions for mobile computing.*, Software Engineering, IEEE Transactions on Volume 24, Feb, 1998 43, 48

[68] http://vast.sourceforge.net/VON/ 92

[69] J.Liebeherr, M.Nahas *Application-layer multicast with Delaunay triangulations.*, Global Telecommunications Conference, 2001. GLOBECOM apos;01. IEEE Volume 3, 2001 58, 64, 66, 67, 79

[70] K.Many Chandy, Jayadev Misra *Parallel Program Design: A Foundation*, Addison Wesley, New York, 1988 43

[71] L.Genovali, L.Ricci, *JaDE: A JXTA Support for Distributed Virtual Environments*, 13th IEEE Symposium on Computers and Communications Program, Marrakesh, July, 2008 1, 35

[72] http://www.wow-europe.com/en/index.xml 1

[73] Dinesh C.Verma *Content Distribution Networks*, by John Wiley and Sons, Inc. 2002 7

[74] *Computer Supported Cooperative Work (CSCW)*, The Journal of Collaborative Computing Editor-in-Chief: Kjeld Schmidt 7

[75] http://www.hitl.washington.edu/projects/knowledge_base/distvr/ 1, 7

[76] http://secondlife.com/ 1