MOPAR: A Mobile Overlay Peer-to-Peer Architecture for Scalable Massively Multiplayer Online Games

by

Peiqun Yu

B.Sc., The University of British Columbia, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

January 2006

© Peiqun Yu, 2006

Abstract

We propose a fully distributed peer-to-peer (P2P) infrastructure supporting networked virtual environment (NVE) applications, such as massively multiplayer online games (MMOGs). While P2P computing is commonly recognized as a useful architecture for improving the scalability of MMOGs, it is still challenging to have a truly scalable system without compromising reliability, responsiveness, consistency, and low overhead. We propose a hybrid infrastructure – A Mobile Overlay Peer-to-Peer Architecture for Scalable Massively Multiplayer Online Games (MOPAR), to address this scalability issue of MMOGs. Our approach exploits the concept of interest management, taking advantage of both a structured overlay (i.e., distributed hash table (DHT)), and unstructured P2P architecture. Our infrastructure is not only more scalable and reliable than other approaches; it provides the benefits of high responsiveness, and low overhead. In this thesis, we present a novel hierarchical architecture and associated algorithms to alleviate the workload of each peer, save network bandwidth, and reduce overhead cost. We also present a novel zoning scheme that guarantees all players have a continuous view. We anticipate that our infrastructure will provide a generic, configurable, and efficient framework that can be used to facilitate user-designed P2P MMOGs or NVE applications.

Contents

Abstract ii

Contents iii

List of Tables vii

List of Figures viii

Acknowledgments x

Chapter 1 Introduction 1

1.1 Motivation 2

1.2 Thesis Contributions 4

1.3 Thesis Organization 6

Chapter 2 Background and Related Work 7

2.1 Peer-to-peer Infrastructure 7

2.1.1 Background 7

2.1.2 Distributed Hash Table and Pastry 8

2.2 Related Work 10

2.2.1 DHT-Based Approach 11

2.2.2 Unstructured P2P-Based Approach 12

2.2.3 Hexagonal Zoning 14

Chapter 3 System Architecture 16

3.1 Hexagonal Zoning 16

3.1.1 AOI Radius and Hexagonal Cells 17

3.1.2 Mapping Positions to Hexagonal IDs 18

3.2 Hierarchical Structure Design 19
3.2.1 Motivation 19
3.2.2 Design Details 21
3.3 Handling the Dynamic Characteristics 25
3.4 Direct Communication scheme 26
3.4.1 Inter-master Communications 27
3.4.2 Master-slave Communications 27
3.5 Fault Tolerance 28
Chapter 4 Protocol and Algorithms 30
4.1 Procedures and Protocol 30
4.1.1 Joining Procedure 31
4.1.2 Moving Procedure 32
4.1.3 Leaving Procedure 35
4.1.4 Protocols 35
4.2 Neighbouring Masters Searching Algorithm 37
4.2.1 Blind Searching Algorithm 37
4.2.2 Intelligent Masters Searching Algorithm 38
4.3 Overhead Costs 42
4.4 System Analysis 44
4.4.1 Advantages 44
4.4.2 Comparison with other schemes 46
Chapter 5 System Implementation 48
5.1 System Design 48

5.1.1 Architecture 48
5.1.2 Major Components 50
5.2 Implementation Details 51
5.2.1 Core Classes 51
5.2.2 Collaboration Diagrams 52
5.3 Visualization 55
Chapter 6 Evaluation 58
6.1 System Scalability 59
6.1.1 Master Workload 59
6.1.2 Overall Bandwidth Consumption 61
6.1.3 Overall Overhead Cost 62
6.2 Intelligent Masters Searching Algorithm 64
6.3 Impact of Varying AOI Radius 65
6.3.1 Impact on Master Workload 65
6.3.2 Impact on Overall Bandwidth Consumption 66
6.3.3 Impact on Overall Overhead Cost 67
6.4 DHT Performance 68
6.4.1 Home Node Searching Cost 68
6.4.2 Home Node Transition rate 69
6.5 Topological Consistency 70
6.5.1 Impact of Masters Initialization 71
6.5.2 Impact of Cell Switching 72
Chapter 7 Conclusions and Future Work 74

7.1 Summary 74
7.2 Future Work 75
Appendix A Derivations 77
A.1 Expected Number of Players per Hexagon 77
Bibliography 79

List of Tables

Table 3-1 Mathematical relationships between AOI radius and a hexagon 17

Table 3-2 Roles of nodes in the hierarchical structure 22

Table 4-1 Variable descriptions 31

Table 4-2 Basic function descriptions 31

Table 4-3 Protocols for MOPAR 36

Table 4-4 Notations in intelligent masters searching algorithm 39

Table 4-5 Comparisons of existing systems 47

Table 5-1 MOPAR layer descriptions 49

Table 5-2 API of MOPAR infrastructure 50

Table 5-3 Major components of the MOPAR infrastructure core 50

Table 6-1 Overhead cost vs. discovery cost 63

List of Figures

Figure 2-1 Looking up an object in Pastry overlay network....9
Figure 2-2 Map is partitioned into fixed regions....11
Figure 2-3 Fails to recover when adjacent nodes leave simultaneously....14
Figure 2-4 Moving between cells joins and leaves the same number of cells....14
Figure 3-1 Mathematical structure of hexagons....17
Figure 3-2 Graphical relationship between AOI and hexagonal cell....18
Figure 3-3 Dividing the game map into hexagonal cells....19
Figure 3-4 Left groups are separated from the right groups....20
Figure 3-5 Connection between master nodes....21
Figure 3-6 Relationship between home nodes and cells....22
Figure 3-7 Global connectivity guaranteed....25
Figure 3-8 Home node transfer process....26
Figure 3-9 Direct inter-master and master-slave communications....27
Figure 4-1 Moving scenarios....32
Figure 4-2 Pseudo code for moving procedure....33
Figure 4-3 Pseudo code for switching cell procedure....34
Figure 4-4 Blind searching....38
Figure 4-5 Cell numbering for describing searching algorithm....39
Figure 4-6 Intelligent master searching algorithm....40
Figure 4-7 Entering as a master or a slave....43
Figure 4-8 Players have continuous viewing area....45

Figure 5-1 MOPAR layered architecture 49
Figure 5-2 Collaboration diagram for joining procedure 53
Figure 5-3 Collaboration diagram for moving procedure 54
Figure 5-4 Collaboration diagram for leaving procedure 55
Figure 5-5 Hierarchical structure and neighbour discovery 56
Figure 5-6 Home node of a hexagonal cell 57
Figure 6-1 Master node bandwidth consumption 59
Figure 6-2 Average number of neighbouring masters 60
Figure 6-3 Effect of growth on overall bandwidth consumption 62
Figure 6-4 Growth of overhead cost 62
Figure 6-5 Ratio of overhead cost vs. discovery cost 63
Figure 6-6 Intelligent neighbouring masters searching algorithm 64
Figure 6-7 AOI radius and master workload 65
Figure 6-8 AOI radius and overall bandwidth consumption 66
Figure 6-9 Average number of inter-master connections 67
Figure 6-10 AOI radius and overall overhead 67
Figure 6-11 Average numbers of hops to reach the home nodes 69
Figure 6-12 Home nodes transition rate 70
Figure 6-13 Effect of the delay of master start-up on topological consistency 71
Figure 6-14 Effect of cell switching on topological consistency 72

Acknowledgments

I would like to thank to my supervisor, Dr. Son Vuong, for his inspiration, guidance and encouragement. It would have been impossible for me to get this far without his support. I am also grateful to Dr. Charles Krasic for being my second reader; his useful comments have improved this thesis.

Thanks to my colleagues in NIC lab, especially Juan Li, Mike Blackstock, Kamran Malik, Jun Wang, Xin Liu, Craig Wilson, Christian Chita, and Gary Huang, for their useful discussions and comments.

Finally, I would like to thank my family members for their endless love, and especially my wife, Haibo Yan, for her understanding and unlimited support.

Peiqun Yu

The University of British Columbia

December 2005

Chapter 1

Introduction

Massively multiplayer online games (MMOGs) are games where typically thousands or even millions of players interact in real time in a shared game world. MMOG is considered to be an application of large-scale simulations or networked virtual environments (NVE), since they share many characteristics of these systems [27]. In an MMOG system, game players, through their game entities such as avatar, share a sense of virtual space, game states and a sense of time in a virtual reality. We have seen many of these kinds of large-scale NVE applications being developed and used in areas ranging from military training, to education [11], to entertainment [10].

A successful MMOG must meet the following basic requirements [25]: (1) scalability – the system must be scalable to a virtually unlimited number of players; (2) responsiveness – latency is always a concern in a real-time system, although different type of games may have different latency tolerances; (3) persistency – game states should persist between game sessions; (4) reliability – games should be fault tolerant in the event of a sudden network failure; and (5) consistency – game states shared among the players should be consistent to a certain extent to allow a meaningful interaction.

Generally speaking, among all these requirements for a successful MMOG, scalability is considered to be the most challenging issue, since the number of players can grow in an unlimited and unpredictable fashion. Improving scalability without compromising the other requirements is the design goal of a successful MMOG.

1.1 Motivation

The client-server model is the most widely used network architecture in commercial MMOGs. For example, one of the most famous massively multiplayer online games EverQuest [10] falls in this category. The client-server model simplifies resolution of issues such as data persistency, states synchronization or consistency, security, and account management. It is able to do so, because the server provides the central control for the whole system, and is responsible for computing and persisting game states, synchronizing game events, updating clients with current game states, and managing player accounts. Clients are only involved in the games through sending commands to the server and updating their cached states based on messages from the server. More specifically, when the server receives a command from a client, it will re-compute the global game states based on the command, and then update those clients that are affected by the new game states (i.e., the new states fall within these players' area of interest (AOI)).

However, the traditional client-server model suffers from scalability problems. As it is hard to predict the number of players when a new game service is introduced, it is difficult for a company to decide on the appropriate server investment. Overestimating the number of players could add unnecessary cost, while underestimating could lead to revenue loss. The servers can also become the bottleneck point when a large number of players are involved in the game concurrently. As a result, clients may notice poor responsiveness, which degrades the performance of the games, especially those games requiring frequent updating (5-10 times per second), such as fast-paced first-shooter

games. In addition, the servers can also become the single point of failure if a sudden network failure occurs at the server side, which can cause the whole game system to crash. To address these issues, especially the scalability issue, a variant of the traditional client-server model, the clustered-servers model, has been introduced. In the clustered-servers model, such as Terazona [32], and Butterfly [6], multiple servers are connected to each other to serve the clients. These systems tend to be more scalable to the number of players, since servers can be added to the cluster to accommodate the potential growth of the game players. However, the downsides of this approach are that it increases system complexity, as well as introducing extra hardware and maintenance costs. Another major shortcoming of all client-server systems is that they can only support limited types of games, which greatly reduces their reusability.

Recently, P2P has been investigated as a way of addressing the scalability issues of the client-server model, in light of its natural properties such as resources sharing and decentralized resource consumption. Many proposals [4][5][18][19][22][25][29][31] have advocated taking advantage of the benefits of P2P architecture. They are based on the idea that every node joining the system will bring more resources to the virtual environments, so that with more users the game has more resources.

While P2P seems to address the scalability issues of MMOGs, it creates some issues that can be easily resolved in the client-server model. More specifically, in MMOGs, each player has limited virtual visibility, for instance, the area within 100 meters of distance. Therefore, broadcasting relevant messages to all players in the virtual world is definitely not bandwidth efficient, and wastes the computing power of those nodes that receive irrelevant messages; each player is usually only interested in the activities happening within his or her area of interest. Interest management [20] is commonly used in the large-scale virtual environments. Network bandwidth consumption is thereby greatly reduced, and the workload of each player is also alleviated. In traditional systems, interest management is easily achieved through server-

side filtering. Before P2P computing can be used to effectively solve the scalability issue, the issue of interest management must be addressed. Specifically, the problem is how a player must be able to dynamically build direct connections with players coming into his or her AOI without the intervention of the server.

1.2 Thesis Contributions

This thesis proposes a peer-to-peer (P2P) infrastructure that facilitates a scalable, responsive, and fault tolerant MMOG. It focuses on addressing the most challenging and common issue of MMOGs – scalability, through exploiting the concept of interest management. However, our framework is generic enough to be easily extended to meet more game-specific requirements, instead of putting them all in the core of the infrastructure. We hope that our approach can be used as a generic P2P games network foundation for the MMOG community, enabling numerous user-defined P2P MMOGs games to be built on top of it.

Our thesis makes the following contributions[1]:

- We propose a novel P2P interest-management scheme, which allows peers to detect neighbours coming into their area of interest. This generic P2P interest management scheme is not restricted to the MMOG. It can also be applied to other neighbour-discovery-related P2P applications.
- We propose a novel location-aware hierarchical architecture providing the benefit of low overhead. Only close-by peers will be grouped, so that the structure is condensed and message exchanges are efficient.
- We propose a hybrid infrastructure that takes advantage of the benefits of structured P2P architecture (distributed hash table (DHT)), including

[1] A paper based on the work of this thesis has been accepted for publication [1]

reliability, self-organizing ability, and proximity-awareness, and of the direct communication scheme of unstructured P2P architecture. Basically, DHT is used to build the hierarchical architecture, while message exchanging is based on the direct communication scheme.

- We present a novel hexagonal zoning scheme, in which a game world is divided into hexagonal zones. Using our special design, players have a continuous viewing area, instead of discrete views as in other similar zoning schemes.
- We design an intelligent coordinate (master) nodes searching algorithm, which significantly saves bandwidth consumption, and improves responsiveness by pruning many unnecessary DHT queries.
- We design a framework that provides the essential functionalities and protocols for user-defined games to be deployed by integrating the game applications into our infrastructure. Game logic is separated from the underneath infrastructure, so that there are no limitations on any user-defined games.
- The infrastructure is configurable to the size of the area of interest, and the size of the map dimension. The infrastructure can be configured to fit the needs of various games.
- We implement a prototype of the infrastructure to demonstrate its feasibility.
- We implement a simple game application and a visualization tool for graphically revealing the design of our architecture.
- We evaluate the performance of the infrastructure by experimenting with the prototype.

1.3 Thesis Organization

The thesis consists of seven chapters. In Chapter 2, we discuss background information and related work in P2P MMOGs. In Chapter 3, we present the high-level details of our hierarchical architecture design. In Chapter 4, we discuss the details of the algorithms we designed for dynamic interest management and the protocols developed for the communications between peers. In Chapter 5, we introduce the implementation details of the prototype, game simulation, and visualization. Chapter 6 presents the empirical analysis of the system's performance. In Chapter 7, we present the conclusions, and discuss potential directions for future research.

Chapter 2

Background and Related Work

This chapter introduces the background of P2P infrastructure and one of the most popular structured P2P overlays – Pastry – as well as some recently proposed P2P architectures for MMOGs.

2.1 Peer-to-peer Infrastructure

2.1.1 Background

Peer-to-peer systems are distributed systems in which each node has equal status and control. One major characteristic of the P2P model that distinguishes it from the traditional client-server model is that every node in the system has dual roles. That is, a node acts as a server when it provides resources, and as a client when it consumes resources. The P2P model has become well-known for some of its famous file-sharing applications such as Napster [23], Gnutella [14], and FreeNet [12].

While the benefits of P2P computing are well realized in file-sharing applications, many attempts have been made to apply the technology to the applications other than file

sharing, for example, MMOG. P2P computing has many realized benefits, including cost sharing, scalability and reliability, resources aggregation, increased autonomy, anonymity, and ad-hoc connectivity. Among these benefits, two are very useful in solving some of the challenging issues in the traditional client-server based MMOG. The first is cost sharing. Instead of investing a large amount of money on a single server, the P2P approach helps to spread costs over all the peers. The other benefit of the P2P architecture is its scalability and reliability. Adding a new player will not use up centralized resources; instead, resources are increased with the addition of the new player. Moreover, because the workload is shifted from a centralized server and distributed onto peers, server problems such as bottleneck and the single point of failure can be eliminated.

2.1.2 Distributed Hash Table and Pastry

In recent years, the overlay based on the distributed hash table (DHT) has become popular in the domain of P2P applications. Many proposals fall into this category, such as CAN [26], Chord [16], Pastry [3], and Tapestry [7]. The DHT-based P2P overlay is also known as a structured P2P infrastructure, and as such is completely distributed, scalable, and self-organizing.

Among all the variants of the DHT-based overlay, Pastry is the most widely used. Several applications have been built on top of Pastry, including a global, persistent storage utility called PAST [2], and a multicast infrastructure called SCRIBE [21].

Each node or application object in the Pastry overlay network is assigned a 128-bit node identifier by applying a random hashing algorithm (i.e., SHA-1) on the node's IP address, or the object's name to get the node_id or object key. The Pastry nodes are organized in a circular node_id space ranging from 0 to $2^{128}-1$, where the node_id is used to identify the position of each node in the circular ring. A distributed hash table is maintained in each node as a routing table. A Pastry node also maintains a set of Pastry

nodes that are numerically closest to it, called a leaf set. An application object is then stored in a determined Pastry node, where the node' node_id is closest to the object key. The object now can be looked up by any nodes in the space, as long as the object key is given. Figure 2-1 gives a graphical explanation of how an object is looked up in a Pastry overlay network. We suppose that an object with the object key as "d46a1c" is stored in the node "d467c4" (because the node_id and the object key are numerically closest), and that a lookup message containing the object key was sent out from the node "d5a1fc". The lookup message is routed to the destination node through three intermediate hops. Every intermediate node looks into its routing table and then forwards the message to the node that has the longest prefix match with the lookup message. The message finally reaches the determined destination node "d467c4", because this node is numerically closest to the lookup message. The actual object stored by this key can then be retrieved by the requesting node.

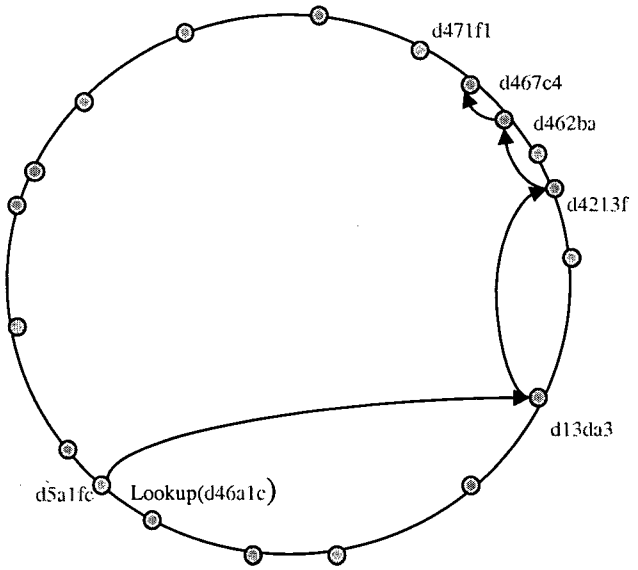


Figure 2-1 Looking up an object in Pastry overlay network

Pastry guarantees that messages are forwarded to their destination within $\log_{2^b}^{N}$ routing hops, due to the longest prefix matching, where N is the number of nodes, and b is a configuration parameter. Pastry also takes the locality into account, which meaning it selects the closest node with the minimum distance to forward the messages. The

distance is based on a scalar proximity metric like the number of IP routing hops. Pastry is also highly reliable, even in the event of concurrent node failure. A message is guaranteed to be delivered, unless $l/2$ of the adjacent nodes on the ring fail simultaneously. "l" is the size of the leaf set. The possibility of this occurrence is very low, due to the diversity of nodes with adjacent node_ids. This diversity results from the random hash algorithm used. As the node_id is generated by hashing the IP address using a random hashing algorithm, there is no direct relationship between the closeness of the nodes in the network and the closeness of their node_ids.

2.2 Related Work

The concept of interest management assumes that players in the games have a limited visibility or area of interest (AOI). Therefore, changes to game states outside of a player's AOI can be considered as irrelevant information for the player. However, in the earlier P2P-based approaches to MMOG, interest management was not exploited to restrict unnecessary message flooding. For examples, in games such as MiMaze [8] and Age of Empires [24] broadcast messages to all players in the game world. These irrelevant messages not only consume extra bandwidth, they also waste computing cycles of those nodes receiving the irrelevant messages. Therefore, this type of approach is not truly scalable. To address this problem, later approaches, such as AMaze [9] and Mercury [4], try to restrict message flooding by sending messages to the interest group through multicasting. However, IP multicast requires a special network infrastructure, which is not widely available.

In recent years, many proposals have addressed the scalability issue by constructing a local aware topology that allows players to send messages either directly or indirectly to those nodes within their area of interest. In general, these approaches can be categorized into two different types based on their P2P overlay architecture: a DHT-based approach, and an unstructured P2P approach.

2.2.1 DHT-Based Approach

In the DHT-based approach, designs rely heavily on the structured P2P overlay, such as Knutson et al's P2P Support for Massively Multiplayer Games [5] and Limura et al's Zoned Federation of Games Servers [29]. These designs divide the game map into fixed regions. An authoritative server is dynamically assigned to each region responsible for coordinating the game states. Such approaches aim to emulate the cluster of servers by dynamically promoting peer nodes as coordinators. While Knutson et al propose to use Scribe, an application-level multicast built on top of Pastry, for the game state dissemination, Limura et al argue that Scribe incurs unnecessary network delay due to the possible number of hops. They propose a direct communication model, in which coordinators cache the game states and maintain direct connections with regional members. Coordinators can then be used for storing members' states and updating them through direct communication between them and regional members. In Limura et al's Zoned Federation of Games Servers, coordinators are selected through a registration process via the DHT.

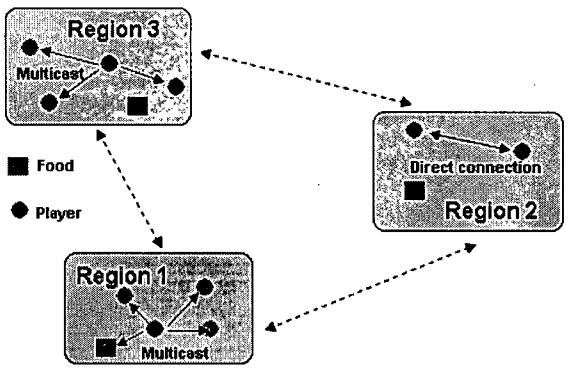


Figure 2-2 Map is partitioned into fixed regions

The common problem with this kind of approach is that they introduce artificial boundaries into the game world. Each player is placed into an artificially partitioned region as depicted in Figure 2-2. Players in different regions cannot view each other, even though they are within each other's AOI in the virtual world. In addition, the lack of the clear specification for partitioning the game world leaves an uncertainty as to the

game applications, as different ways of partitioning may significantly affect the games performance.

The significant shortcomings with the DHT-based approaches are that, the coordinators act as a server in each region, and there is no mechanism for players to build direct connections with other players within the same AOI. This can cause three problems: (1) game-state communications must be relayed through coordinators, so that extra network delays are introduced; (2) players may receive irrelevant information, since the regional server blindly sends updates to every player in the region; and (3) the regional servers can potentially be a bottleneck. For example, if the players within the same region generate lots of activities in a simultaneously, the region servers can be easily overloaded.

2.2.2 Unstructured P2P-Based Approach

The second kind of approach to resolving the scalability issue is based on an unstructured P2P overlay, in which all nodes must maintain connections even though they are dispersed in the virtual world. In this type of scheme, the challenge is to guarantee both global connectivity (the topology must be fully connected) and local awareness (of the nodes within the AOI) [18].

There are three major proposals in this category of approach. They are Kawahara et al's Neighbour List Exchange scheme [29], Keller, et al's Solipsis [18], and Hu et al's Voronoi scheme [25]. A common problem with these proposals is inefficient use of resources. In Kawahara et al's approach, each player continuously exchanges a neighbour list with its nearest neighbours. The messages generated for neighbourhood maintenance grow at $O(n^2)$, where n is the average number of neighbours of each player [13]. In Solipsis, each player acts as a "watchman" of the others. They watch every movement of their topology neighbours, and notify others when new topology neighbours are discovered for them. Note that the topology neighbours may not be close in the game

world (i.e., the topology neighbours can be far apart in the game world). In the Voronoi scheme, a Voronoi diagram is employed to regulate the network topology in order to maintain the globe connectivity. Every player needs to look ahead through their nearest topology neighbours in order to dynamically maintain topology integrity for every single movement. One of the major problems of the above proposals is that they tend to involve every node in the interest-management activity. This not only generates additional network communications, but also adds extra workload to each node.

The most significant problem with these schemes is that none of them is truly successful in maintaining consistent topology, or in discovering neighbours. In the Neighbour List Exchange scheme, players exchange neighbour lists only with their nearest neighbours. As a result, a player or a group of players can be isolated if they are far away from other players in the game world. Solipsis cannot guarantee neighbour discovery either, since directly connected neighbours are not reliable in notifying incoming neighbours of all scenarios, as described in [25]. The Voronoi scheme can hardly maintain consistent topology, if two or more adjacent boundary neighbours leave the game at the same time. As depicted in Figure 2-3, the Voronoi topology can hardly be recovered, if the nodes around the round node leaving the game world simultaneously, since based on the Voronoi scheme, a node relies on the connected boundary nodes to dynamically maintain the Voronoi-like topology.

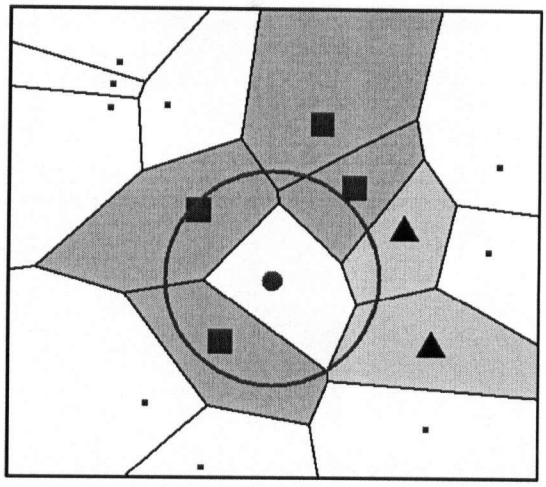


Figure 2-3 Fails to recover when adjacent nodes leave simultaneously

2.2.3 Hexagonal Zoning

The advantages of hexagons in dividing a game map have been recognized. Hexagons have uniform orientation and uniform adjacency. In addition, an AOI is usually defined by radius, and hexagonal zoning is more approximated to circles than is quadrant zoning. As depicted in Figure 2-4, a player moving from one cell to another joins and leaves the same number of cells. In Macedonia et al's proposal [22], a player's AOI consists of a radius of grid cells, where a player joins new cells at the leading edge and leaves old cells at the trailing edge as it moves forward. Each cell is mapped to a multicast group.

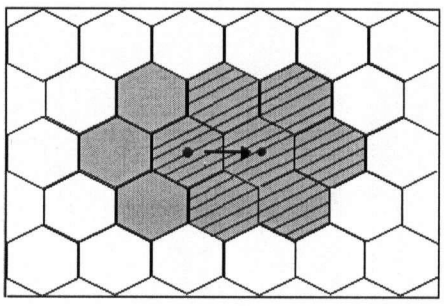


Figure 2-4 Moving between cells joins and leaves the same number of cells

However, in traditional hexagonal zoning, the view of each player in the virtual world is discrete, and will not change until the player moves from one cell to another. Such discrete view seems unrealistic in MMOGs, especially for these applications in which the players' visibility tends to be fairly sensitive. Another problem is that the traditional hexagonal zoning relies on a special network support such as IP multicast to forward a message from one cell to the other. Unfortunately, this special IP multicast infrastructure is not widely available. Moreover, when a game player is moving from one cell to another, the node needs to join and leave a set of cells. This joining and leaving can be costly in terms of bandwidth efficiency and responsiveness.

Chapter 3

System Architecture

This chapter presents the high-level details of our MOPAR system. We describe how we divide the game map into hexagonal zones, and how DHT is used to build the hierarchical structure. We also discuss how to deal with the dynamics of MMOGs, and the fault tolerance of our infrastructure.

3.1 Hexagonal Zoning

Each node maintains two configurable data: the dimensions of the game world and the AOI radius. The AOI radius defines the visibility of a player. In order to simplify the scenario, we assume that each player has the same AOI radius. Based on these two configurable data, a game application is able to divide the game map into hexagonal cells, and determine which cell a player is currently located in based on the player's position (x, y coordinates).

3.1.1 AOI Radius and Hexagonal Cells

Based on the mathematical structure of a hexagon [13] described in Figure 3-1, we define the variables of a hexagon and their mathematical relationships with the AOI radius in Table 3-1.

Symbol	Definition	Mathematical relationship
s	Side length	s = aoi radius
h	Height	h = sin(30°) * s
r	Distance	r = cos(30°) * s
b	Height of the surrounding rectangle	b = s + 2 * h
a	Width of the surrounding rectangle	a = 2 * r

Table 3-1 Mathematical relationships between AOI radius and a hexagon

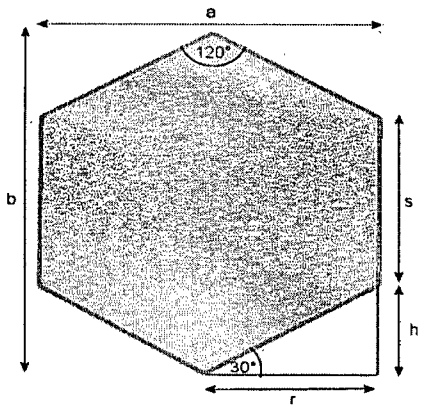


Figure 3-1 Mathematical structure of hexagons

Based on the above definition of a hexagonal cell, we can see that the size of our hexagonal cell is determined by the AOI radius of the game players. Figure 3-2 gives a graphical view of the relationship between AOI and a hexagonal cell, in which the circle represents the AOI of a player.

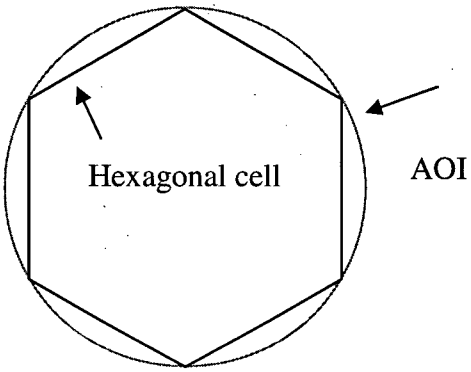


Figure 3-2 Graphical relationship between AOI and hexagonal cell

3.1.2 Mapping Positions to Hexagonal IDs

Once the game world is divided into the hexagonal cells, the cell's coordinates of a player located in needs to be determined based on the player's position coordinates on the map. We assume that information about the hexagonal cells is stored in a two-dimensional array (or cell coordinates), and we use the cell coordinates as the IDs of hexagonal cells. Basically, the problem now becomes how to map the position coordinates to the hexagonal cell IDs. [13] provides an algorithm for converting the pixels on the game map to the corresponding cell coordinates. We apply this algorithm to map a player's position in the game world to a hexagonal cell's coordinates (cell_id). As Figure 3-3 shows, the dot represents a player that is located in the cell with the cell_id [1, 1].

Unlike other schemes such as Macedonia et al's proposal [22], each player in our scheme is guaranteed to have a continuous viewing area, instead of discrete. The reason for this will be discussed in the next section, since it is also related to the design of our hierarchical architecture.

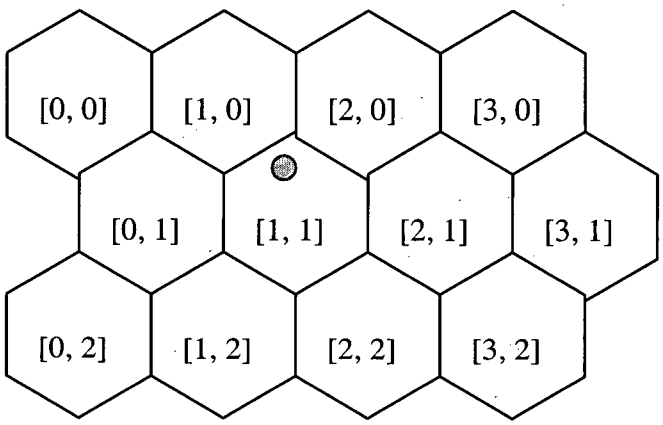


Figure 3-3 Dividing the game map into hexagonal cells

3.2 Hierarchical Structure Design

3.2.1 Motivation

As discussed in the previous chapter, having each node involved in interest management is not efficient, and cannot guarantee topology consistency. We observed that the source of the problems is in the nature of the topology model used – an unstructured flat model. For example, content searching in flat P2P model usually involves the message flooding. Some proposals [30] suggest that the hierarchical model can restrict message flooding effectively. Inspired by this idea, we want to design a hierarchical architecture that can be applied to interest management. The goal is that, in this hierarchical architecture, only the super nodes will be responsible for interest management. The super nodes (termed master nodes in our scheme) act as a "watchman" for other nodes, and notify other nodes whenever there is a node entering their area of interest. By doing this, we hope that neighbour discovery (interest management) can be achieved more efficiently and reliably.

We first need a reliable mechanism to facilitate building the hierarchical structure in a distributed manner. Inspired by the zone server selection method in Limura et al's proposal, we agree that DHT may be used for master node selection.

However, having the hierarchical structure only is not enough for performing the tasks of interest management. We also need to solve the following two problems: (1) the master nodes as a "watchman" need to be able to look ahead to immediately detect any incoming nodes; (2) the infrastructure needs to guarantee global connectivity. In other words, there must not be any isolated nodes globally. As shown in Figure 3-4, all nodes are connected with some other nodes locally, but the group on the left is separated from the group on the right. This is a common problem in the unstructured flat model. The movement of nodes in one group towards the other group will not be detected.

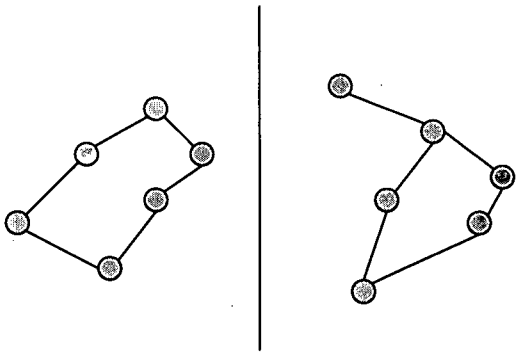


Figure 3-4 Left groups are separated from the right groups

Bearing these two problems in mind, we realize that, first, in order to make super nodes be the "watchman", the master nodes need to build connections with the master nodes in adjacent cells, if any exist (as shown in Figure 3-5). By doing this, a master node's covering area is enlarged to seven cells, which is large enough for covering the area of interest of a player at any particular position in the current cell. Second, if there are any isolated nodes or node, we need a mechanism to allow the nodes in one isolated group to connect to the nodes in another group when they are moving closer.

Fortunately, we find that the properties of the DHT, combined with our hexagonal zoning can solve these two problems nicely.

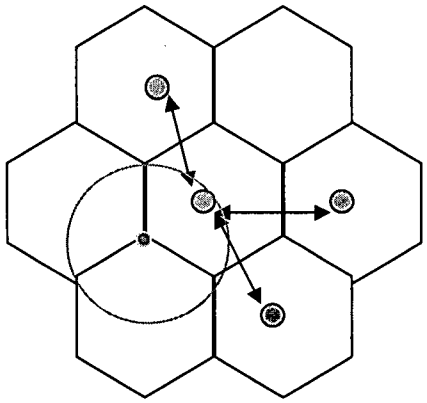


Figure 3-5 Connection between master nodes

3.2.2 Design Details

We choose Pastry as the particular variant of DHT. As with other variants of DHT, Pastry is a self-organizing, reliable, and decentralized overlay that provides the functionality of a distributed hash table. However, we choose Pastry because of two properties specific to it that are extremely useful for our application. The first is its leaf set property. The nodes in the Pastry leaf set of a node are the numerically closest nodes to the current node and can be reached in one hop (directly). This property is very useful for state replication, and also for a newly joined node to be able to immediately locate and obtain the states belonging to it; we will elaborate in a later section. The second desirable property is proximity. When a node routes a message, it always selects the closest node in distance from the candidate nodes. This property of Pastry is very important in meeting the responsiveness requirement of MMOGs.

Before we discuss the design details, we identify three types of nodes in our scheme: master nodes, slave nodes, and home nodes. The master nodes are the super nodes in the hierarchical topology, and the slave nodes are the nodes in the lower level.

The home nodes are the virtual nodes in the DHT space. Table 3-2 gives a description of the roles of each type of node.

Type of node	Description
Home node	A virtual node that acts as a home for a cell for master node registration or game states storage. A cell is mapped to a deterministic home node.
Master node	A node that acts as a coordinator of a cell. Each cell has at most one master node. Master nodes also have connection with the master nodes in neighbouring cells.
Slave node	Nodes other than the master node in a cell.

Table 3-2 Roles of nodes in the hierarchical structure

Home Node Initialization

We first describe how we map a cell to its home node, which is used as the rendezvous point for master node registration and potential game states storage. Based on our hexagonal zoning design, every cell has a unique cell_id (cell coordinates). We then map the cell_id to the identifier in Pastry space (we call it hex_id) by applying the SHA-1 hashing algorithm on the cell_id. The hex_id can now be used to locate the home node of the cell. The node that is the home node for the cell must be numerically closest to the hex_id. The binding between cell and the home node is determined. That is, a cell has only one deterministic home node; however, a node may be the home nodes for multiple cells (as shown in Figure 3-6).

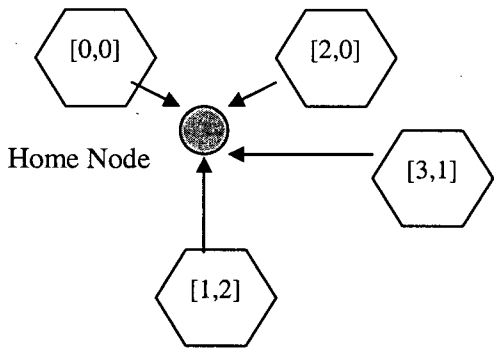


Figure 3-6 Relationship between home nodes and cells

Note that home nodes are virtual nodes, and that they are not necessarily positioned in the same cell with master or slave nodes. Because of random mapping via SHA-1, it is unlikely that a master or a slave node of a cell is also the home mode for the cell. The home nodes are initiated on demand; they will be initiated only when there are master nodes in these cells.

Master Node Registration

We assume that there is a node entering an empty cell (the cell does not contain any players). This node will become the master node of the cell. The following procedures will be executed:

1. A cell_id is calculated based on the position coordinates of the node on the map. For example, a node at position (30, 50) can be mapped to the cell_id (0, 0), if the side length of the hexagon is equal to 60.
2. Hashing the cell_id to the 128-bit hex_id by applying the SHAI-1 hash function.
3. Sending a query message containing the hex_id to locate the home node. The message will reach the destination node, which is now identified as the home node of the cell (the node's ID is numerically closest to hex_id). The home node starts the service.
4. The home node then notifies the requesting node that it is assigned as the master node for this cell.

Nodes entering the cell after the master node will be notified by the home node to become the slave nodes. Unlike the home node, master and slave nodes are physically located in the cells that they belong to. Switching cells will cause the structure to be re-constructed.

Building Inter-master Connections

As we discussed earlier, a master node needs to look ahead through connections with the masters in adjacent cells. This can also be achieved through searching the neighbouring master nodes. Whenever a master node starts the service, it will search the neighbouring masters in adjacent cells. The procedure is described as follows:

1. Neighbouring cell_ids are identified based on the master node's own cell_id. As shown in Figure 3-3, the neighbouring cell_ids of the cell [1,1], are cells [1.0], [2,0], [2,1], [2,2], [1,2] and [0,1]. These cells are then mapped to corresponding hex_ids.
2. Query messages are sent to each home node of these cells to retrieve the master node in these cells.
3. Connections with neighbouring master nodes are built if these nodes exist.

Global Connectivity Maintenance

As we mentioned before, interest management requires global connectivity to guarantee topology consistency, which is very costly in the unstructured flat model. Fortunately, this is not an issue in our architecture. In our architecture, master nodes will never lose connections with each other, if they are close enough. For example, in Figure 3-7, when node N moves from cell [3,1] into cell [2,1], it will attempt to build connections with all the neighbouring master nodes (as described in the previous section) including node M. Thus, global connectivity will never be broken.

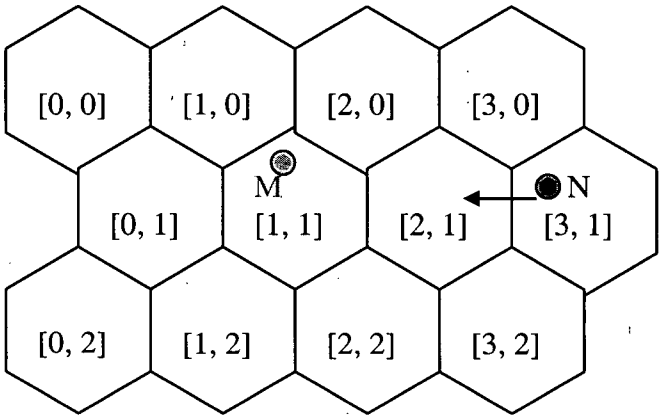


Figure 3-7 Global connectivity guaranteed

3.3 Handling the Dynamic Characteristics

Unlike typical applications for DHTs such as file sharing, MMOGs are much more dynamic in terms of frequency of nodes joining and leaving the environment. When a node first joins the environment, it can become the home node for some cell or cells immediately. This situation can happen when the newly joined node is numerically closer to these cells than their original home nodes. In this case, the newly joined node will become the home node for these cells, and therefore the master node registration records on the original home nodes must be transferred to the newly joined node immediately, in order to maintain the integrity of the system.

This home node transferring process could be very costly in some DHTs, as locating the original home nodes can be challenging in a distributed system. Fortunately, Pastry's leaf set property solves this issue efficiently. As mentioned before, in Pastry, every node maintains a set of nodes that are closest on the Pastry ring, and this set is named a leaf set. To describe the process more clearly, we use a concrete example. We assume that a node N0 is joining the game, and that it is inserted into Pastry ring between nodes N1 and N2, as shown in Figure 3-8. Before N0 is inserted, cell C1's home node is N1, because $|N1-C1|<|N2-C1|$. After N0 is inserted, N0 becomes the home node of C1,

as |N0-C1|<|N1-C1|. Therefore, C1's master node registration record will be transferred from N1 to N0. C2 will not be affected, since it is even further away than N2. As we can see from Figure 3-8, the insertion of the N0 will only affect N1 and N2, and those cells with IDs between N1 and N2. As N1 and N2 are in the leaf set of N0, N0 only needs one hop to reach them. Basically, whenever a node joins the game, it only needs to query the nodes on its right and left on the ring for transferring all qualified home nodes.

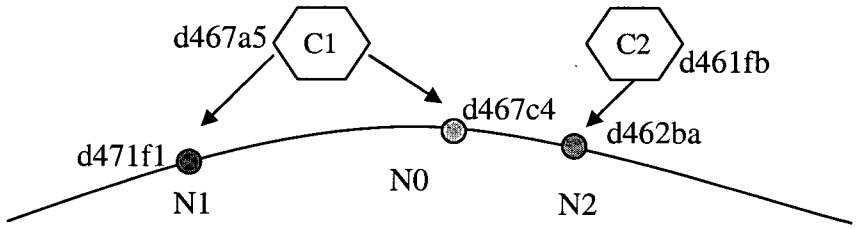


Figure 3-8 Home node transfer process

3.4 Direct Communication scheme

While DHT has many advantages, such as being reliable, and self-organizing, it also has some shortcomings. For example, messages take multiple hops to reach the destination node. Therefore, limiting the use of DHT is one of our design goals, since responsiveness is one of the important requirements of MMOGs. Based on this design goal, we build a direct connection between master and slaves, so that communication between them is direct, instead of being relayed by intermediate nodes, as in other DHT-based schemes such as Knutsson et al's proposal. Communications for interest management can be categorized into two types: inter-master communications and master-slave communications.

3.4.1 Inter-master Communications

Inter-master communications are the backbone communications for updating the current positions of the players among adjacent cells. Each master needs to collect position information from the neighbouring cells in order to cover the area across the boundary for its slave nodes. Messages exchanging position updates are sent periodically, and are also used as heartbeat messages for ensuring the health of the neighbouring masters. In Figure 3-9, the larger dots represent the master nodes, and the smaller dots represent the slave nodes. The lines connecting the master nodes represent inter-master communications. The circle represents the AOI of the node in the center of the circle.

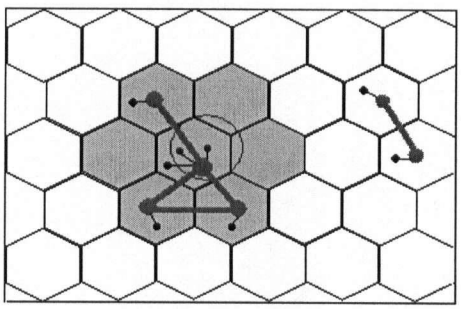


Figure 3-9 Direct inter-master and master-slave communications

3.4.2 Master-slave Communications

Master-slave communications are communications among masters and slaves in the same cells. Master-to-slave communications are for notifying the slaves of the new incoming neighbours; messages are sent only when there are new neighbours entering a slave node's AOI. Slave-to-master communications are for the slaves to update the masters of their current positions; updating messages are sent periodically. Slave-to-master messages are also used as heartbeat messages for ensuring the health of the slaves.

In Figure 3-9, the lines connecting master and slave nodes represent communication between masters and slaves.

3.5 Fault Tolerance

A node may leave the game without warning for many reasons, for example, due to network failure. This ungracefully leaving node can be a master node, slave node, or a home node. When the failure node is a home node, the master registration for this cell can be lost. Fortunately, Pastry nodes replicate the objects on the k nearest nodes in the leaf set, where k is the number of replication. The replication factor k can be increased to cope with high-failure network; however, the invariant for the number of replicas should always be kept. Therefore, a home node is dead, the master node registration record will not be lost, and the node that is numerically closest to the dead Pastry node will be chosen as the new home node automatically and transparently. This replacement accomplished in the following manner. When a node receives a master registration request from a node in a particular cell, it always checks its storage first. If there is a master registration record found for this cell, it knows that the original home node for this cell is dead, and so it replaces that home node.

The second case is when the failure node is a slave node. A slave node is supposed to periodically update the master node with its position. The updating message is also treated as a heartbeat message by the master node. Master nodes will detect the failure of a particular slave node from the absence of updating messages, and therefore, remove the slave node from the list.

The third case is when the failure node is a master node. The master nodes send the messages to the slave nodes if there are new incoming neighbours detected. However, if the time lapse with no incoming neighbours for a particular slave node exceeds a certain threshold, for example, two seconds, the master node will send a

keepAlive message to the slave node. This keepAlive message intends to notify the slave node that the master is still healthy. If a slave node does not receive either the keepAlive message or the neighbour notification message after two seconds, the slave node assumes that the master node has died. If this happen, the slave node will send an electing message to the home node, and then the home node will first check the health of the master node to decide on whether the master node needs to be re-elected or not (the first requesting node will be elected as a master node).

Chapter 4

Protocol and Algorithms

This chapter describes how the system dynamically adjusts and maintains the hierarchical topology to adapt to the continuous movement of players in the game world. It also provides the system analysis and comparisons with other schemes.

4.1 Procedures and Protocol

In this section, we describe the three major procedures that must be used by the peer at the application level to behave as a MOPAR application, and maintain the topological consistency of the whole system.

When a player's avatar moves around the map, its position will change continuously. Different positions of a player's avatar on the map will potentially affect the hierarchical topology in the infrastructure layer. For example, when a master node moves from one cell to the other, a new master node needs to be assigned to replace the leaving master node, if there are more nodes left in the cell.

We describe the joining, moving, and leaving procedures in the following. Table 4-1 and Table 4-2 describe the variables and the basic functions used respectively for the pseudo codes that describe the algorithms.

Variables	Description
cell_id	The ID of the hexagonal cell
pos	The node's x and y coordinates
master	The node handle of a master node
slaves	A list of slaves that a master node maintains
neighbourMasters	A list of neighbouring masters that a master node maintains

Table 4-1 Variable descriptions

Basic functions	Description	Return
getCellId(pos)	Get the cell_id based on the position of the node.	cell_id
isMaster()	If the node is a master node.	true/false
assignMaster(slaves)	Assign a new master selected from the given slave list.	none
stopMaster()	A master node stops its role.	none
stopSlave()	A slave node stops its role.	none
startSlave(master)	A slave node starts its role by registering to the master node of a cell.	none
getNeiMaster(cell_id)	Get the neighbour master node in the neighbouring cell with the given cell_id	neighbour master
searchNeighbourMasters(master, cell_id)	Search for the neighbouring masters of the cell with the given cell_id. "master" is the master node of the cell that the node is moving from.	neighbourMasters
startMaster(neighbourMasters)	Start the master role with the given neighbouring masters	none
SwitchCell(cell_id)	Switch to the cell with the given cell_id	none

Table 4-2 Basic function descriptions

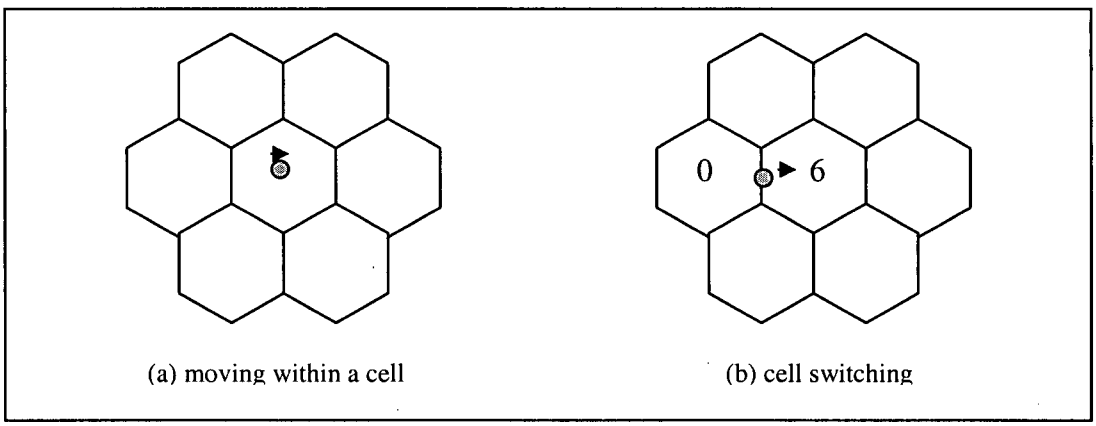
4.1.1 Joining Procedure

The joining procedure is executed whenever a node joins the game world, as follows:

1. The joining node will first be bootstrapped to be inserted into the Pastry overlay. The bootstrapping node's IP can be obtained from a lightweight server.
2. The joining node sends a joining request to the home node of the cell it is currently located in.
3. The joining node will either become the master node or the slave node, depending on the response from the home node.

4.1.2 Moving Procedure

Figure 4-1 describes the two scenarios involved in moving, and Figure 4-2 presents the pseudo code for the moving procedure. When a player moves within the same cell, the hierarchical topology will not affected, while when a node moves out of the current cell, the switching cell procedure needs to be executed.



(a) moving within a cell (b) cell switching

Figure 4-1 Moving scenarios

```
1: move(new_pos){
2:      pos = new_pos;
3:      //get the new cell_id based on the new_pos
4:      new_cell_id = getCellId(new_pos);
5:      if (cell_id == new_cell_id){
6:        //the node is moving with the same cell
7:      }
8:      else{
9:         //the node is moving out of the current cell
10:        switchCell(cell_id);
11:     }
12: }
```

Figure 4-2 Pseudo code for moving procedure

Cell Switching

If a player moves out of the current cell, a cell switching procedure will be executed to maintain the integrity of the hierarchical topology. Figure 4-1(b) illustrates the cell switching, in which a node move from cell 0 to cell 6. The cell switching procedure can be considered to consist of two steps. The first step is leaving the current cell, and the second step is entering the next cell.

There can be two scenarios in the first step. The first scenario is that the leaving node is the master of the cell. In this scenario, the node must hand over the master role to another node in the cell; however, if the master node is the only node in the cell, then the master node needs to stop its role by notifying all the neighbouring master nodes of its leaving. The second scenario is that the leaving node is a slave node. In this scenario, the slave node needs to un-register from the master node, so that the master node removes it from its slave list.

There are also two scenarios in the entering step. When the entering cell is empty, the node will become the master node of this cell. In this scenario, the node needs to search for the neighbouring masters of the entering cell first, and then start the master

role by notifying the neighbouring master nodes of its joining. In the second scenario, the entering node will be the slave node. In this case, the node needs to register to the master node of the entering cell, so that the master node can add the node into its slave list.

Figure 4-3 presents the pseudo code that describes the switching cell procedure.

```
1: switchCell(cell_id){ //the cell_id is the entering cell's cell_id
2:   //step1: leaving the current cell
3:  if(isMaster()){  //if the node is the master node of the current cell
4:     if(slaves.size()>0){ //if the master maintains more than one slaves
5:        assignMaster(slaves);
6                 stopMaster();
7:           }
8:           else{ //the master node is the only node in the cell
9:                //stop the master role
10:                stopMaster();
11:          }
12:  }
13:  else{ //if the node is a slave node in the current cell
14:    stopSlave();
15:  }
16:  //step 2: entering the next cell
17:  master = getNeiMaster(cell_id); //get the master node of the entering cell
18:  if(master!=null){ //there exists a master node in the entering cell
19:     //register the slave to the master node
20:    startSlave(master);
21:   }
22:   else{ //there does not exist a master node in the entering cell
23:      //search for the neighbouring masters for the new cell
24:      neighbourMasters = searchNeighborMasters(m0, cell_id);
25:      //start as a master in the new cell
26:      startMaster(neighborMasters);
27:  }
28: }
```

Figure 4-3 Pseudo code for switching cell procedure

4.1.3 Leaving Procedure

The leaving procedure is executed whenever a node leaves the game world gracefully. There are two scenarios: the leave master is either a master node or a slave node.

1. If the leaving node is a master node, it assigns a new master selected from its slave list. If the node is the master, but the only node in the cell, it first notifies all the neighbouring masters of its leaving, and then un-registers itself from the home node of the cell.
2. If the leaving node is a slave node, it simply sends an un-registration message to the master, so it will be removed from the slave list.

The leaving node might also be the home node for some cells, so it might hold some master registration records for these cells. Fortunately, this problem is resolved automatically and transparently. The master registration records are always replicated on the "k" nearest nodes, and the next nearest node will be promoted to be the home node with all the complete information. This process is almost identical to the ungraceful leaving scenario, which we have discussed in the previous chapter.

4.1.4 Protocols

We also developed a set of protocols for supporting the MOPAR communications. They are described in Table 4-3. There are two types of communications; one is communication between the node and the home node, the other is direct communication between the nodes.

Message	Parameters	Response	Purpose
DHT messages			
join	IP address	IP address (master node)	Join the game
leave	IP address	None	To remove the master registration
elect	IP address	IP address (newly elected master node)	To request the home node to elect a new master node, as the master node with the given IP address is detected death.
queryMaster	None	IP address (master node)	To get the master of a cell
registerMaster	IP address	None	To register a master node (this is used by an assigned master to force the home node to change the master registration record)
unregisterMaster	IP address	None	To remove the master registration record from the home node
Direct Communications			
queryNeighbourMaster	[cell_id]	[IP address]	To query the neighbouring masters to a master node by giving the list of cell IDs
assignMaster	neighbourMasters slaves	True/False	To assign a slave node to be the new master by giving the neighbouring masters and slaves
changeMaster	IP address	None	Notify the slaves of the changes of the master
changeNeighbourMaster	IP address	None	Notify all the neighbouring masters of the change of the master
unsubscrbeNeighbourMaster	IP address	None	Notify the neighbouring masters of its leaving
unsubscribeSlave	IP address	None	Un-register from a master
registerSlave	IP address	None	Register a slave to the master
updateSlavePosition	Position	None	Update the master with the current position
updateNeighbourMaster	[IP address, position]	None	Exchange the positions of the nodes in the current cell with the neighbouring masters
newNeighbours	[IP address]	None	Notify the slaves of the new entering neighbours
keepAlive	[IP address]	None	Notify the slaves of the healthy of the master

Table 4-3 Protocols for MOPAR

4.2 Neighbouring Masters Searching Algorithm

Every time a node moves into an empty cell, the node will become the master node of this cell. The master node will need to search for the masters in the neighbouring cells. One simple algorithm is to query the home nodes of the neighbouring cells. However, home node queries are DHT queries, which are fairly costly, as the query message may take multiple hops to reach the home nodes. Therefore, minimizing the DHT query will be useful for reducing bandwidth consumption and communications delay.

4.2.1 Blind Searching Algorithm

The simple algorithm is straightforward. The entering master node sends a multi-hops query message (DHT message) to all the home nodes of the neighbouring cells; we call it blind searching. The main disadvantage of blind searching is that messages are sent to all the home nodes of the neighbouring cells without considering that some of the cells might be empty, so the searching can generate many unnecessary messages. As shown in Figure 4-4, the master node sends DHT query messages to the home nodes of all the neighbouring cells, even if these cells are all empty. At maximum, there can be as many as six unnecessary DHT queries (every cell has six neighbouring cells). In addition, DHT messages as multi-hops messages suffer from potential latency and bandwidth inefficiency. However, blind searching is unavoidable when a node first joins the game world as a master of a cell.

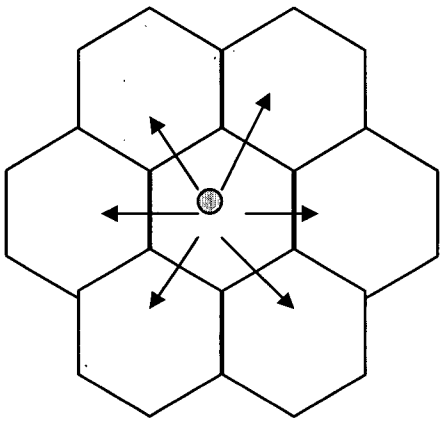


Figure 4-4 Blind searching

4.2.2 Intelligent Masters Searching Algorithm

The intelligent masters searching (IMS) algorithm makes the best use of knowledge of the known neighbouring master nodes to minimize the number of multi-hop DHT queries and detect possible empty cells.

As shown in Figure 4-5, we assume that node N6 is moving from cell 0 to cell 6 and will become the master node in cell 6. We also assume that M0, M1, M2, M3, M4, M5, M6 represent the maser nodes in the cells C0, C1, C2, C3, C4, C5, C6, respectively. The algorithm is based on one invariant that, every master node is aware of the master node in each of its neighbouring cells, unless a particular cell is empty. The intelligent search algorithm is designed based on the following observations.

1. M6 does not need to send a DHT query to obtain M0, since C0 is its origin. M6 knows the master node of C0, either because M6 was a slave node in C0 or itself was a master of C0 before leaving.
2. As M1 and M5 are the neighbouring masters of M0, M6 can obtain the information about them directly from M0, instead of via DHT searching.

3. If M1 does not exist, then M6 can safely conclude that C1 is empty, so DHT searching for M1 can be pruned. The same idea applies to M5.
4. If M1 exists, then M6 can obtain M2 from M1 via a direct query; otherwise, a DHT query for M2 needs to be sent. The same idea applies to M4.
5. By the same token, M6 can obtain the complete information about its neighbouring masters in C0, C1, C2, C3, C4, and C5.

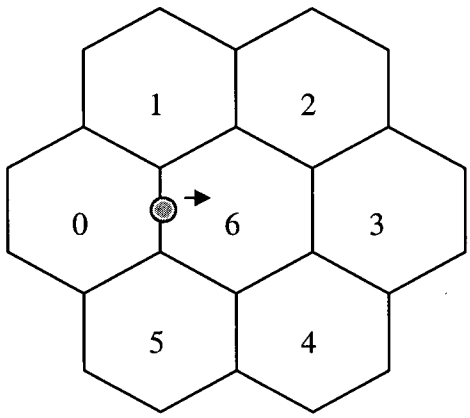


Figure 4-5 Cell numbering for describing searching algorithm

Before we describe the intelligent masters searching algorithm, we define some notations in Table 4-4.

Symbol	Description
m_i	The masters of the neighbouring cells; "i" represents the number of the cell.
$cell_id_i$	The IDs of the neighbouring cells; "i" represents the number of the cell.
queryNeiMasters(cell_ids)	Query the neighbouring masters from a master node for the given list of cell_ids.
DHTQuery(cell_id)	Query for the master node to the home node of a cell with the given cell_id via the DHT.

Table 4-4 Notations in intelligent masters searching algorithm

```
1: searchNeighborMasters (m0, cell_id0){ //m0: master of origin cell
2:    if (m0 equals m6){ //local query
4:        [m1, m5] = getNeiMasters([cell_id1, cell_id5]);
5:    }
6:    else{ //direct query
7:      [m1, m5] = queryNeiMasters([cell_id1, cell_id5]);
8:    }
9:    if(m1!=null){
10:       //case 1: m0 knows m1.  m6 queries for m2 from m1
11:       [m2] = m1.queryNeiMasters([cell_id2])
12:   }
13:   else{
14:      //case 2: m1 does not exist, m6 queries for m2 via DHT
15:       m2 = DHTQuery(cell_id2);
16:   }
17:   if(m2!=null){
18:      //case 3:  m2 exits.  m6 queries for m3 via m2
19:      [m3] = m2.queryNeiMasters([cell_id3]);
20:   }
21:   if(m5!=null){
22:      //case 4: m5 exists.  m6 queries for m4 via m5
23:      [m4] = m5.queryNeiMasters([cell_id4]);
24:   }
25:   else{
26:      //case 5: m5 does not exist.  m6 queries for m4 via DHT
27:      m4 = DHTQuery(cell_id4);
28:   }
29:   if (m2==null && m4==null){
30:     //case 6: neither m2 nor m4 exists.  m6 queries for m3 via DHT
31:     m3 = DHTQuery(cell_id3);
32:   }
33:   else if(m4!=null &&m2==null){
34:      //case 7:  m4 exists, but m2 does not. m6 queries for m3 via m4
35:      [m3] = m4.queryNeiMasters(cell_id3);
36:   }
37: }
```

Figure 4-6 Intelligent master searching algorithm

There are a total of eight cases, including the initial case, involved in probing the complete set of neighbouring masters:

1. *Initial case*: If M0 and M6 is the same node, a local query is processed to obtain M1 and M5; otherwise, a direct query message is sent to M0 to obtain M1 and M5.
2. *Case One:* If M1 exists, M6 sends a direct query message to M1 to obtain M2.
3. *Case Two:* If M1 does not exist, M6 sends a DHT query message to the home node of C2 to obtain M2.
4. *Case Three:* If M2 exists, M6 sends a direct query message to M2 to obtain M3.
5. *Case Four:* If M5 exists, M6 sends a direct query message to M5 to obtain M4.
6. *Case Five:* If M5 does not exist, M6 sends a DHT message to the home node of C4 to obtain M4.
7. *Case Six:* If neither M2 nor M4 exists, M6 sends a DHT query message to the home node of C3 to obtain M3.
8. *Case Seven:* If M4 exists, but M2 does not exist, M6 send a query message to M4 to obtain M3.

Figure 4-6 presents the intelligent neighbouring masters searching algorithm. Compared to the blind searching algorithm, the costly DHT query is minimized, and unnecessary searching is pruned. In the worst case, there are three DHT queries at most happening at lines 15, 27, and 31. The best case is that all the neighbouring masters can be retrieved the direct queries from the existing masters, for example, each neighbouring cell has a master node. This best-case scenario is extremely beneficial to system

scalability, since this scenario often happens when the population of the game world is large.

4.3 Overhead Costs

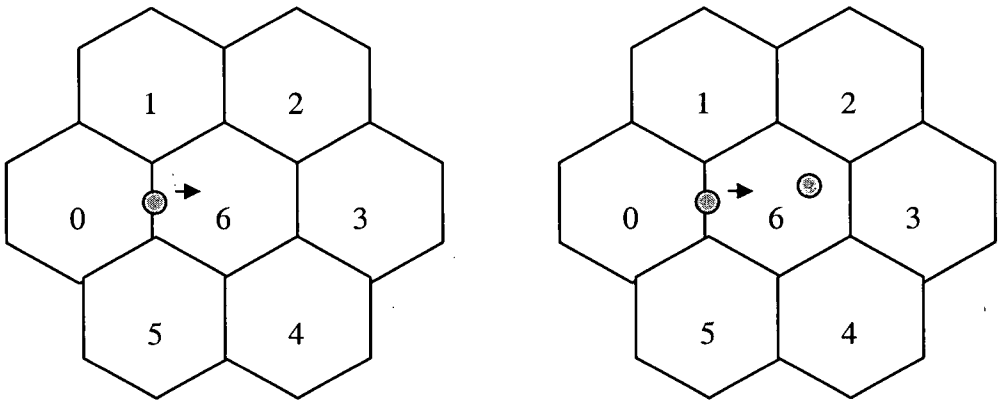
In this section, we will analyze the overhead of our dynamic interest-management scheme. First, we want to define what we mean by "overhead" in our context. We regard the inter-master and master-slave communications for neighbour discovery as the regular cost of our scheme. The other costs, aside from the regular neighbour discovery costs, are considered to be overhead. Basically, the overhead costs are those incurred whenever a player switches from one cell to the other. In another words, overhead costs are the processing costs of maintaining the hierarchical master-slave structure dynamically. We divide overhead costs into two kinds: costs incurred when leaving a cell, and those incurred when entering a cell.

Leaving Costs

In the leaving process, there are two cost scenarios depending on the role of the leaving node. If the node is a slave node, then the cost incurred is un-registration from the master node. If the node is a master node, then the costs incurred can be different, depending on whether the master node has slaves or not. If the master node has more than one slave nodes, the following leaving costs incurred: (1) a new master is assigned by transferring the information about slaves and neighbouring master nodes; (2) the newly assigned master notifies the neighbouring masters of the change; (3) the newly assigned master notifies the slave nodes of the change; (4) the newly assigned master registers itself to the home node of the cell. In the other case, if the leaving master is the only node in the cell, then the costs are: (1) un-registering itself from the home node; (2) un-registering itself from the neighbouring masters.

Entering Costs

In the entering process, there are also two scenarios, depending on the role of the entering node. One cost that is common to the two scenarios is that the entering node needs to query the master node (if it is not the master node itself there) of the origin cell to determine its role in the entering cell. If the entering cell does not have a master node, the entering node will be master node in that cell; otherwise, it will become a slave node (as shown in Figure 4-7). When the entering node is the master node, then the costs are: (1) searching the neighbouring master nodes of entering cells; (2) notifying the neighbouring masters of the newly joined master; and (3) registering itself to the home node. In the other scenario, the entering node will be the slave node. In this case, the overhead cost consists of registering itself to the master in the entering cell.



(a) Entering a s master

(b) Entering as a slave

Figure 4-7 Entering as a master or a slave

4.4 System Analysis

4.4.1 Advantages

Message Aggregation

In the flat-model approaches, every node is involved in maintaining the neighbourhood topology and discovering neighbours, either by periodically exchanging neighbourhood information with the current neighbours, or by looking ahead of the nearest neighbours. We combine these two ideas, and apply the result in a hierarchical fashion. In our scheme, only master nodes exchange neighbour lists with other master nodes in the neighbouring cells, and look ahead for their slave nodes for the incoming neighbours. Slave nodes are notified only if their neighbour sets have any changes, i.e., there are some new neighbours moving into the AOI. Moreover, a game player who first joins the game immediately gets the neighbour set through one aggregated message from the master node. The delay for the joining process is thus reduced, compared to the flat model, in which newly joining nodes need to probe for neighbourhood information by themselves.

Continuous Viewing Area

Compared to regular hexagonal zoning, in our scheme the view of each player in the virtual world is continuous instead of discrete. This is because the AOI of each player is smaller than the covering area of the master nodes. In the specific example shown in

Figure 4-8, a master node's viewing area covers 7 cells, while a slave node's viewing area is within the 7 cells. Therefore, a master node can provide continuous neighbourhood information for the slave nodes in its governing cell. A master node's viewing area is extendible, in case a slave node's AOI is larger than its covering area.

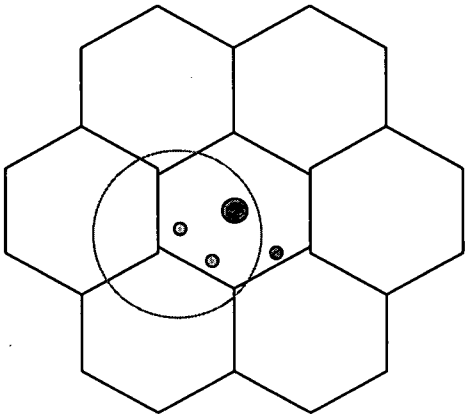


Figure 4-8 Players have continuous viewing area

A Hybrid of DHT and Unstructured P2P

Since a DHT has overhead for the $O(\log N)$ hops from the source node to the destination node, our design goal is to minimize the use of the DHT for interest-management communications. In our scheme, the DHT is used only for maintaining the hierarchical structure, or registering master nodes. If the master nodes rarely leave a cell, and every cell has a master node, the DHT will not be used. Slave nodes can obtain direct connections with the master nodes of the neighbouring cells through the current master node, when they want to move to neighbouring cells. Although it is an ideal case, players in MMOG tend to move slowly and tend to gather together. By minimizing the use of DHT and taking advantage of direct communication scheme, we provide a benefit of low latency to our scheme.

On the other hand, it's very challenging for pure unstructured P2P architecture to maintain global connectivity. Global connectivity makes sure that any node or group of nodes can never be isolated. However, few of existing systems successfully maintain global connectivity, as discussed earlier. In our scheme, global connectivity can never fail, unless there is a massive network failure, which causes the network to be partitioned; otherwise, a node can always connect to another node, through the assistance of home

nodes, if necessary. Therefore, only the nodes that are close-by need to be grouped together, which is very efficient and low-overhead in maintaining the global connectivity.

4.4.2 Comparison with other schemes

We compare our scheme with other schemes in the following aspects: (1) scalability – the costs incurred for maintaining the topology should be low; (2) reliability – the system should not become broken due to node failure; 3) persistency – games states need to be stored persistently; and (4) consistency – the discovery of neighbours should be consistent.

MOPAR tends to be more scalable in terms of low overhead costs for interest management. There are much fewer nodes involved in message exchanging, and only the master nodes exchange the neighbour lists, which is more efficient than the Neighbour List Exchange scheme. We also have fewer "watchmen" who need to look ahead for others. In both the Solipsis, and Voronoi schemes, each node always maintains extra connections to maintain global connectivity; they are even dispersed on the map. However, it's common for there to be clusters of players in MMOGs. Players who are doing different activities in different virtual locations do not need to be aware of each other. Our scheme allows them to be entirely separated without worrying about any node being isolated.

Our scheme is also more reliable than pure unstructured P2P schemes through the use of the DHT. A node failure can be resolved automatically through the support of DHT. Other systems may require a server to support them (e.g., the Neighbour List Exchange scheme and Solipsis), or may fail under certain circumstances (e.g., the Voronoi scheme).

It is hard for pure unstructured P2P schemes to address game states persistency and consistency issues. Although addressing these issues is not in the scope of this work,

they can be addressed easily through our hierarchical structure and the support of the storage persistence service from Pastry.

Moreover, the consistency of the neighbour discovery should be fairly high, given that it is the main purpose of interest management. Our scheme guarantees neighbour discovery in theory. However, other schemes may fail neighbour discovery under certain circumstances. In the Voronoi scheme, there is a speed limit for the players. Moving too fast can potentially cause the un-discovery of a node. In the Neighbour List Exchange scheme, nodes can be isolated. In Solipsis, an incoming node may be undiscovered by the directly connected neighbours.

Table 4-5 summarizes comparisons of the existing systems.

	Solipsis	**Neighbour List Exchange**	**Voronoi**	**MOPAR**
Scalability	Median overhead	High overhead	Median overhead	Low overhead
Reliability	Server support	Server support	Fully distributed (no guarantee)	Fully distributed (guaranteed)
Persistency	N/A	N/A	N/A	Available
Consistency (discovery)	No guarantee	No guarantee	No guarantee	Guaranteed

Table 4-5 Comparisons of existing systems

Chapter 5

System Implementation

In this chapter, we present the implementation details of the MOPAR prototype, and the game simulation and visualization built on top of it. In doing so, we demonstrate the effect of neighbour discovery and explore the roles of the different types of nodes, such as a master, slave and home node in the MOPAR infrastructure.

5.1 System Design

5.1.1 Architecture

MOPAR is implemented in a layered architecture consisting of four layers, as described in Table 5-1. Layers are separated from each other, and replaceable. The infrastructure implementation therefore has flexibility the implementation flexibility. For example, in this implementation, we use FreePastry [3] for implementation at the DHT layer, but it can potentially be replaced by another implementation of Pastry.

Layer	Description
Application	The application layer is where the specific game's implementation should be placed, including the game logic, UI. Game implementation can also access the network layer directly.
MOPAR infrastructure core	The MOPAR infrastructure core layer is responsible for the core implementation of MOPAR interest management and neighbour discovery, including implementation of the master, slave, and home nodes. This layer sits directly on top of the DHT layer, and can access the network layer as well.
DHT (Pastry)	This layer constructs a P2P network, and provides the distributed data placement and lookup service. This layer sits directly on top of the network layer.
Network	The network layer is the low-level network communication layer. It facilitates communications among peers via TCP or UDP.

Table 5-1 MOPAR layer descriptions

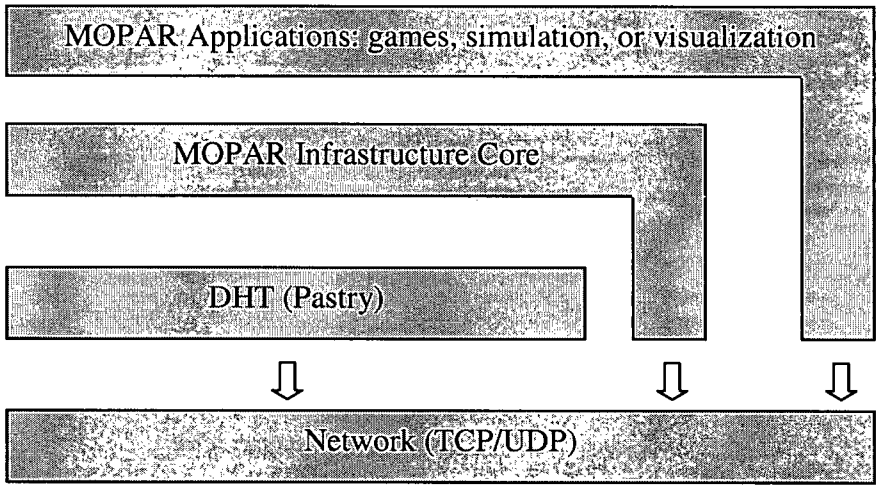


Figure 5-1 MOPAR layered architecture

Figure 5-1 displays the relationships of layers in the MOPAR infrastructure. MOPAR infrastructure defines a set of interfaces which a MOPAR infrastructure core must implement. The interface provides an abstraction between the game application layer and the MOPAR infrastructure's actual implementation. The game application layer uses this set of interfaces as the MOPAR API to integrate game applications into the MOPAR infrastructure. This set of interfaces is described in Table 5-2.

API	Description
join (double[] pos)	This method is called when a game player joins the game. The MOPAR infrastructure computes the hex_id based on the given position, and forwards the joining message to the home node of the associated hex cell.
leave()	This method is called when a player finishes a game and leaves the game world. The method allows for graceful leaving.
move(double[] pos)	This method is called whenever a player makes a movement in the game world. The given position is used to determine if the player is moving within the same cell, or is switching to another cell.
getNeighbours()	Gets the list of players that fall within the AOI

Table 5-2 API of MOPAR infrastructure

5.1.2 Major Components

Table 5-3 describes the major components of the MOPAR infrastructure core layer.

Package	Description
participant	This package includes classes that implement the core API of the MOPAR infrastructure such as join, leave, and move.
moparappl	The classes in this package provide implementation of Pastry's application interface, including the methods forward, deliver, and update. This is how MOPAR hooks up with Pastry.
homenode	This package provides the service for initializing the home nodes, including transferring home node from the nearest Pastry nodes, as well as the implementation of the home node.
master	This package includes classes for implementation of the behaviours of the master node, including inter-master communications, neighbour discovery, neighbour notification, and so on.
slave	This package includes the classes for the implementations of the slave node.
messaging	This package defines message types for interest management services, and provides the message-passing service.
util	This package maintains the utility of the infrastructure, including constants, variables, and the hexagonal division utilities.
properties	This package contains the configuration files for the MOPAR infrastructure. The configurable variables include AOI radius, game world dimension, etc.
game	This package implements a simple game simulation for demonstration purposes.
visualization	This package implements visualization of the game simulation, and the MOPAR infrastructure.

Table 5-3 Major components of the MOPAR infrastructure core

5.2 Implementation Details

5.2.1 Core Classes

The following are the core classes that provide the important functionalities of the MOPAR infrastructure. The names on the right of each class denote the interface or super-class of the implemented class.

master.MasterImpl — *master.IMaster*

Provides neighbour discovery and neighbour notification services.

slave.SlaveImpl — *slave.ISlave*

Implements the behaviour of a slave node. A slave node periodically updates its current position with the master node.

participant.MoparParticipant — *IMoparParticipant*

Implements the functionalities, including joining, moving, leaving, and the efficient neighboring master searching algorithm.

homenode.HomeNodeFactory

A class called to initialize a home node, and performing the action required to transfer the home nodes from the neighbouring Pastry nodes to the current node, if the current node becomes the more qualified home node(s).

homenode.HomeNode

Provides the service for master registration and responds to the query for master registration lookup.

moparappl.MoparAppl

Implements the Pastry's application interface, so that MOPAR becomes a Pastry application and be integrated into Pastry layer.

messaging.MessageSender

Provides the service for sending a Pastry message from current node to another Pastry node.

util.MapHelper

Provides the service for converting players' positions to cell IDs.

game.MoparGame *participant.MoparParticipant*

Implements a simple game logic for demonstration purposes.

game.MoparNeiworkSimulator

Simulates the TCP/UDP network communications between game players.

GUI.gamePanel *javax.swing.JPanel*

Draws the game world map, as well as players and their interactions.

GUI.gameViz *javax.swing.JFrame*

The GUI of the MOPAR visualization.

5.2.2 Collaboration Diagrams

This section describes the flow of object interactions in the MOPAR infrastructure through the assistance of UML sequence diagrams. We choose three major functionalities provided by MOPAR infrastructure: the joining, moving and leaving procedures.

Joining Procedure

Figure 5-2 illustrates class collaboration under the scenario of a game player initiating a new game. We assume that the player's avatar falls in an empty cell, so that the joining node will become a master node of the cell. In the diagram, we can see that the node will first be inserted into the Pastry overlay, and then the Pastry overlay will be used for registering the master node and searching the neighbouring masters.

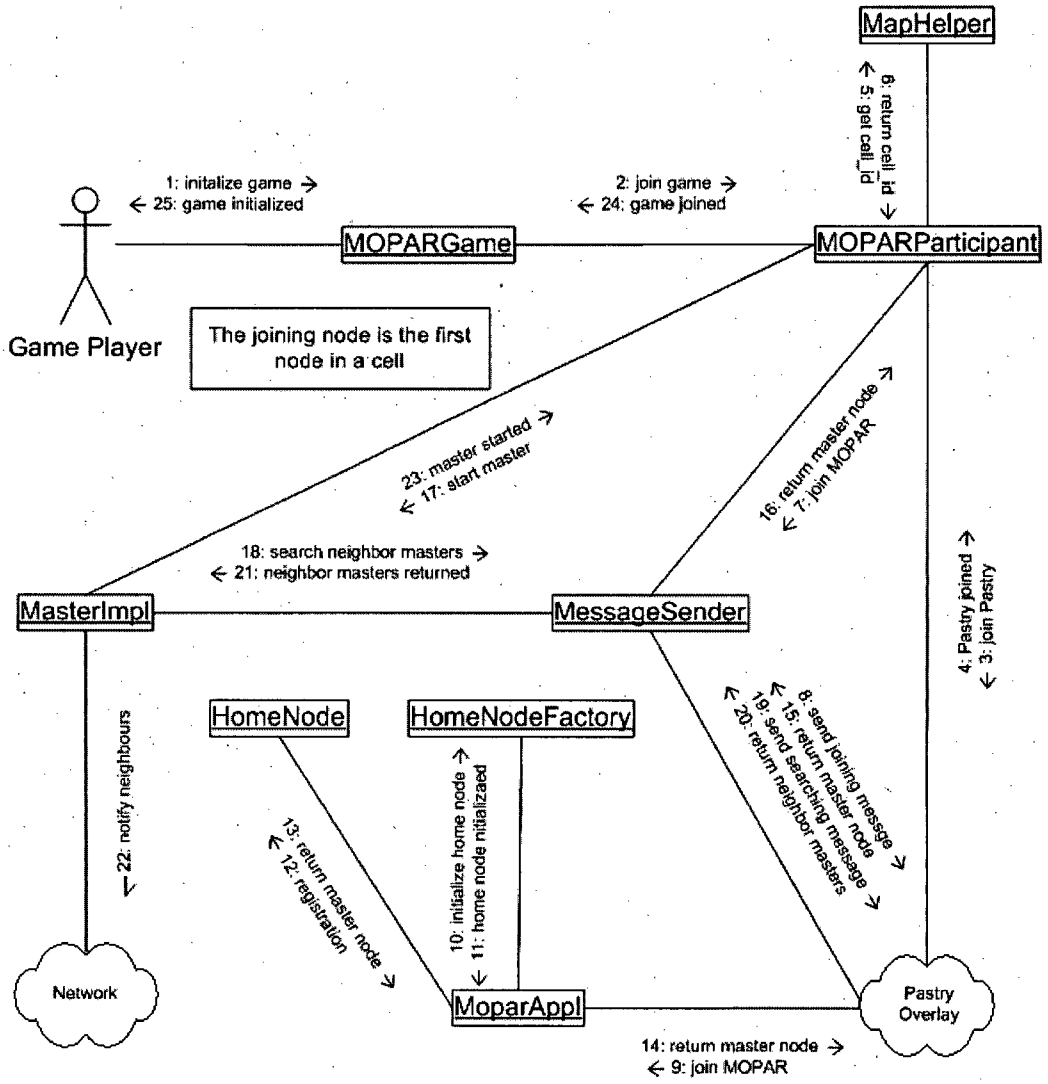


Figure 5-2 Collaboration diagram for joining procedure

Moving Procedure

Figure 5-3 describes the operations performed in the MOPAR infrastructure layer when a game player makes a movement to the player's avatar. We assume that the node is a master node in the current cell, and will become a slave node in the entering cell. As we can see from the diagram, the Pastry overlay is used only when the assigned master node registers itself, and all other communications are performed directly via the network layer.

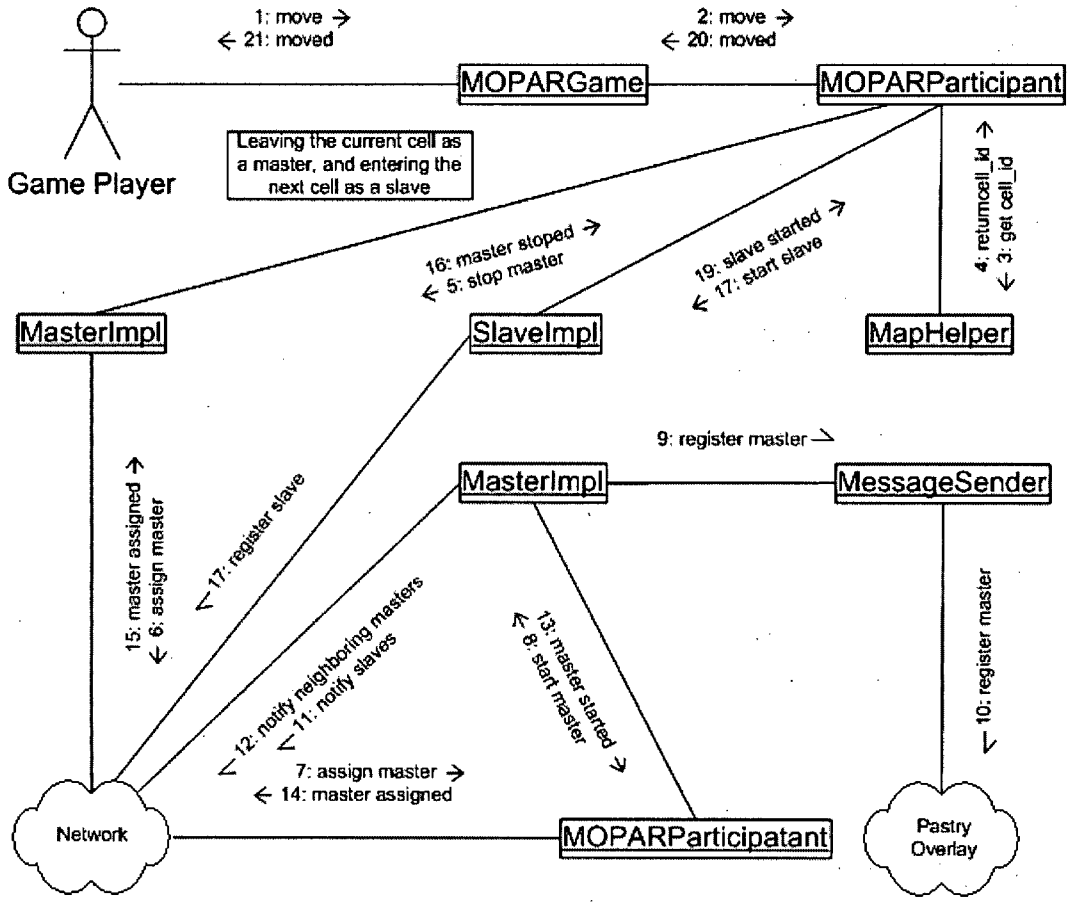


Figure 5-3 Collaboration diagram for moving procedure

Leaving Procedure

Figure 5-4 illustrates class collaboration under the scenario of a game player leaving the game. We assume that the node is a master node and the only node in a cell. Therefore, the master node simply un-registers itself from the home node via the Pastry overlay, and then notifies all the neighboring masters of its leaving.

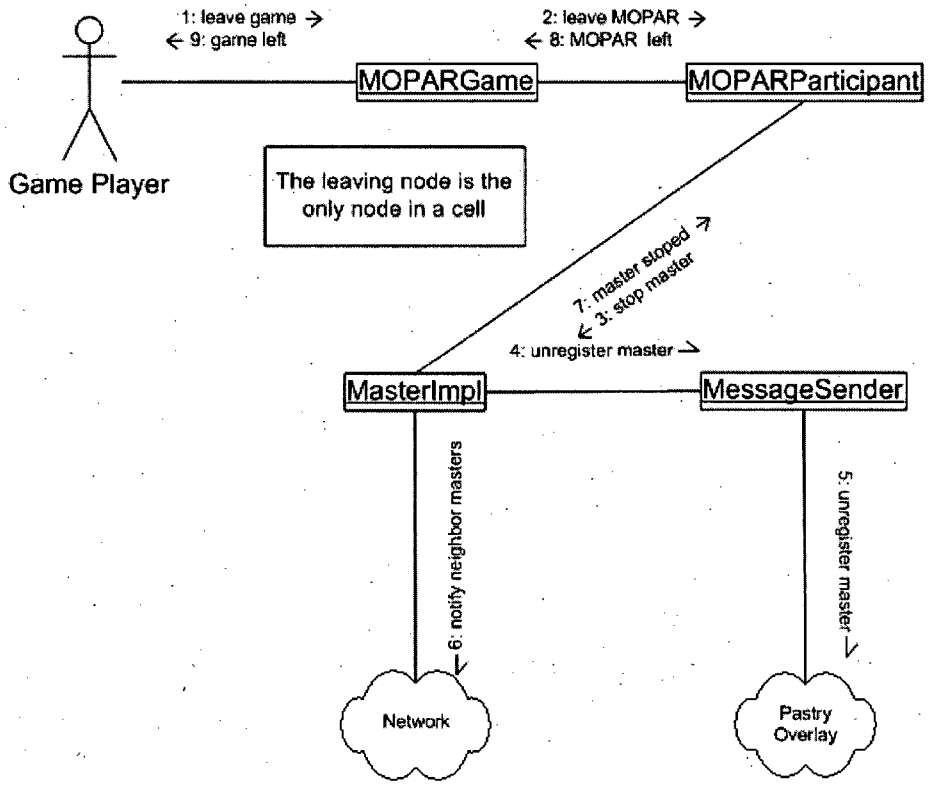


Figure 5-4 Collaboration diagram for leaving procedure

5.3 Visualization

In order to demonstrate the effect of neighbour discovery and explore the MOPAR hierarchical architecture, we have also implemented a simple game simulation and visualization tool. The game is simple; players move randomly on the map. There is one activated player at a time. The activated player sends message to others who fall in its AOI. The players who receive the message blink by continuously changing colour between black and red. In the visualization tool, the master nodes distinguish themselves by doubling their size relative to slave nodes. One can easily view how a node switches roles between being a slave and master node dynamically. The hexagonal zoning can also be shown as the background of the map. In addition, when the visualization tool is on home-node mode, clicking on a cell will trigger the node that is the home node for the

cell being popped up. In summary, the visualization tool aims to provide the following demonstration functions:

1. Demonstrate the dynamical construction of the hierarchical structure.
2. Show the effect of node moving, cell switching, and role switching between master and slave.
3. Demonstrate the relationship between home nodes and cells.
4. Show the relationship between the AOI radius and players, and the relationship between AOI radius and hexagonal cells.
5. Demonstrate the effect of the neighbour discovery.
6. View the hexagonal cell divisions.

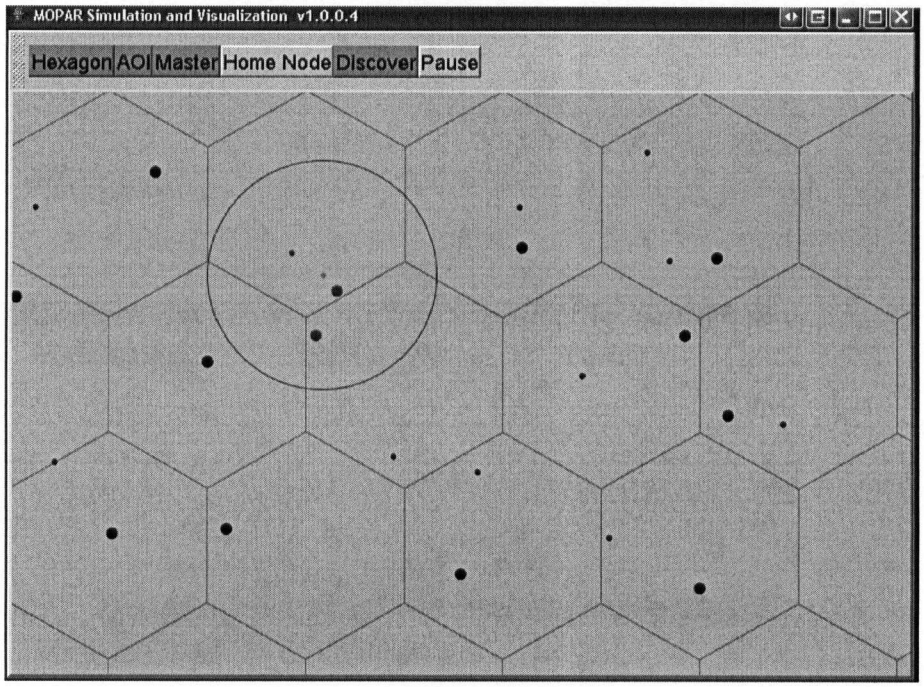


Figure 5-5 Hierarchical structure and neighbour discovery

In Figure 5-5, the larger dots represent the master nodes, and the smaller dots represent the slave nodes. The red circle indicates the AOI of the player in the middle of

the circle. The player in the middle of the circle keeps sending messages to the players that fall in its AOI, and makes these nodes blink. We can see that each cell has at most one master node, but can have multiple slave nodes.

In Figure 5-6, the largest blue dot represents the node that is the home node for the cell highlighted in blue. It can be seen that a home node is a virtual node, and is not necessarily located in the cell the node is home for.

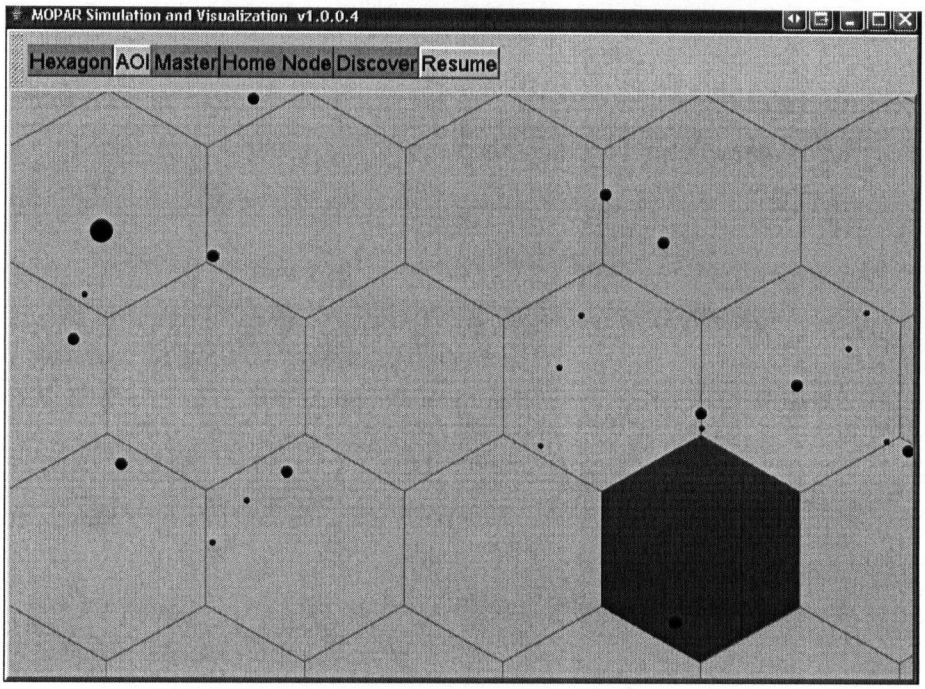


Figure 5-6 Home node of a hexagonal cell

Chapter 6

Evaluation

In this chapter, we evaluate five aspects of the performance of MOPAR: system scalability, efficiency of the intelligent neighbouring masters searching algorithm, impact of varying the AOI radius, DHT performance, and topology consistency. Note that it is not meaningful to discuss scalability without taking the following three parameters into account: the virtual world's dimension, the number of players, and the AOI radius. For example, in a virtual world with dimension 1000x1000, a system containing 300 players with an AOI radius of 100 units, is considered to have the same scalability as a system containing 14,154 players and with an AOI radius of 15 units. This is because, in both systems, each player's AOI is on average covers the same number of neighbours (around 10) within its AOI range. Since MOPAR, is a fully distributed system, it is not an issue to handle a large number of players, providing the cell size or AOI radius is set appropriately. In our experiments, if not specified otherwise, we set up the map dimension to be 1000x1000, the population to range from 25 to 325 players, and the AOI radius at 100 units. We also vary the AOI radius to expose the impact of varying the AOI radius on the systems performance. We run the simulation for 1200 simulated steps in each experiment. In each simulated step, a game player makes a movement, and also

performs the MOPAR actions according to its role as master or slave. In addition, we assume that our game model is a fast-paced first-shooter action game, where the game updates occur around 10 times per second. Therefore, the 1200 simulated steps are equivalent to 2 minutes of game play in the real world.

6.1 System Scalability

6.1.1 Master Workload

Since the master nodes play a major role in neighbour discovery, their workload directly determines the scalability of the system. The master nodes are mainly responsible for four types of messages involved in maintaining neighbour discovery for each player, as follows:

Sending: messages to the masters of the neighbouring cells

Sending: messages to the slaves in the current cell (neighbour notification)

Receiving: messages from the masters of the neighbouring cells

Receiving: message from the slave nodes (position updating)

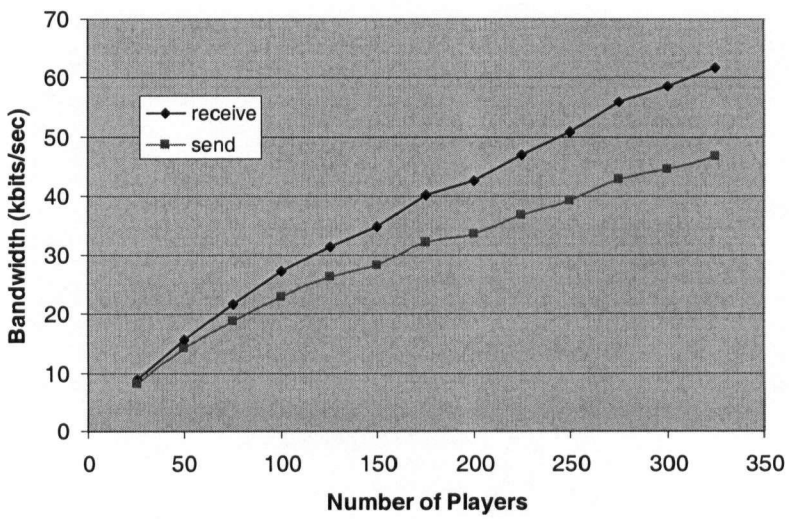


Figure 6-1 Master node bandwidth consumption

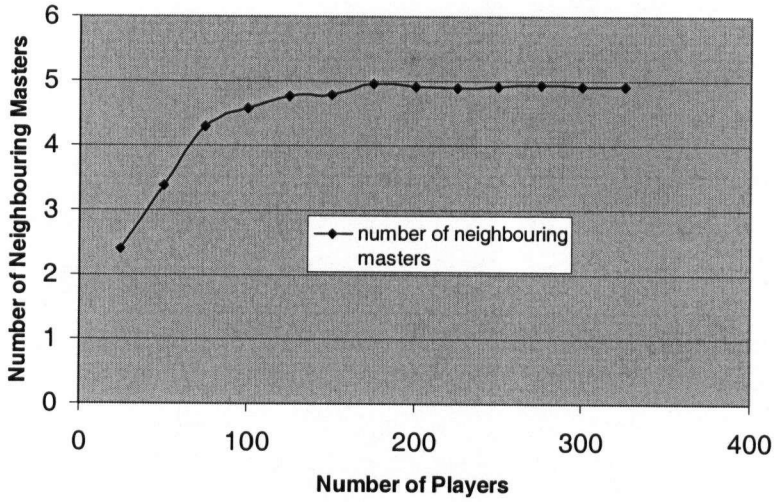


Figure 6-2 Average number of neighbouring masters

Figure 6-1 shows that the workload of a master node is fairly light, considering to the typical capacity of current broadband (between 512 kbps and 10 mbps). We also need to be aware that our game model is a fast-paced first-shooter action game, with frequent game updates. The workload can be a lot lighter, when the game model is MMOG (3-5 updates per second). Figure 6-2 shows that the average number of inter-master connections is around 3.38, in a 50-node trial. Our 50-node trial is equivalent to the 5000-node trial in the Proximity [19] experiments (a P2P architecture for MMOGs proposed recently), based on the average number of neighbours for each player. However, Proximity claims it requires an average of seven inter-coordinator connections per coordinator. In addition, the maximum number of neighbouring masters of MOPAR is bounded to six, and the minimum is zero, based on the structure of the hexagonal division, while Proximity's maximum may grow unbounded by the number of players, and the minimum number of connections is four.

In MOPAR, each slave player has only one extra connection with the master node other than the neighbours within AOI range. It is a marked improvement over other schemes, such as Solipsis and Voronoi. For example, based on Voronoi's experimental result, each node may require a maximum of around 20 extra connections with the

players outside of the actual AOI range in order to maintain the neighbourhood in a 250-node trail.

Another encouraging result regarding the scalability of MOPAR is that, as Figure 6-1 shows, the cost of sending messages grows at a lower rate than the cost of receiving messages, and the gap between them tends to get wider, when the number of players grows. This is because, as opposed to collecting the position updates from the slave nodes periodically, the master nodes send the neighbour notifying messages to the slave nodes only when necessary (when there are incoming neighbours). This result is also desirable, because uploading capacity is usually less than downloading capacity. The larger the population (or higher density), the bigger the saving can be achieved in bandwidth consumption.

6.1.2 Overall Bandwidth Consumption

Another measurement is the growth of overall bandwidth consumption across the network as the number of players grows. In some P2P-based neighbour discovery schemes, such as Kawahara et al's approach, in which each node exchanges a neighbour list with its nearest neighbours, the messages generated for neighbour discovery can grow exponentially by the average number of neighbours across the network.

Figure 6-3 shows that the growth in bandwidth consumption for MOPAR is linear as the number of players grows. Moreover, this growth has a tendency to curve downward, as a result of the aggregation of position updating through the inter-master communications. As the density increases, each master will have more slave nodes, therefore, each inter-master message will aggregate more position updates of the slave nodes.

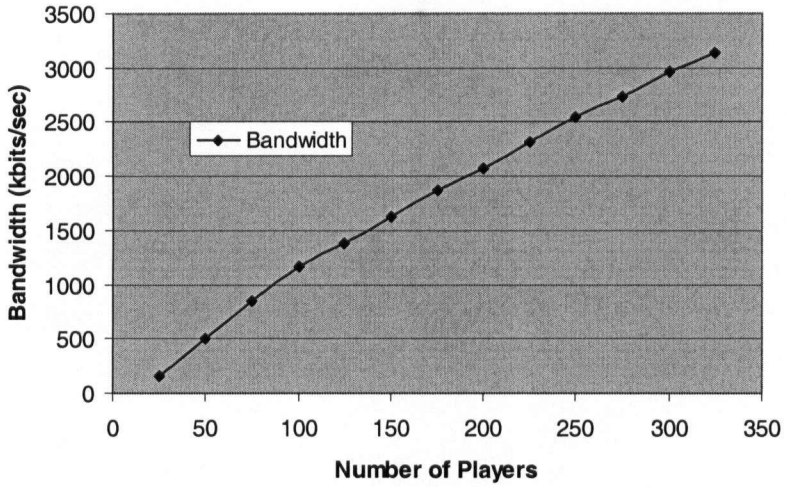


Figure 6-3 Effect of growth on overall bandwidth consumption

6.1.3 Overall Overhead Cost

The improvement that MOPAR provides over other P2P neighbour discovery schemes is certainly not without cost. As with any technological improvement, overhead cost should not outweigh the resulting improvement. We also investigate the overhead cost of MOPAR below.

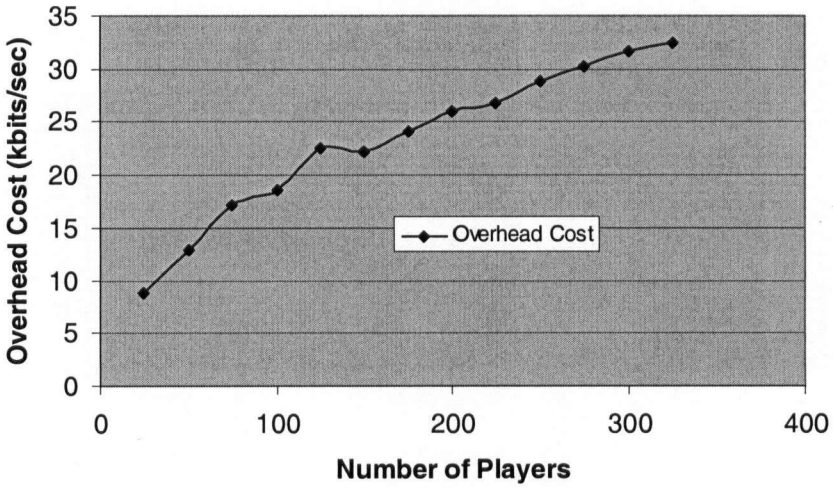


Figure 6-4 Growth of overhead cost

As discussed in Chapter 4, overhead cost results from cell switching of players and from role switching (master or slave). Figure 6-4 shows that the overall overhead cost that MOPAR generates to the network grows at a decreasing rate. The cause of this phenomenon is that the cost for moving into the next cell as a master node introduces more overhead than moving into the next cell as a slave node; However, while the density of the virtual world increases, the number of empty cells decreases. As a result, there will be fewer occurrences of moving into the next cell as a master, and thus, the growth of overhead cost will drop.

number of nodes	25	75	125	175	225	275	325
discovery cost	230.17	819.78	1365.81	1834.58	2290.56	2750.26	3190.44
overhead cost	9.36	16.48	19.41	23.82	28.18	30.98	31.95
percentage	4.07%	2.01%	1.42%	1.30%	1.23%	1.13%	1.00%

Table 6-1 Overhead cost vs. discovery cost

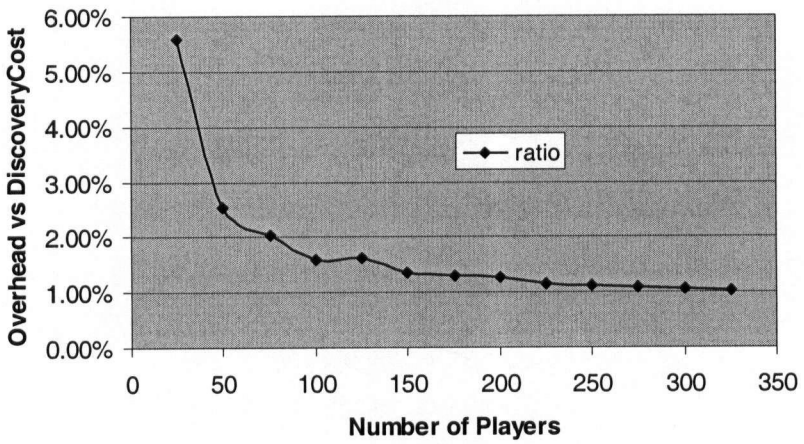


Figure 6-5 Ratio of overhead cost vs. discovery cost

Table 6-1 and Figure 6-5 show that overhead only accounts for a small percentage of neighbour discovery cost, and that this percentage tends to get even smaller, as the population grows. Overhead therefore will not explode the network, and meaning that the improvement that MOPAR provides will not be traded off by its overhead. Analysis

of other schemes such as Proximity, Solipsis, and Voronoi did not provide an evaluation of overhead costs.

6.2 Intelligent Masters Searching Algorithm

As we discussed in Chapter 4, the intelligent neighbouring masters searching algorithm is supposed to be more bandwidth efficient than the blind searching algorithm, as the former will intelligently prune unnecessary DHT searching, which is rather costly.

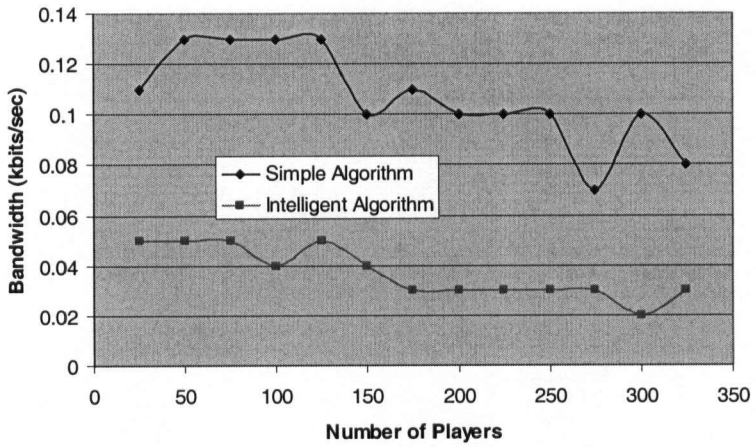


Figure 6-6 Intelligent neighbouring masters searching algorithm

Figure 6-6 plots the costs for cell switching to an empty cell (becoming a master node in the cell), when the intelligent and blind neighbouring masters searching algorithms are employed. Figure 6-6 shows that the intelligent searching algorithm is on average three times more efficient than the blind searching algorithm, in terms of the bandwidth cost per master node across the full range of trail nodes. The saving yielded by the intelligent searching algorithm is therefore consistent and significant.

6.3 Impact of Varying AOI Radius

How AOI radius can affect the performance of MOPAR is another important aspect that we need to investigate. In this section, we will investigate the impact on master workload, overall bandwidth consumption and overall overhead cost, when the AOI radius increases from 30 to 120, in increments of 10. The number of trail nodes is 200.

6.3.1 Impact on Master Workload

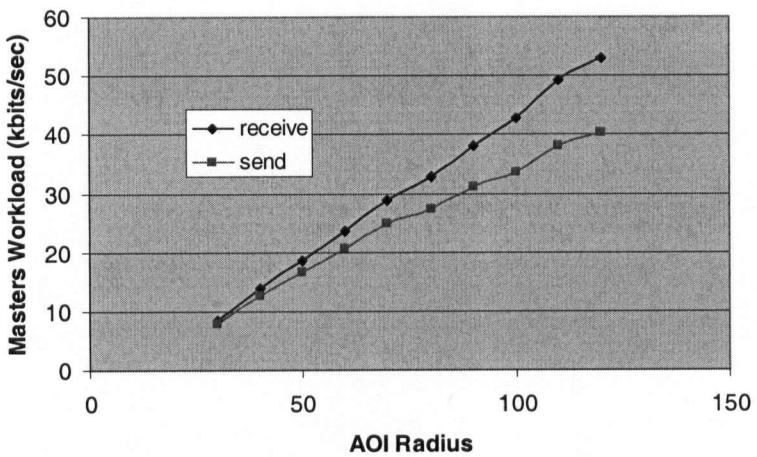


Figure 6-7 AOI radius and master workload

By intuition, the growth of the masters' workload is dependent on the number of players in the cells. When the AOI radius grows, each master node will need to coordinate more slave nodes in the cells, which grows by a factor of the square of the AOI radius, as shown in the following equation (see appendix A1 for derivation of this equation):

$$P_E = \frac{3\times\sqrt{3}\times N\times(AOI)^2}{2\times D^2}$$

P_E: expected number of players in a hexagonal cell

N: total number of players

AOI: AOI radius

D: virtual world's dimension

However, we can see from Figure 6-7 that the growth of the masters' workload grows almost linearly rather than exponentially. This is because the master nodes will aggregate more position updates when each of them covers a larger portion of the area. This helps to balance out the potentially fast growth of the workload. Meanwhile, we also observed that, the bandwidth consumption for sending tends to get less, compared to that for receiving, when the AOI radius increases. In general, the larger the AOI radius, the bigger the gap there will be between bandwidth consumption for sending and receiving.

6.3.2 Impact on Overall Bandwidth Consumption

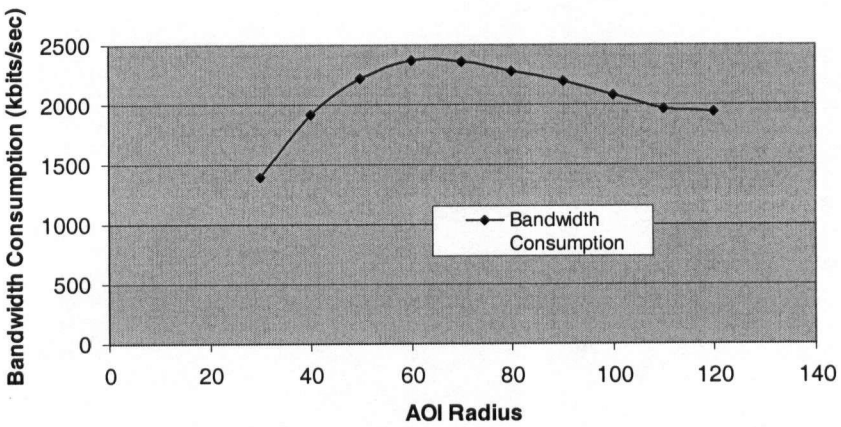


Figure 6-8 AOI radius and overall bandwidth consumption

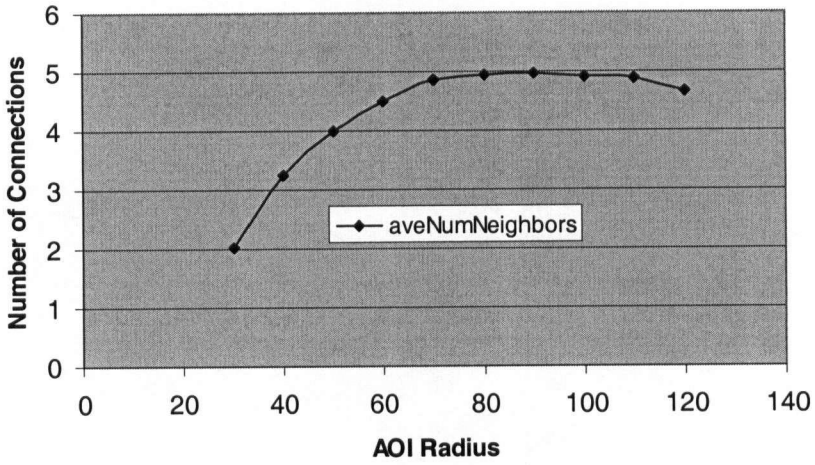


Figure 6-9 Average number of inter-master connections

We can see from Figure 6-8 that bandwidth consumption across the network increases when the AOI radius is between 30 and 60, but then starts to decrease. The reason for the overall bandwidth consumption increase at the beginning is that, when AOI radius increases below 60, the number of inter-master connections increases relatively quickly (see Figure 6-9), which drives rapid growth of overall bandwidth consumption. However, once the inter-master connections stop growing, the benefit from message aggregation takes over, causing overall bandwidth consumption begin to drop.

6.3.3 Impact on Overall Overhead Cost

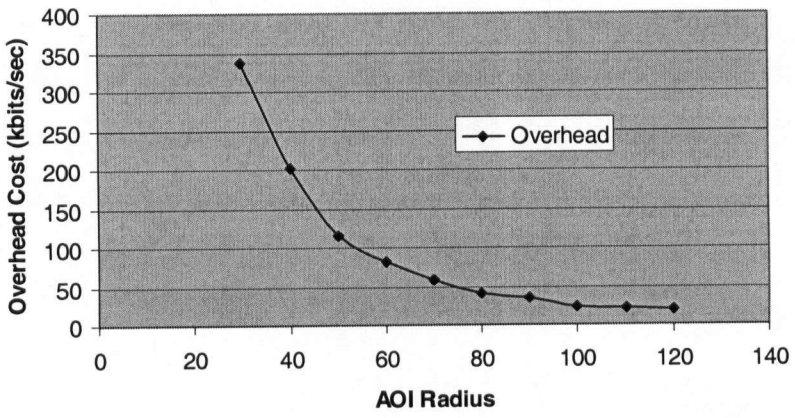


Figure 6-10 AOI radius and overall overhead

Figure 6-10 shows that the overall overhead drops drastically when the AOI radius increases. The reason is that, the larger the AOI radius, the lower the frequency of cell switching, and therefore less overhead cost. In extreme case, when the AOI radius is as big as the virtual-world dimension, the MOPAR overhead would become zero.

Based on our experimental results, we can conclude that, although increasing AOI radius will add more workload on each master node, the overall overhead of the network will decrease significantly; and at the same time, the overall bandwidth consumption across the network will also drop when the AOI radius is larger than 60. Therefore, as long as the workload does not exceed the capacity of the master nodes, better performance will be achieved globally with larger AOI radii.

6.4 DHT Performance

DHT performance is independent of AOI radius and the virtual world's dimension; it is determined by the nature of Pastry. In MOPAR, there are two types of DHT-specific costs. The first is incurred whenever a node sends a message to the home node of a cell, for example, for master-node registration. The second cost is incurred when a new node first joins the system, and inserted into the Pastry ring structure. This insertion causes the mapping between MOPAR cells and home nodes to potentially be adjusted. As a result, some home nodes may need to be switched from one Pastry node to the other. In this section, we investigate the impact of population growth on these two types of costs.

6.4.1 Home Node Searching Cost

As we discussed earlier, in Pastry a message is expected to take an average of $\log_{2^b}^{N}$ routing steps to reach its destination (where b is a configuration parameter and N is the number of nodes). To MOPAR, this expected number of routing hops is the cost of searching home nodes.

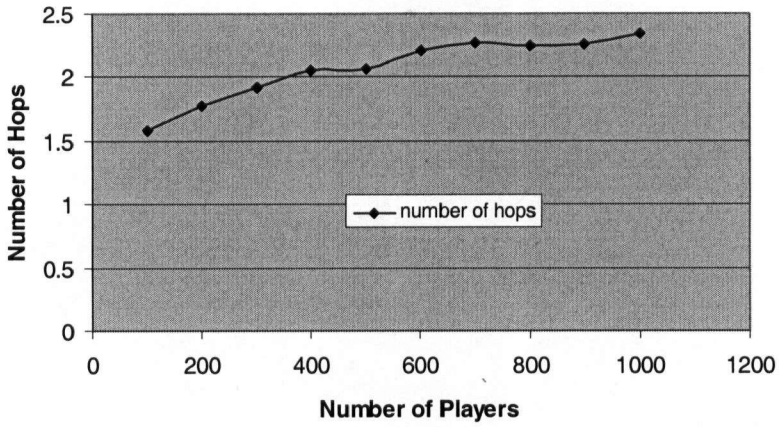


Figure 6-11 Average numbers of hops to reach the home nodes

Figure 6-11 shows that the cost of searching home nodes is bounded, and the average number of hops is between 1.5 and 2.5, when the number of players ranges from 100 to 1000, which is identical to the expected number of hops based on the theoretical calculation. As searching home nodes is the key to our scheme, and can happen fairly frequently, this result is a very positive sign regarding the scalability of the MOPAR system.

6.4.2 Home Node Transition rate

When a player joins the virtual environment, MOPAR will add the new node into Pastry. As a result, the home nodes for some cells may be changed. We want to investigate how frequently the changes occur, when the population of the virtual environment is growing. The metric we use to evaluate this transition rate is the average number of switchings per node.

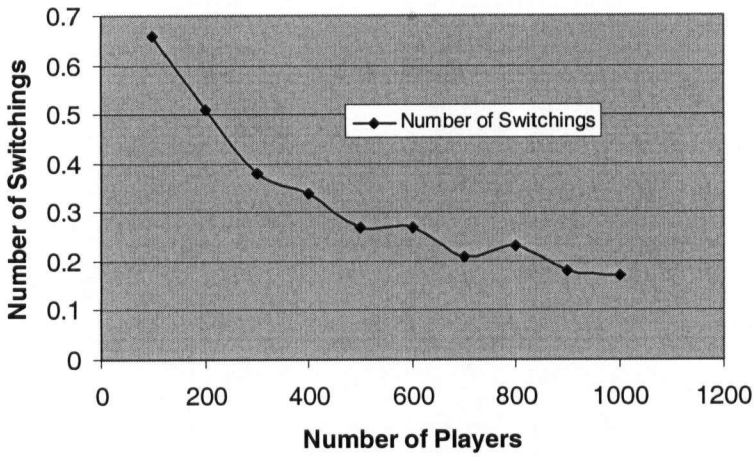


Figure 6-12 Home nodes transition rate

Figure 6-12 shows that the transition rate will drop drastically when the number of players grows. This result gives us the confidence that the MOPAR system will not be overloaded by home nodes transition for a virtual environment with a large population.

6.5 Topological Consistency

In this section, we investigate the topological consistency of the MOPAR scheme, that is, the ratio of observed neighbours versus actual neighbours of each player in the system. Theoretically, the MOPAR scheme itself does not cause any topological inconsistency, if zero network delay or message loss is assumed. However, in MOPAR, since master node initialization and node switching between cells require some extra network communication steps, these steps can be more sensitive to network delay or message loss, and potentially cause topological inconsistency. Therefore, we want to investigate the topology consistency of MOPAR when message delay or message loss is assumed under these scenarios. The metric we use to evaluate consistency is defined by Kawahara et al:

$$Consistency = \frac{1}{N}\sum_{i=1}^{N}\frac{P(i)}{Q(i)}$$

N: total number of players in the virtual world

P(i): the number of neighbours observed by player i

Q(i): the actual number of neighbours of player i

Therefore, the metric describes the average consistency of all the players.

6.5.1 Impact of Masters Initialization

Whenever a node enters an empty cell, it will become the master node of the cell. However, this master node initialization requires some extra communication steps, such as searching neighboring master nodes and notifying the neighbouring masters. Therefore, any network delay will prevent the master node from instantly starting its role. As the master nodes are the backbone in exchanging the position updates, the delay will result in the topological inconsistency for all players in the affected cells.

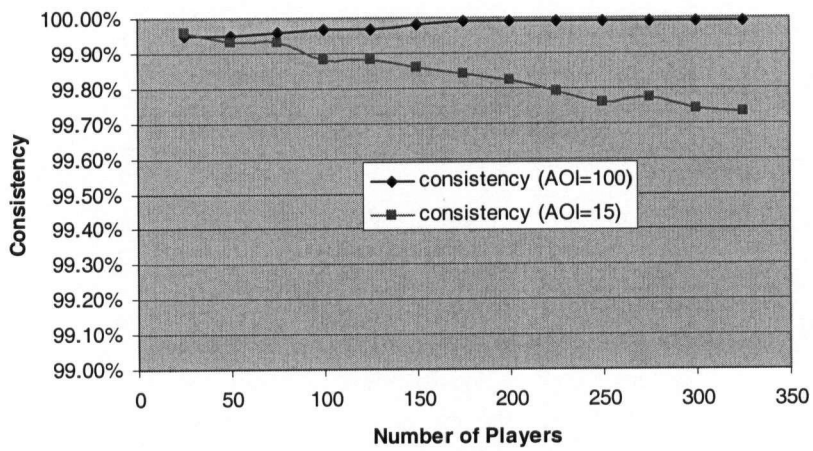


Figure 6-13 Effect of the delay of master start-up on topological consistency

In this experiment, we assume that due to network delay, master node start-up always gets delayed by one simulated step. Figure 6-13 shows that the topological consistency is fairly high, as it is over 99.90% when the AOI radius is 100 and over 98.80% when the AOI radius is 15. Note that when the AOI radius is 100, consistency

tends to go up when the population grows. The reason is that there will be less master node initialization in the high-density environment, since there are fewer empty cells. For the same reason, when the AOI radius is 15, because there are significantly more cells in the same dimension of the virtual world compared to when the AOI radius is 100, consistency drops when at the same range of trial nodes. However, we can expect that consistency will go up again when the population reaches a certain size.

We need to be aware that consistency here is based on simulation steps, and that every ten simulation steps account for one real-world second. Therefore, topological inconsistency of 0.1% or 0.2% of is hardly noticeable in the real game.

6.5.2 Impact of Cell Switching

When a node switches from one cell to another, it can be delayed due to the network delay. As a result, this node may not be detected as a new neighbour by others. To simplify the problem, we assume that the delay only affects updating of its own position (as we know, a master node can potentially affect more players).

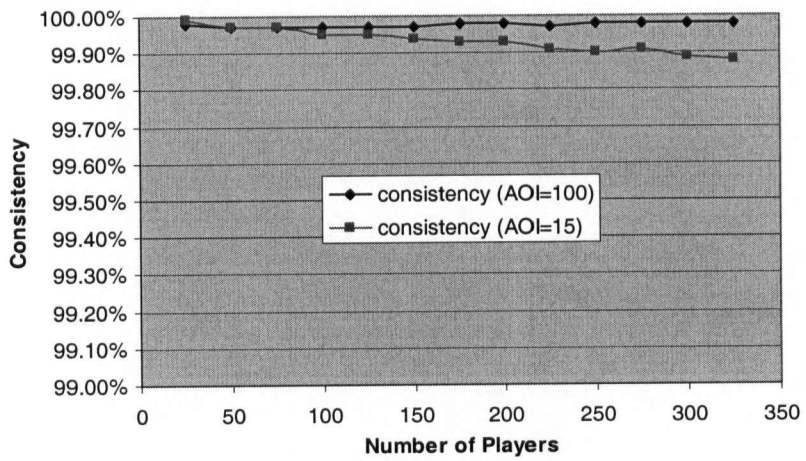


Figure 6-14 Effect of cell switching on topological consistency

We can see from Figure 6-14 that topological consistency is fairly high (over 99.97%), when the AOI radius is 100. In comparison, the consistency is a little

lower when the AOI radius is small, for example, 15. The reason is that, when a world map is divided into more cells, the frequency of cell switching gets higher, resulting in lower topological consistency. In conclusion, MOPAR is very robust to network delay and message loss.

Chapter 7

Conclusions and Future Work

7.1 Summary

While large-scale networked virtual environments and their applications such as massively multiplayer online games are emerging, their most popular current system architecture, the client-server model, significantly suffers from scalability. As the number of players in a MMOG system can grow virtually without limitation, servers can quickly become a bottleneck. The P2P model is the natural solution for addressing this scalability issue, due to its desirable properties such as resource growing and decentralized resource consumption. However, the P2P model makes some things challenging that are rather simple in the client-server model. One of them is the interest-management mechanism. Without such a mechanism, scalability is not achievable in the P2P model, since every peer has to broadcast a state update, for example, position update, to every other peer in the game world. This not only floods the whole network, but forces every peer to receive many irrelevant messages.

We propose a novel hybrid infrastructure, MOPAR, for supporting scalable MMOGs, with a focus on providing a distributed interest-management mechanism. Our

scheme takes advantage of both the DHT overlay and unstructured P2P architecture, so that our infrastructure inherits the benefits of both to provide a system that is self-organizing and fault-tolerant with low latency and low overhead.

Our system divides the game map into hexagonal cells. Each cell has a corresponding home node through DHT mapping; the home node is used for building a hierarchical structure, in which the master node acts as coordinator for the cell. Our hierarchical design and direct communication scheme allow the system to exchange messages in an efficient way. Our design also supports a continuous view for nodes in the virtual world.

We have also implemented a simple game simulation and visualization tool, by which players move randomly and communicate with their neighbours. The visualization tool allows users to interact with the infrastructure to explore and understand the design of hexagonal zoning, as well as the roles of master and slave nodes.

7.2 Future Work

There are four major directions for further research. First, if there are too many slave nodes appear in one cell, the master node may potentially become a bottleneck. To deal with this issue, our initial idea is to assign a secondary master node to offload some of the workload from the main master node. Dynamically splitting and merging cells may also be a possible solution.

Second, we realized that the distributed interest-management scheme should not be limited to MMOGs. Any application in which each entity is location based and mobile can be potential users of our infrastructure, for example, mobile phones, laptops, or even vehicles supported by GPS system. The potential applications are as following:

- Service discovery – Mobile devices dynamically locate the nearest resources or services (i.e. printers, mobile doctors).

- Mobile content discovery and sharing -- Contents (personal digital asset) discovery and sharing can be more widespread, dynamic and efficient via more capable mobile devices or handsets.
- User profile matching – People can locate other people or places through profile matching via their wireless devices, such as Bluetooth enabled cell phones (with GPS). Master nodes will watch not only location or movement of neighbours but also profile info (metadata or attributes) of neighbours.

Third, we will extend our infrastructure to address other requirements of MMOGs, such as state persistency, event consistency, and security. Although state persistency is not a focus of this thesis, our infrastructure will readily support it with little additional work. Basically, as Pastry provides the facility for storage persistence and replication, we can use the home node to store a portion of the global game states. There are some existing proposals for event consistency, such as bucket synchronization [8]. Our infrastructure is flexible enough to bring in other components to meet more requirements.

Finally, we will further develop our system and integrate it with a real MMOG, so that we can evaluate our infrastructure in a real and large-scale P2P network. Eventually, we want our infrastructure to be extensively used as a network framework for easily developing and deploying user-defined MMOGs in the community.

Appendix A

Derivations

A.1 Expected Number of Players per Hexagon

Virtual world's dimensions = D x D

Number of players = N

AOI radius = AOI

Expected number of players per hexagon = P_E

Based on our design, a hexagon's side length is equal to AOI radius. Therefore, the area of a hexagon is as follows:

Area of hexagon = $\frac{3\times(AOI)^2\times\sqrt{3}}{2}$.

Number of hexagons of a game world is the virtual world's dimensions divided by the area of a hexagon, or $\frac{D^2\times 2}{3\times(AOI)^2\times\sqrt{3}}$

The expected number of players per hexagon is then equal to the total number of players divided by the total number of hexagons of the virtual world, yielding the following equation:

$$P_E = \frac{3 \times \sqrt{3} \times N \times (AOI)^2}{2 \times D^2}$$

Bibliography

[1] Anthony Yu, Son Vuong, MOPAR: A Mobile Peer-to-Peer overlay architecture for the interest management of massively multiplayer online games. In Proceedings of NOSSDAV 2005. ACM Press.

[2] A. Rowstron, P. Druschel. Storage management and caching in PAST, a large-scale persistent peer-to-peer storage utility. In Proc. ACM SOSP'01, Banff, Canada, Oct. 2001.

[3] A. Rowstron et al. Pastry: Scalable decentralized object location, and routing for large-scale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM international conference on Distributed Systems Platforms (Middleware). November 2001.

[4] Ashwin R. Bharambe, Sanjay Rao, and Srinivasan Seshan. Mercury: a scalable publish-subscribe system for internet games. In Proceedings of the first workshop on Network and system support for games, pages 3-9. ACM Press, 2002.

[5] B. Knutsson, H. Lu, W. Xu, B. Hopkins. Peer-to-peer Support for Massively Multiplayer Games. In INFOCOM,Mar, 2004.

[6] Butterfly.net, Inc. The butterfly grid: A distributed platform for online games, 2003. www.butterfly.net/platform/.

[7] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-0101141, UC Berkeley, April 2001.

[8] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the internet. IEEE Networks magazine, 13(4), July/August 1999.

[9] E. J. Berglund and D. R. Cheriton. Amaze: A multiplayer computer game. IEEE Software, 2(1), 1985.

[10] EverQuest: http://everquest.station.sony.com/

[11] E. Prasolova-Forland, Monica Divitini. Collaborative virtual environments for supporting learning communities: an experience of use. In Proceedings of SIGGROUP, pages 58-67, New York, 2003. ACM Press.

[12] FreeNet Website: http://freenet.sourceforge.net

[13] GameDev.net: Coordinates in Hexagon-Based Tile Maps http://www.gamedev.net/reference/articles/article1800.asp

[14] Gnutella Website: http://www.gnutella.com

[15] Honghui Lu, Bjorn Knutsson, Margaret Delap, John Fiore, and Baohua Wu. The Design of Synchronization Mechanisms for Peer-to-Peer Massively Multiplayer Games.

[16] Ion Stoica, Robert Morris, David Karger, Fans Kaashoek, and Hari Balakrishnan, Chord: A scalable Peer-to-Peer lookup service for internet applications. In Roch Guerin, editor, Proceedings of SIGCOMM-01, volume 31, 4 of Computer Communication Review, Pages 149-160, New York, August 27-31 2001. ACM Press.

[17] J. E. Jaramillo, L. Escobar, and H. Trefftz. Area of Interest Management by Grid-Based Discrete Aura Approximations for Distributed Virtual Environments. In proceedings of the 6th Symposium on Virtual Reality (SVR 2003), pp 342-353, Ribeirao Preto, SP - Brazil, October 15-18, 2003

[18] J. Keller, G. Simon. Solipsis: A Massively multi-Participant Virtual World. In Proc. of PDPTA 2003. Pg. 262-268. 2003.

[19] Kamran S. Makik. Proximity: A scalable P2P Architecture for Latency Sensitive Massively Multiplayer Online Games. Master's thesis, University of British Columbia, 2005.

[20] Katherine L. Morse. Interest management in large-scale distributed simulations. Technical Report ICS-TR-96-27, University of California, Irvine, 1996.

[21] Miguel Casro, Michael B Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman, An evaluation of scalabe application-level multicast built using peer-to-peer overlays. In Infocom'03, April 2003.

[22] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, P. T. Barham. Exploiting Reality with Multicast Groups. IEEE Computer Graphics and Applications, Volume 15, Issue 5, Pages: 38-45, Sept. 1995.

[23] Napster Website: http://www.napster.com.

[24] Paul Bentner and Mark Terrano. GDC 2001: 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond, March 2001. www.gamasutra.com/features/20010322/terrano_01.htm.

[25] S. Hu, G. Liao. Scalable Peer-to-Peer Networked Virtual Environment. In SIGCOMM'04 workshops.

[26] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker, A scalable content-addressable network. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pages 161-172. ACM Press, 2001.

[27] Shinghal, S., and Zyda, M., Networked Virtual Environments, ACM Press, 1999.

[28] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M.A. Lopez. Indexing the positions of continuously moving objects. In Proc. of the 2000 ACM SIGMOD int'l. conf. On Management of Data, pages 331-342, 2000.

[29] T. Limura, H. Hazeyama, Y. Kadobayashi. Zoned Federation of Game Servers: a Peer-to-Peer Approach to Scalable Multi-player Online Games. In SIGCOMM'04.

[30] Vuong, S., Li, J. ECSP: an efficient cluster based P2P architecture. In Proceedings of the International Conference on Internet Computing, Jun 2003.

[31] Y. Kawahara, T. Aoyama, H. Morikawa. A Peer-to-Peer Message Exchange Scheme for Large-Scale Networked Virtual Environments. Telecommunication Systems Vol. 25 Issue 3, Pages 353 370, 2004.

[32] Zona Inc. Terazona: Zona application framework white paper, 2002. www.zona.net/whitepaper/Zonawhitepaper.pdf.