An Efficient Interest Management System for Networked Virtual Environment

by

Yunjing Zhang

B.Sc., The University of British Columbia, 2004

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

(Vancouver)

August 2010

Abstract

Over the past decades we have witnessed an enormous development in multiplayer games. They have evolved from the initial state with two players on the same computer to the current state with tens of thousands simultaneous players in a virtual world via the Internet. In dealing with the large scale of massive multiplayer on-line games (MMOGs), various peer-to-peer (P2P) solutions have been proposed. Although they have improved the scalability of MMOGs in various degrees, they encounter new serious challenges in interest management. No existing works have provided a satisfying solution though they all have some attractions.

In this thesis, we propose a fully distributed P2P infrastructure to achieve interest management for MMOGs. Our system divides the virtual game world into cells. Players in each cell are connected to the cell's master-node that is selected from the players in the cell. Master-nodes are also connected with master-nodes of adjacent cells, forming a backbone network. Players provide information about their position and area of interest (AOI) to their master-node. The master-node informs the players in the cell about which other players are in their AOI so that the players can obtain game updates directly from players in their AOI. A routing scheme is applied on top of the backbone network so that master-nodes can efficiently exchange information about nodes in each cell. To help players to locate master-node of each cell, a distributed hash table (DHT) is used for bookkeeping. As illustrated by our experiments, our proposed system is not only scalable and effective; it also provides the benefits of high responsiveness and reliability without compromising any game features.

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my supervisor, Dr. Son Vuong, for his inspiration, guidance and encouragement. Without him, it would have been impossible to complete this thesis. I am also grateful to Dr. Charles Krasic for being my second reader and for his useful comments that have improved this thesis.

Thanks to my colleagues in NIC lab, especially Peiqun Yu and Juan Li, for their useful discussions, comments, and encouragement.

I gratefully acknowledge the financial support from the Computer Science Department at UBC in the forms of TA and RA.

Yunjing Zhang

The University of British Columbia

August 2010

Chapter 1

Introduction

A massively multiplayer online game (MMOG or simply MMO) is a networked game capable of simultaneously supporting typically thousands of players playing in a single persistent game world. The game is played over the Internet. And the players interact, through game entities such as avatars, with other players in a networked virtual game environment. In a typical game, a player sees a window of the virtual universe and can perform actions such as moving the avatar, picking up objects, or communicating with other players etc.

MMOGs have two distinguishing features. First, they host a large number of players in a single game world, and all of those players can interact with each other at any given time [5]. The magnitude of the number of concurrent players sharing a single game world is typically on the order of 10^4 or more. Second, MMOGs have persistent game state, including state of virtual game world and state of each game player. Unlike other types of networked games that end after some goal is completed, a MMOG game can continue indefinitely regardless of joining and leaving of individual game players. A player's state (such as position of his/her game avatar in the world, and his/her avatar's abilities, health and possessions etc) in the game at the point he/she leaves the game is

saved and restored when he/she comes back. Changes made to the game world by any player are recorded and remain effective even after the player who made the changes has left the game.

There are several types of massively multiplayer online games [5][40], among which, first-person shooters (FPS), role-playing games (RPG), and real-time strategy games (RTS) are three well-known types. In a FPS, the main goal is typically to kill the other players with various weapons from the virtual world. In a RPG, one of the primary goals is to develop one's character in the game by increasing abilities and gathering new equipment. These goals are achieved by grouping with other players to explore new lands, kill monsters, and sometimes by fighting with other players. In a RTS, a player is in control of units in a virtual world, where a unit might be a soldier, vehicle, or building. The player acts as the commander of the units and instructs them on what actions to take. Players compete with each other to either destroy all the units of the other player, or capture some vital piece in a game.

MMOGs are interactive massively distributed systems. Since their game worlds are Networked Virtual Environment (NVE), which is computer-generated synthetic worlds allowing simultaneous interactions of multiple participants [25], MMOGs are recognized as applications of NVE as well. As so, when designing a MMOG we have to consider a number of problems [5][25][33]: (1) consistency – the difficulty of sharing a game state consistently across participants; (2) scalability – how well a system adapts to the number of players; (3) security – the prevention and detection of cheats; (4) persistency – provision of long-term (between game sessions) persistence of game state; (5) responsiveness – system response speed which varies between applications with different latency tolerances; (6) reliability – fault tolerance in event of a sudden network failure.

Although some of the above problems are interrelated [33] (addressing one particular issue impacts others), consistency and scalability are considered the most important ones since these two decide the usability of a MMOG. Improving scalability

without compromising the other requirements, especially game consistency is the design goal of a successful P2P based MMOG.

1.1 Motivation

Traditionally, MMOGs use client-server communication architecture. The clients running on the players' machines convert players' commands to events and pass them to the server; the server, on the other hand, orders these events, resolves conflicts, calculates game states in the simulation, sends game updates (filtered based on the area of individual player's interest) to clients, and also acts as a central repository for data as well.

In this centralized architecture, the server acts as a single authority, a data storage and message-distributing centre. This architecture makes resolving conflicts to achieve game consistency easy and also makes data persistency and player account management easy. It also makes the game secure. However, it suffers from the scalability problem for the following reasons; First, since messages between players are always forwarded through the server, the delay introduced could become intolerable when the number of concurrent players increases. Second, traffic at the server increases with the increased number of players, resulting in localized congestion when the number of concurrent players reaches to a certain level. Third, this architecture is limited by the computational power and storage capacity of the server. The system could become very slow when the number of concurrent players exceeds the server's capacity. Increasing the number of concurrent players could result in significant degradation in system responsibility and performance because the server is a potential computation, storage, and network bottleneck. Although we can throw in more servers, disk farms, and bandwidth lines to temporarily alleviate the problem, this will significantly raise the system complexity and cost.

To solve the scalability problem, recent research in MMOG and VNE has looked into P2P networks for solutions. And various distributed, P2P-based architectures for MMOGs have been proposed [5][25][1][19][29][31][36]. These existing architectures try to overcome the scalability problem in client/server model by harnessing resources (computation, storage, and bandwidth) of players' machines: the more players join the game, the more resources the game has. When demand on the game system increases as the number of concurrent players goes up, the total capacity of the system also increases.

However, while these architectures try to take advantage of distributed system to address scalability issue, they encounter difficulties in interest management because the game state is entirely distributed in the system. How to share a game state consistently across players in P2P-based MMOGs becomes a serious challenge.

The virtual environment of a MMOG is so large that an action at one place affects only part of the game world, i.e. its neighbourhood. Therefore, a local action does not need to be propagated to remote players. In addition, because of the large scale of MMOGs, broadcasting all information in the game to each player is not viable. Players should receive only the information that they are concerned. Whether a piece of information is relevant to a player or not depends on the play's area of interest (AOI), e.g. on his/her avatar's visibility or perception, which is usually defined as a sphere/circle around the avatar by the interest management. Normally visibility has a different definition for each object. A radar object has a much wider domain of interest (but less detail) than an infantry unit. The size of a player's area of interest may also change over time: it expands when the user has access to radar, and shrinks when the player's walkie-talkie runs out of battery. Interest management [33] is the art of defining areas of interest for each player. It also manages the game synchronization at the intersecting regions of interest.

In client/server model, interest management is performed at the server side by filtering data according to areas of the player's interest. Being disseminated from the

same source, the game updates received by any two players are identical for intersecting regions of their interest. Therefore, the game state is consistent across players.

On the other hand, P2P architectures don't have any central server. The game states are distributed in the system. The interest management of P2P architectures must ensure players gain all game updates occurred in their AOIs and have consistent game view in the intersecting regions of players' interest. However, due to the lack of global view of the game world and synchronization in the system, how to effectively locate game updates and how to efficiently obtain only relevant game updates for players is not an easy task for interest management in distributed systems. So far, no existing works have provided a satisfying solution though they all have some attractions. A suitable P2P architecture that enables an effective interest management to satisfy both scalability and consistency requirements of MMOG remains as a challenging research topic. Therefore, one of the most important goals of this paper is to design a P2P architecture along with effective interest management algorithms to support MMOGs to achieve scalability and game consistency.

1.2 Thesis Contributions

In this thesis, we propose a structured P2P architecture supporting MMOG to achieve good scalability, effective interest management, and efficient communication. In the proposed system, the virtual game world is divided into cells. Players in a cell are connected to the cell's master-node that is selected from players in the cell. Master-nodes are also connected with master-nodes of adjacent cells, forming a backbone network. Players provide information about their position and AOI to their master-node. The master-node of a cell provides AOI neighbour discovery service to players in the cell. In this way, a player in the system is fully aware of his/her AOI contents and can obtain game updates directly from players in his/her AOI by establishing communication

connections with them. This architecture can reduce the latency for messages propagation between players and avoid delivering messages to unrelated players.

In order to provide neighbour discovery service, master-nodes exchange information about nodes in each cell through connections between master-nodes. To efficiently and effectively route messages on the backbone network, we have designed a scheme for master-nodes to subscribe and obtain desired information through their neighbour master-nodes. Utilizing only a small amount of topology info, the scheme can efficiently deliver messages without duplicates and avoids delivering messages to unrelated master-nodes. We adopt a distributed hash table (DHT) into our system to help players to locate master-nodes and to help maintain our P2P overlay network.

Our P2P architecture is structured, self-organized, and reliable. Its interest management fulfils both scalability and game consistency requirements of MMOGs without compromising any game features. We expect our approach can be used as a generic P2P game network foundation for the MMOG community, enabling numerous user-defined P2P MMOGs to be built on top of it.

Our thesis makes the following contributions:

- We proposed an interest-management scheme that facilitates players to obtain game updates directly from players in their AOI, resulting in a timely and efficient message delivery mechanism that can even satisfy the requirements of real time games.
- We proposed an efficient P2P overlay structure and communication protocol to facilitate effective discovery of neighbours in each player's AOI. This technique is not restricted to the MMOG. It can also be applied to any location-based application in which entities are mobile.

- We designed an efficient subscription and routing scheme to apply on the backbone network for master-nodes to exchange local information. The scheme guarantees players have continuous view of the virtual game world as their avatar wandering around. It enables players to have different AOI sizes and also allows the area of a player's interest to expand or shrink during game.
- We proposed a load-balancing scheme to reduce the load from overloaded master-nodes that are in the hot spots of a game, i.e. their cells have too many players.
- We proposed an effective DHT overlay to facilitate efficient neighbour discoveries. Basically, this overlay is used to build and maintain the hierarchical architecture, making it self-organized and fault tolerant.
- We implemented a prototype of the proposed system and extensively evaluated it. Furthermore, we implemented a visualization tool for demon.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 introduces background information and related work in P2P MMOGs. Chapter 3 describes the system design, which includes system structure, components, general operation, and functionalities. Chapter 4 first presents the subscription and routing scheme for master-nodes to exchange messages through the backbone network and then algorithms for structural maintenance. Details regarding implementation and decisions made for the prototype implementation are discussed in Chapter 5. Chapter 6 presents the empirical analysis of the system's performance. Chapter 7 concludes the thesis and discusses potential directions for future research.

Chapter 2

Background and Related Work

In this chapter we first present the background information about P2P technologies which enables our proposed framework architecture. In particular, we present one of the most important P2P structures – DHT-based P2P structure and its representative implementation, Pastry. Then we discuss some recently proposed related work on P2P-based MMOGs.

2.1 Peer-to-peer Network

2.1.1 Background

Because of its three distinct features: decentralization, scalability, and fault tolerance P2P network comes naturally to mind when we seek solution for the scalability problem stated in Chapter 1. A P2P network is a distributed system without any central servers; all peer nodes simultaneously function as both “clients” and “servers” to the other nodes on the network. All peer nodes provide resources, including bandwidth, storage space, and computing power to other peers in the P2P network. Therefore, as more nodes join in the network, the demand on the network increases, while the total

capacity of the network also increases. This property of P2P network eliminates the scalability problem of client/server model. Moreover, unlike client/server architectures, in which central servers are potential points of failure, P2P networks do not have single point of failure in the system because data are replicated over multiple peers.

P2P networks can be classified as unstructured or structured based on how the nodes are linked to each other in its overlay network. Here, a node is "linked" to another means that the former knows the location of the latter and therefore there is a directed edge from the former to the latter in the overlay network. Employing a globally consistent protocol, structured P2P networks guarantee that a node can efficiently retrieve any piece of data distributed in the network within a small number of hops. However, the unstructured do not have such property. By far the most common type of structured P2P networks is the distributed hash table (DHT). Our proposed system is based on one of well-known DHT implementation – Pastry.

2.1.2 DHT and Pastry

Distributed hash tables (DHTs), such as CAN [26], Chord [16], Pastry [3], and Tapestry [7], are a class of decentralized distributed systems that partition ownership of a set of keys among participating nodes, and can efficiently route messages to the unique owner of any given key. A partitioning scheme splits ownership of the key-space among the participating nodes. Each node maintains a set of links to others; together these form the overlay network. For any key *k*, a node either owns *k* or has a link to a node that is closer to *k* in terms of the key-space distance. Routing of a message to the owner of any key *k* can use a greedy algorithm: at each step, forward the message to the neighbour whose ID is closest to *k*. When there is no such neighbour, then it must have arrived at the closest node, which is the owner of *k*.

Pastry is a structured P2P routing protocol that implements the DHT abstraction. In a Pastry network, each node has a unique, uniform, randomly assigned nodeId in a circular 128-bit identifier space. Given a message and an associated 128-bit key, Pastry reliably routes the message to the live node whose nodeId is numerically closest to the key. In a Pastry network consisting of N nodes, a message can be routed to any node in less than $\log_{2^b} N$ steps on average, where b is a configuration parameter. Each node stores only $O(logN)$ entries, where each entry maps a nodeId to the associated node's IP address. Specifically, a Pastry node's Routing Table is organized into $\lceil \log_{2^b} N \rceil$ rows with $(2^b\text{-}1)$ entries each. Each of the $(2^b\text{-}1)$ entries at the row n of the Routing Table refers to a node whose nodeId shares the first n digits with the present node's nodeId, but whose $(n+1)^{th}$ digit has one of the $(2^b\text{-}1)$ possible values other than the $(n+1)^{th}$ digit in the present node's nodeId. Pastry stores multiple candidates per Routing Table entry to increase availability. In addition to a Routing Table, each node maintains a Leaf Set, consisting of $L/2$ nodes with the numerically closest larger nodeIds and $L/2$ nodes with the numerically closest smaller nodeIds, relative to the present node's nodeId. L is another configuration parameter. In each routing step, the current node forwards a message to a node whose nodeId shares with the message key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current node's ID. If no such node is found in the Routing Table, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node but is numerically closer to the key than the current nodeId. Such a node must exist in the Leaf Set unless the nodeId of the current node or its immediate neighbour is numerically closest to the key.

2.2 Related Work

As mentioned before, propagating messages to all players is neither necessary nor viable, because (a) the scale of the game world is too large and the number of players is too high in MMOGs; (b) the areas of player's interest is only limited to the player's local

area. Only events happened within a player's AOI are relevant to the player and must be sent to the player. Therefore, the main challenge of designing P2P architectures for MMOG is how each node receives relevant messages as they move around. In this section, we discuss some of the existing P2P solutions for MMOGs proposed in recent years.

2.2.1 Subscriber/Publisher Model

A subscriber/publisher model, such as *Mercury* [4], *NPSNET* [22], and *SimMud* [5], works by creating a mechanism whereby the game creates different types of content to be published in the game. Interested parties subscribe to the publisher of the information they desire. When a publisher publishes some information, the information is only propagated to interested subscribers without flooding to other unrelated players.

Let us take the *SimMud* designed by Knutsson et al. as an example. The *SimMud* uses the DHT structure Pastry [3] and Scribe [21]. The virtual world is divided into regions, each of which maps to a multicast group through Scribe [21]. Players in each region, as publishers, send updates to the region coordinator, the root of the multicast tree. Then the players that subscribe to multicast groups within their area of interest can receive updates multicast from the coordinators of the regions.

The publisher/subscriber model can guarantee the scalability of the system and also guarantee players to receive their desired game updates. However, it cannot guarantee how fast a player can get the information he is interested in due to message relays. It takes average several hops between players for a message to finally reach its destination recipient. Message latency could be up to several seconds.

2.2.2 Recursive Spatial Partition

Rhalibi et al. designed an interesting MMOG infrastructure [36] consisting of multi-layers of P2P networks. The virtual world space is recursively divided to create different level in a hierarchy of communication. In each iteration, a region is partitioned into sub-regions, controlled by sub-region super-nodes that are connected to form a P2P network. Players in the lowest sub-region form a group. Updates generated within a sub-region are broadcasted in the sub-region group. If a player needs to obtain updates outside his/her local sub-region, the player has to submit a subscription list to his super-node. The super nodes then try to locate the subscribed information from its neighbor super-nodes or parent super-nodes in the super-node hierarchy. Figure 2-1 shows the architecture.

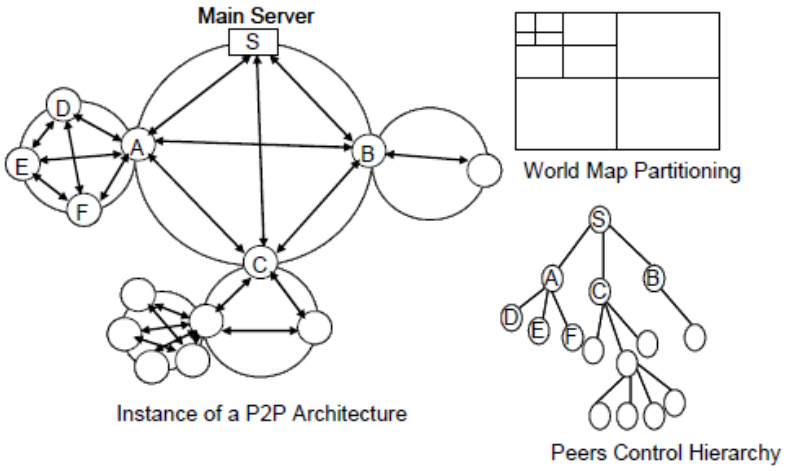


Figure 2-1 Architecture Proposed by Rhalibi et al.

This scheme can guarantee scalability of the system. The hierarchal communication system built in the system does benefit message exchanges between some players. However, at the same time it worsens message exchanges between some other players. For example, messages from players close in the virtual game world but separated into different sub regions may have to travel up and down the whole hierarchy to reach the

other nodes. The more levels the hierarchy has, the higher message latency these players have.

2.2.3 AOI Neighbor Discovery

Some designs advocate players to obtain updates directly from players in their AOI. To establish connections with nodes in his/her AOI, a player has to know which nodes are in his/her AOI. Therefore, the main challenge in these approaches becomes neighbour discovery, and may be stated as: given a node's position and AOI radius, find all neighbouring nodes within the AOI.

Kawahara et al. propose a fully distributed scheme [29] where each node directly connects with a fixed number of nearest neighbours and constantly exchanges neighbour-lists to discover new nodes. However, list exchanging between all the players may incur too much overhead, as much of the transmission is redundant. Moreover, when separated by long distances, nodes may also lose mutual contacts and cause overlay partitions.

In *Solipsis* [18], each node connects to all the neighbours within its AOI. Neighbour discovery is achieved by notifications of approaching foreign nodes from a player's known neighbours. However, a node approaching an AOI may not be known to all of the AOI residents who watch their topology neighbours for incoming virtual neighbours, resulting in incomplete neighbour discovery.

Hu et al's *Voronoi-based Overlay Network (VON)* [25] employs the Voronoi diagram to maintain its P2P topology. Each node connects with the nearest neighbours called enclosing neighbours and propagates position updates to other nodes by message forwarding. However, simultaneous leaves of adjacent enclosing neighbours of a given node could result in unrecoverable broken overlay topology, damaging both global connectivity (whether the overlay is fully-connected) and local awareness (whether each

node is fully aware of its AOI neighbours) [25]. Figure 2-2 depicts the aforementioned scenario.

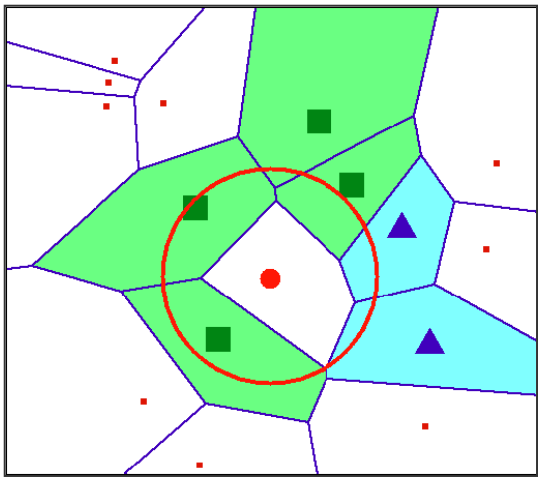


Figure 2-2 Fails to recover when adjacent nodes leave simultaneously

In *MOPAR* [1], player nodes are organized with the help of a DHT overlay. This overlay can ensure local awareness and global connectivity. The virtual space is divided into cells. Nodes in the same cell form a group and are severed by a master-node selected from them. The master-node notifies its slave-nodes (nodes in its cell) of changes in their AOI by maintain a list of all nodes in its local and its adjacent cells through communication with its slave nodes and neighbouring master-nodes. The limitation of MORAR is that AOI sizes of players are unified and fixed in the game.

In summation, there have been many strategies proposed to solve the scalability problem of MMOGs. However, most of these existing approaches suffer from varying deficiencies in their interest management that cause problems when updating players on game states. Players either cannot receive game updates in time because of high message latency or miss game updates completely unawares. This results in game inconsistent across players. We need better interest management algorithms for P2P architecture to achieve both scalability and game consistency for MMOGs.

Chapter 3

System Design

In this chapter, we present the design of our P2P solution for MMOGs. The proposed solution not only resolves the scalability problem but also facilitates efficient interest management. We first introduce the interest management scheme used in the system and then present the system architecture. Later, we detail the functionalities and general operations of two P2P overlays that form the system infrastructure.

3.1 Overview

Our design, which focuses mainly on interest management, is comprised of two structured distributed P2P overlay networks. Each overlay is composed of all of player nodes in the game, but each provides a different service. One of the two overlays is *Neighbour Discovery Overlay* (NDO) and the other is rendezvous overlay.

As its name suggested, the main function of *Neighbour Discovery Overlay* is to help player nodes to find all game entities in their areas of interest. To achieve the goal, all nodes in *Neighbour Discovery Overlay* are organized in a hierarchical structure and a communication protocol is applied on top of it. Changes in a player's AOI, for instance, a game entity enters or leaves the AOI, will soon be sent to the player. Even if objects in

the game world have different visibility and the size of players' AOI changes over time, the topology and the protocol employed ensure that each player node knows all game entities in its AOI. Therefore, every player could open direct connections to those peers in his AOI to receive their game updates directly. It is the basic idea of the interest management for our system.

The rendezvous overlay in our infrastructure is utilized to help construct and maintain the *Neighbour Discovery Overlay*. The peers rely on the rendezvous overlay to join the *Neighbour Discovery Overlay*. Also, the rendezvous overlay helps fix node crashes on backbone of *Neighbour Discovery Overlay* to make it fault tolerant.

3.2 Neighbour Discovery Overlay

In our design, the game world is partitioned into hexagonal cells. Figure 3-1 shows the virtual game world with dividing hexagonal cells. In the following sections, we present the detailed protocol and structure design of the neighbour discovery overlay.

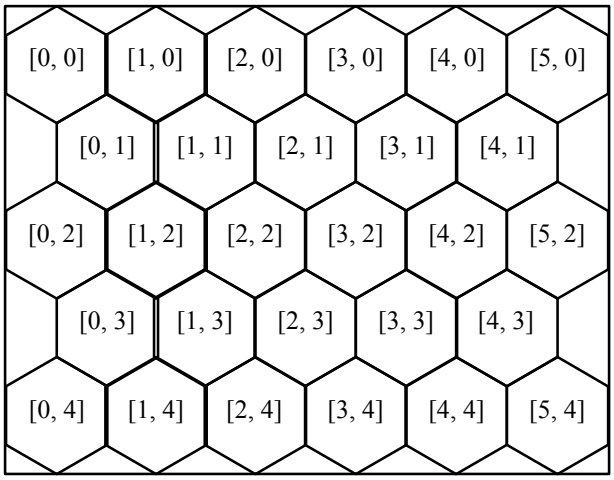


Figure 3-1 Virtual World Divided Into Hexagonal Cells

3.2.1 Hierarchical Topology

As mentioned, the virtual game world is divided into cells. Nodes inside same cell form a group and one of them is selected as the cell's master-node. Each master-node has direct connections (TCP/UDP) with master-nodes of adjacent cells and with all nodes in the cell. Figure 3-2 illustrates hierarchical structure of the Neighbour Discovery Overlay. Master-nodes are connected forming a virtual upper layer, which is the backbone of the Neighbour Discovery Overlay. Master-nodes of adjacent cells are neighbours on the backbone network. Member nodes of each cell form a group in the lower lever of the hierarchy.

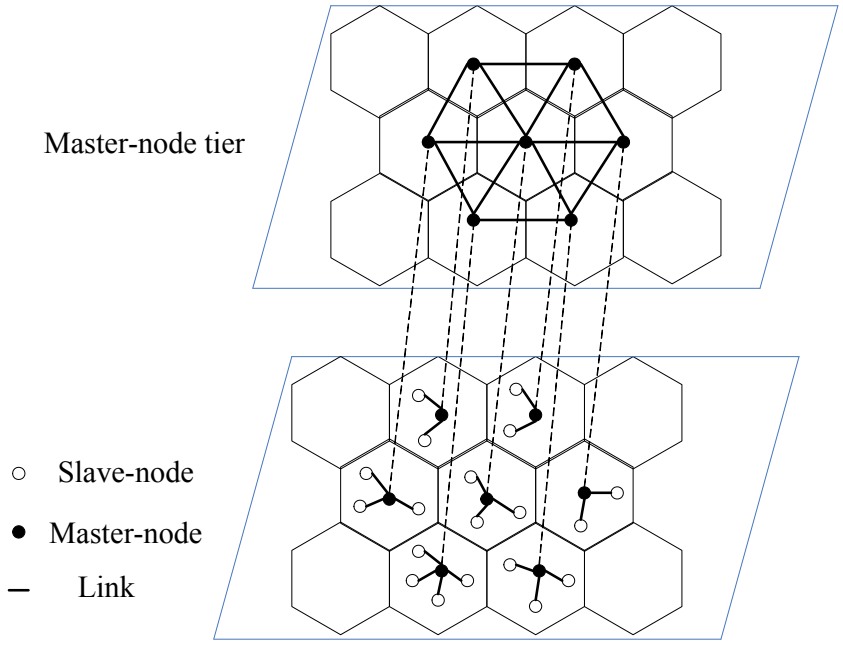


Figure 3-2 Hierarchical Structure

3.2.2 Slave-nodes and Master Nodes

3.2.2.1 Slave-nodes

A slave-node is a player node, acting as a client in its group. We usually use the term "slave-node" to present a player node when we want to distinguish this member

node from its master node. When joining the game or entering a new cell while wandering around in the virtual game world, a player node establishes a TCP (or UDP) connection with the new cell's master-node, which then becomes this joining player's current master-node. Through this direct communication channel, the node periodically reports its location, AOI size, and movement (direction and speed) to its master-node. Section 3.3 explains how player nodes find and register with their master-nodes.

In our interest management scheme, player nodes open and maintain TCP or UDP connections with nodes in its AOI to receive game updates directly from them. In order to achieve this, player nodes have to know all nodes in their AOI first. This is realized by notification from their master-nodes. As long as a cell member has registered its location, AOI size, and movement with its current master-node, the master-node will discover all other nodes in this member's AOI and notify this member through the direct connection between them. Section 2.2.3 and Section 3.2.4 explain how master-nodes find out nodes in area of their group members' interest.

3.2.2.2 Master-node

A master-node is a player node acting as a local server to all player nodes of a cell. The particular cell is called the master-node's local cell and all nodes in the cell are cell members (or group members of the cell). The master-node of a cell is selected from nodes in the cell except there are no other nodes in the cell. Which node would be chosen to be the cell's master-node depends on a number of factors, such as its storage capacity, processing power, bandwidth, and its AOI size. Master-node assignment procedure is presented in Section 3.3.1.

A master-node and its slave-nodes communicate through their TCP (or UDP) connection, which is built when the slave-nodes join its cell. If two master-nodes of adjacent cells need to communicate, a TCP (or UDP) connection is established and maintained between them. Prior the establishment of such a connection, one of the

master-nodes consults the Pastry overlay to get necessary information about the other master-node. The details of how to locate a master-node are presented in Section 3.3.

A master-node has three responsibilities. The first responsibility of a master-node is to gather information from all slave-nodes in its local cell and disseminate this information to master-nodes of adjacent cells if they are interested. The second responsibility is to obtain cell-information from master-nodes of adjacent cells and forward them to adjacent requesting cells. The last, but not least, is to discover and notify its slave-nodes about changes in their AOI.

3.2.3 Fundamentals of Master-Node Tier

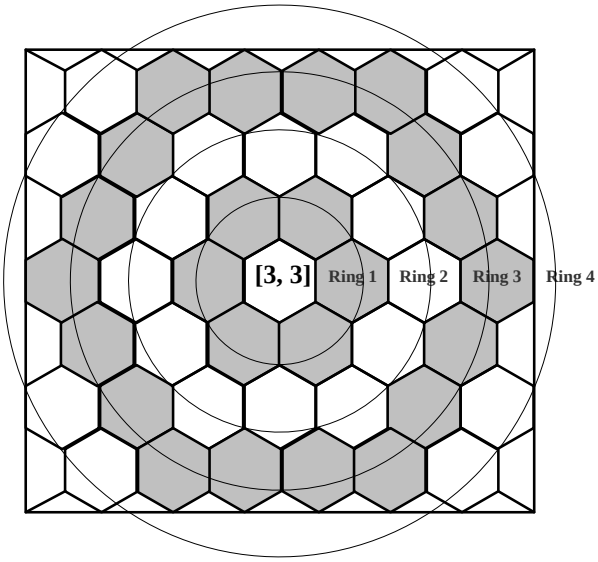


Figure 3-3 Message Spreading

As mentioned before, master-nodes of adjacent cells are connected to each other, forming the backbone overlay network. Since the backbone overlay covers all cells of the entire game world, messages can be routed between any cells in the map through communications between master-nodes. Figure 3-3 illustrates how a message originated in a cell spreads out all over the entire backbone. First, the master-node of the cell where the message originates forwards the message to its neighbours. After receiving the

message, those neighbours then forward the message to their neighbours and so on until it reaches the destination. The message is propagated across entire game world as signal radiated from the source. In Figure 3-3 the message generated in cell [3, 3] is first passed to cells on ring 1, and then passed to cells on ring 2, and then cells on ring 3 and 4….

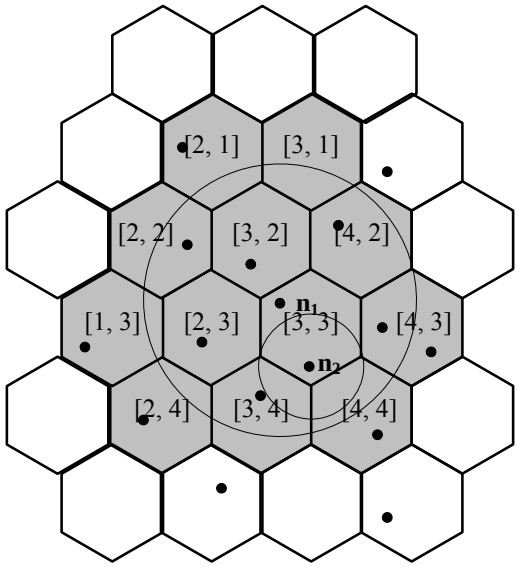


Figure 3-4 Cells Covered by all AOIs of Nodes in a Cell

Another important duty of a master-node is to discover the changes in its members' AOI and inform these members. Knowing only updates within its local cell is inadequate for a master-node to make neighbour discovery decision, because the area of its members' interest could cover several cells. The master-node, therefore, needs update information of all cells overlapped with AOIs of all nodes in its local cell. In Figure 3-4, areas of interest of nodes n_1 and n_2 in cell [3, 3] overlap with twelve cells. Therefore, the master-node of cell [3, 3] needs to find out information about nodes in these twelve cells. Fortunately, with the backbone overlay network a master node can get any cell's information (nodes in that cell) from its neighbour master-nodes. If every master-node reveals information of its own cell to its neighbours and forwards messages received to other neighbours, information of any cell would be made known to all master-nodes on the backbone network. As long as a master-node has information about all nodes in cells

overlapped by AOI of its members, it can decide and inform its members of entities in their area of interest.

3.2.4 General Operation of Neighbor Discovery

Since master-nodes need node-information of only cells overlapping with areas of its members' interests, information should not be sent to unrelated nodes. Therefore, in our design, messages are not propagated to all master-nodes in the system, but only to those that have registered for the messages. Messages are delivered to master-nodes on need basis, filtered by master nodes along their spreading way.

In order to receive other cells' information, a master-node registers with its neighbours by sending them a subscription list of cells whose node-information is expected from them. If any of cells on the received subscription list is not its own cell, the neighbour master-node, in turn, put it onto its own subscription list and register with its other neighbours.

More specifically, the *neighbour discovery overlay* operates as follows. First, every node periodically reports its position, AOI size, and movement to its master node. Then, according to the received information, each master-node calculates cells covered by area of their members' interest and makes a list of cells (excluding its own) whose node information it needs to know. If the master-node is also requested by its neighbour master-nodes for information about cells other than its own, it adds those cells to the list. If not empty, the list is split into sub-lists corresponding to each neighbour master-node. The resulting subscription lists are then sent to neighbour master-nodes respectively. Later on, when receiving any node information from neighbours or slave-nodes, the master-node forwards the message to neighbours who subscribed to and updates its slave nodes if the message results in changes (nodes entering/leaving) in area of their interest.

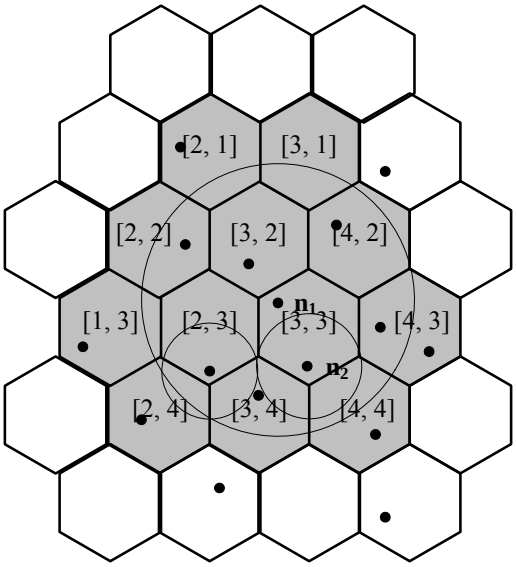


Figure 3-5 Example of neighbour discovery

We use an example to explain the neighbour discovery process. In Figure 3-5, nodes in cell [3, 3] cover 12 cells with their AOI, namely cell [2, 1], [3, 1], [2, 2], [3, 2], [4, 2], [1, 3], [2, 3], [3, 3], [4, 3], [2, 4], [3, 4], and [4, 4]. The master-node of cell [3, 3] registers with master-node of cell [2, 3] for information of cell [2, 3], [1, 3], and [2, 2]; registers with master-node of cell [3, 2] for information of cell [3, 2], [2, 1], and [3, 1]; registers with master-node of cell [4, 2] for information of cell [4, 2]; registers with master-node of cell [4, 3] for information of cell [4, 3]; registers with master-node [4, 4] for information of cell [4, 4]; registers with master-node of cell [3, 4] for information of cell [3, 4] and [2, 4]. With their AOIs, nodes in cell [2, 3] cover cell [2, 3], [2, 4], and [3, 4]. For nodes in its own cell, the master-node of cell [2, 3] needs to subscribe to only cell [2, 4] and [3, 4]. But since information of cell [1, 3] and [2, 2] is required by master-node of cell [3, 3], the master-node of cell [2, 3] has to put cell [1, 3] and [2, 2] onto its subscription list and register with master-nodes of cell [1, 3] and [2, 2] for information about nodes in their cells respectively. Latter on, when receiving messages about nodes in cell [1, 3] or [2, 2] from their master-node, master-node of cell [2, 3] forwards them to master-node of cell [3, 3]. In this way, entities in AOIs can be efficiently located.

3.3 Rendezvous Overlay

We designed another overlay, the rendezvous overlay, to assist the construction of the *Neighbour Discovery Overlay*. The rendezvous overlay of our system consists of all player nodes as its network nodes. It is implemented with a Pastry [3]. Each node in the rendezvous overlay has a unique identifier (*nodeId*) and each cell also has a unique identifier (*cellId*) as well. The topology of *Neighbour Discovery Overlay* is constructed with the assistance of the rendezvous overlay. Upon joining a game, a player node bootstraps into the rendezvous overlay network first, and then joins *Neighbour Discovery Overlay* to start playing game.

Information about master-nodes is stored in the rendezvous overlay and can be retrieved with *cellId* so that player nodes can find their corresponding master-nodes through the rendezvous overlay. Likewise, a master-node can find adjacent neighbour master-nodes through the same rendezvous overlay. The format of information stored in the rendezvous overlay is as (*key, value*) pair. In our case, the key is *cellId* and the value is information about master-node of the cell, such as identifier, IP address, and port number of the master-node. Since Pastry is reliable and fault tolerant, information of the master-node of a cell is always can be retrieved from the Pastry node whose *nodeId* is numerically closest to the *cellId* of the cell. For detailed information about how Pastry operates to keep the promise, the reader can refer to [3]. We call the node whose *nodeId* is numerically closest to the *cellId* of a cell the home node of the cell.

3.3.1 Master-Node Assignment

A node may consult the rendezvous overlay for information about master-node of a cell in the following situations. When joining the game, a node calculates with its geographical position in the game world to decide *cellId* of the cell it belongs to and then seeks information about the master-node of its accommodation cell via the rendezvous

overlay. Also, when moving from a cell to another, a node consults the rendezvous overlay for the master-node of the adjacent cell it is entering. Moreover, master-nodes also use the rendezvous overlay to obtain information about master-nodes of adjacent cells to open direct communication connection with them. In each case, the enquiry message routes in the rendezvous overlay and finally reaches the home node of the cell. The home node then sends back a message about master-node of the cell to the inquirer.

In case the cell being inquired does not have a master node yet, the home node records information of the inquirer and assigns it as the master-node of the cell. The inquirer could be a node that is entering the cell or a master-node of adjacent cell. In either case, the inquirer becomes the master-node of the cell. Obviously, a non-empty cell always has a master-node and a cell without a master-node must be empty.

An empty cell may be with or without a master-node. Connection between two master-nodes of adjacent cells is constructed on need base. Originally there is no connection between two neighbour master-nodes until one of them needs the other's assistance. The master-node in need consults the rendezvous overlay to get the other's information and then opens a direct connection between them. In case the adjacent cell is empty and without a master-node, the enquiring master-node immediately becomes the master-node of the adjacent cell. Now the empty cell has an alien master-node, which is also the master-node of one of its adjacent cells.

Figure 3-6 depicts example situations in which an empty cell is with or without a master-node. To simplify the problem, we assume there are only those five nodes in the virtual world. Let us take a look at master-nodes of empty cell [4, 2], [2, 3], [3, 3], [3, 4], and [4, 4]. Master-node of cell [2, 4] is also the master-node of cell [3, 4]. Cell [4, 4] has an alien master-node that is the master-node of cell [5, 4]. Cell [2, 3] also has an alien master-node that is either the master-node of cell [2, 2] or that of cell [2, 4]. Master-node of cell [4, 2] is either the master-node of cell [3, 2] or that of [4, 3], depending on which of them inquiring the rendezvous overlay for the master-node of the cell [4, 2] first. Since

none of neighbours want to get any message from master-node of empty cell [3, 3], cell [3, 3] does not have a master-node.

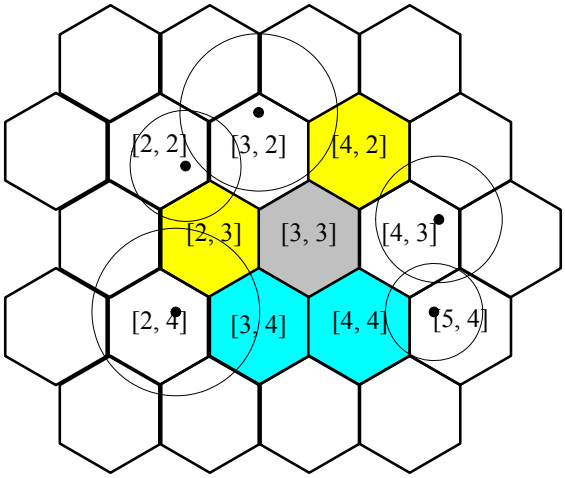


Figure 3-6 Empty Cells with/without a Master-node

As shown in Figure 3-6, master-node of cell [2, 4] is also the alien master-node of cell [3, 4]. Its alien master-node position is derived from its master-node position of cell [2, 4]. Therefore, we say its alien master-node position is a derivative of its master-node position of cell [2, 4]. The master-node positions owned by a node could form a tree, where the root is the master-node position from where the others are developed. Figure 3-7shows an example. Because of its master-node position of cell [3, 4], Node *n* becomes alien master-node of other eleven adjacent cells. Each of its alien master-node positions derives from its predecessor in the tree.

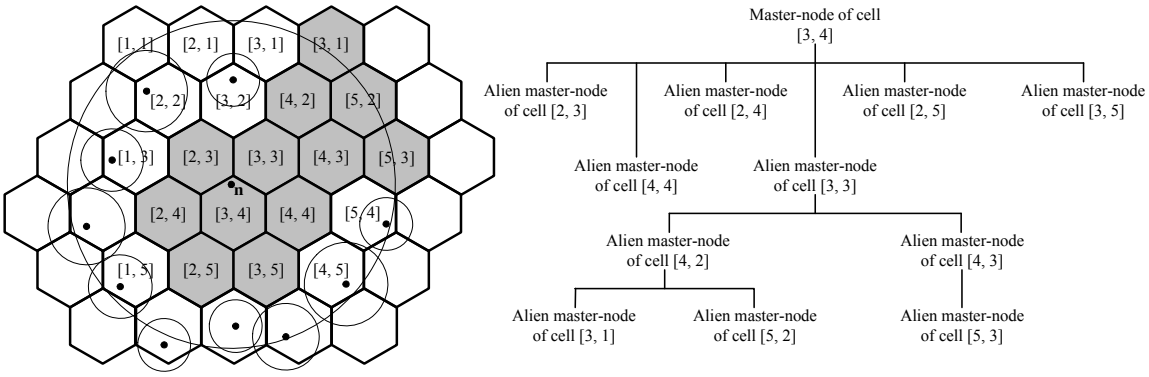
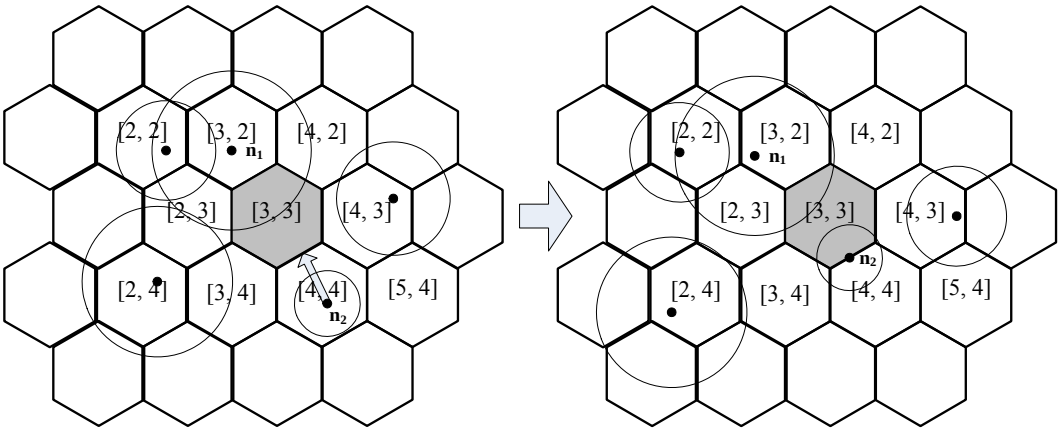


Figure 3-7 Master-node/Alien Master-node Tree

3.3.2 Alien Master-node Replacement

Being an alien master-node of adjacent empty cells adds only a little extra load on the master-node since those cells are empty without slave-nodes to look after. Those adjacent empty cells are linked together. The situation is similar to a master-node with a larger territory but same number of slaver-nodes.



Node n_1 is the alien master-node of cell [3, 3]. Node n_2 is moving towards cell [3, 3].

Node n_2 replaces node n_1 as the master-node of cell [3, 3] when entering it.

Figure 3-8 Alien Master-node Replacement

We always try to avoid putting too much load on alien master-nodes. Therefore, as soon as an empty cell with an alien master-node has a resident, the alien node resigns from the master-node position with the new comer as its successor. Figure 3-8 illustrates the replacement of an alien master node.

The details about how a node resigns from its master-node position are presented in Section 3.3.3.

3.3.3 Master-node Resignation

When leaving its cell for another one or leaving the game, a node resigns as master-node of the cell. A node resigns if it is also the alien master-node of a cell as soon as the cell has a resident. If there are any slave-nodes in the cell, the master-node selects one with more resources as its successor based on their storage capacity, processing power,

bandwidth, and AOI. If there is no slave-node in the cell, the master-node selects one from neighbour master-nodes that it has direct connection with as its successor. Then the master-node informs the home node of its resignation and the successor if there is one. The home node modifies the record accordingly and then sends back a confirmation.

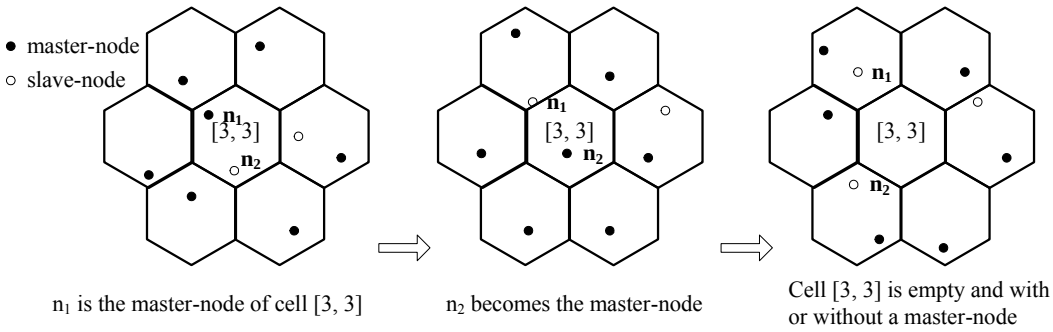


Figure 3-9 Master-node Resignation

After receiving the confirmation from home-node, the master-node informs the successor of its new role. Then after introducing the new master-node to its slave-nodes and neighbour master-nodes that it has direct connection with, the retiring master-node closes its connections with them. The slave-nodes in the cell will build connection with their new master-node. So do the master-nodes of adjacent cells when they need.

If the retiring master-node does not have a successor, it only needs to inform the home node of its resignation. The cell becomes empty and without a master-node. Figure 3-9 shows two examples of master-node resignation.

As stated in section 3.3.1, being a master-node (or alien master-node) of one cell may cause the node to be an alien master-node of another adjacent cells. A node should also be exempt from its alien master-node positions originated from the master-node position that the node is just removed from. As shown in Figure 3-7, its position as the master-node of cell [3, 3] causes node n to be the master-node of a tree of cells [4, 2], [3, 1], [5, 2], [4, 3], and [2, 3] as well. Resignation as master-node of cell [3, 3] means node n resigns from all master-node positions of the branch led by master-node of cell [3, 3].

3.3.4 Fault Tolerance

As stated before, the rendezvous overlay is constructed on top of Pastry that is reliable and fault tolerant. Therefore, our major concern about fault tolerance is the *neighbour discovery overlay*.

Node crash or network failure could happen to any nodes in the system. When failure happens, the dead node in *Neighbour discovery Overlay* causes the part of network structure involving it to crash as well. How to repair the damage depends on the role of the falled node, a slave-node or a master-node.

When a slave-node fails, its master-node will soon be aware of that because the direct connection between them fails as well. Therefore, the master node can surely and safely erase the slave-node from the system as if the slave-node quits the game normally.

When a master-node fails, its direct connections with its slave-nodes and with master-nodes of adjacent cells go down as well. Being disconnected unexpectedly and abnormally, those nodes will consult the rendezvous overlay for a new master-node of the cell. When the home-node receives the request, it will probe the master-node. If the master-node is unreachable, the home node will pick the first requesting one to fill in the position.

3.4 Master-node Load Balance

A master-node is crucial to its slave-nodes because they depend on it to get updates of entities in their AOI. A master-node is also important to master-nodes of neighbouring cells because they depend on it to get updates of node-information of cells on their subscription list. The updates will be delayed if the master-node is overloaded. Our design focuses on the network bandwidth overhead, because it is the major load of the master-node.

3.4.1 Causes of Master-node Overloading

Based on the discussion before, we know that a master-node needs to receive every position update sent out from nodes in cells on its subscription list. I.e., a master-node needs to track each node in cells on its subscription list. The number of updates received by a master-node depends on the number of nodes it has to watch. The more nodes it watches, the more updates it will receive. When the update frequency is very high, the performance of the master-node could degrade significantly.

There are two unavoidable situations that could result in master-node overloading. Each of them raises the number of nodes some master-nodes need to watch. When a master-node has a member with a big area of interest or when one of its neighbour master-nodes has a member with a big area of interest (and thus has to send it a long subscription list), the number of cells on its subscription list increases, resulting in the number of nodes it watches increases. Also, increasing in node density increases number of nodes master-nodes need to watch. Players tend to gather at hotspots where interesting events occur. This interesting phenomenon is called crowding or flocking. When crowding happens, the node densities of the cells near hotspots increase. The master-nodes whose subscription includes those crowded cells may become overloaded.

To deal with the aforementioned problem, our design offloads some of the workload from the master-nodes to its slaves and also reduces the amount of position updates among players using strategies detailed in the following sections.

3.4.2 Subscribing Cells instead of AOI

In order to notify members of their AOI contents, a master-node needs to watch every movement of each node in cells overlapping with its members' AOIs. If any node in those cells makes a movement, the master-node receives a position update. At the meantime, its members also update the master-node changes of their interest area caused

either by their movement or by change of AOI size. If any movement causes any node moves into or out of an AOI of its members, the master-node will detect it and inform the member.

If the master-node watches cells instead of AOIs, its network traffic will decrease significantly. In this case, master-nodes inform their slave-nodes about nodes in cells overlapping with their AOIs instead of those in their AOIs. More specifically, player nodes update their master-nodes on cells overlapping with their AOIs, and then master-nodes inform their slave-nodes nodes in their subscribed cells. Player nodes subscribe themselves to nodes in cells overlapping with their AOIs for position updates to decide which of them are in their AOIs.

Upon entering a new cell, a player node registers to the cell's master-node with a list of cells (which overlap with its AOI) as its subscription. The master-node sends back all nodes that are in those cells as response. The player node then subscribes to those nodes for position updates. When the player's AOI expands or shrinks in the game, the player's subscription changes because the set of cells overlap with his/her AOI change. When the player moves in the cell, its AOI moves along with him/her, The movement may causes the AOI overlapping with a different set of cells. Whenever the player's subscription changes, the player updates its master-nodes with his/her current subscription. On the other hand, the master-node detects and informs the slave-node when any player node entering cells on its subscription list.

Instead of receiving updates of each movement of nodes in cells overlapping with AOIs of its member nodes, a master-node now receives updates only when the movement brings a node into cells on its subscription list or causes its member nodes' AOI shifting to overlap with set of different cells. The network traffic of master-nodes thus decreases significantly.

There is a trade-off between master load and player load though. The network traffic of player nodes increases since they now receive position updates from nodes in cells overlapping with their AOI. However, updating on each movement of a node is not necessary. Because it usually takes steps for *node A* to cross the edge of area of *node B's* interest (depending on their movement speed, direction, and distance from the edge), the useful position updates from *node A* to *node B* are either the one of *node A* stepping into (or out of) *node B's* AOI or the one after which the movement of *node B* will include *node A* into its AOI. Other movements of *node A* are considered as irrelevant to *node B.* Updating *node B* on such movements of *node A* is unnecessary. Therefore, when *node A* is beyond certain distance from the edge of *node B's* AOI, we could safely eliminate updates of *node A's* movements there to lower network traffic. The updates received by *node B* drops because *node A* now only updates *node B* when it is close enough to the edge of *node B's* AOI.

3.4.3 Extended AOI

Another way to reduce network traffic of master-nodes is to lower position update frequency. Without any master-node balance mechanism, a player node sends position update for each of its movements to its master-node. From there, the message is then forwarded inside a tree of registered master-nodes, which in turn inform their slave-nodes of the movement only if the moving node crossed the edge of their AOIs. Since each movement of nodes inside the cells on subscription list of a master-node generates a position update for the master-node, the communication overhead may become unaffordable for the master-node. To deal with this problem, we could let player nodes send updates periodically, after several movements instead of each movement, to lower the number of position updates received by master-nodes.

It is possible that a node moves into others' AOI between its two successive updates. Therefore, an AOI has to be extended to enclose all nodes with the ability of

moving into the AOI before its next update. The extended AOI has its real AOI inside and also some extra space where player nodes outside the real AOI may exist. Assume t is the period of time between two successive updates and v is the fastest moving speed of a node. In the Figure 3-10, d_1 and d_2 are the maximum distance that could be covered by *node A* and *node B* between their two successive updates respectively. Obviously $d_1 = d_2 = vt$. Therefore, radius of expanded AOI is the radius of the original AOI plus $2vt$.

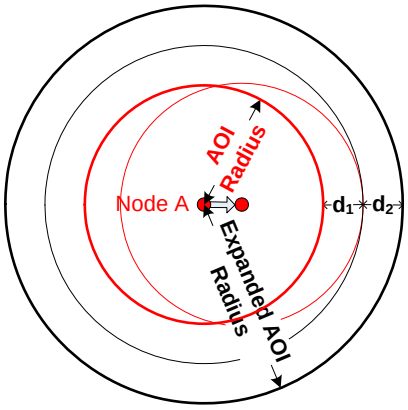


Figure 3-10 Expanded AOI

The master-node can tell whether a node is inside the extended AOI or not, but it cannot tell whether a node inside the extended AOI is inside the real AOI. To identify nodes inside its real AOI, the player node can connect with nodes in its extended AOI to get their position updates directly and more frequently. Instead of a position update per movement, these updates can be set at different frequency to reduce communication overhead. Just as described in the previous section, nodes near the edge of the real AOI update more frequently while nodes far away from the edge update less frequently.

3.4.4 Master-node Helper

On occasion such as crowding, the node densities of cells go high. The master-nodes of those crowded cells can be overwhelmed by excessive communication and computation load imposed on by the large number of player nodes gathered in its cell. To

deal with such problem, we could select some nodes as master-node helpers to assist their master-nodes.

3.4.4.1 Hierarchical Approach

Master-node-helpers are dynamically selected from slave-nodes by their master-nodes according to their capacity and AOI size. Nodes with more capacity and bigger AOI are more likely to be chosen. When a master-node's workload exceeds its workload threshold, the master-node assigns a slave-node as its helper and gives part of its load to the helper. A master node may have several helpers and a master-node helper may also have its own helpers as well.

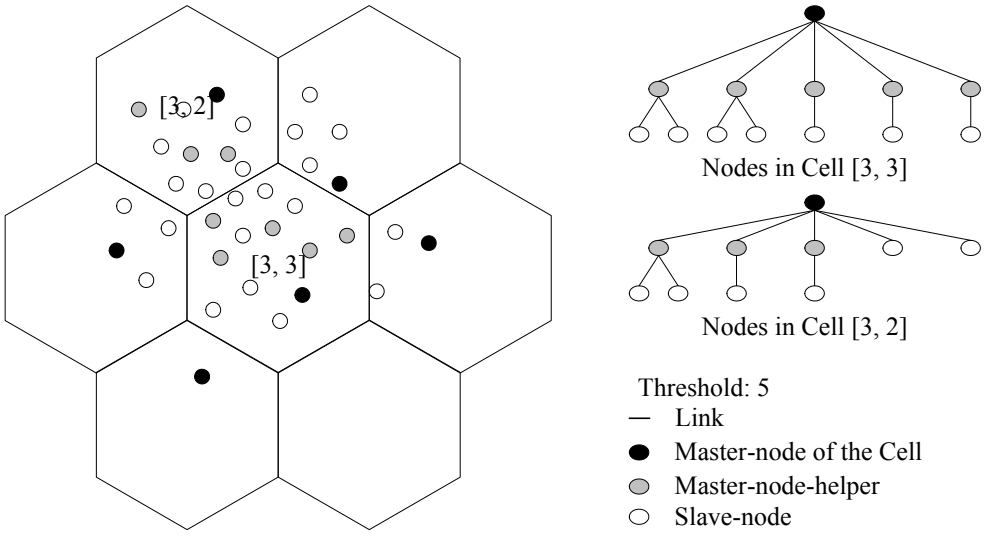


Figure 3-11 Tree Structure of Master-node, Master-node Helper, and Slave-node

A master-node helper serves its slave-nodes just like an ordinary master-node. Its slave-nodes may not know if their master-node is a real master-node or a master-node helper. To its master-node, a master-node helper is similar to a regular slave-node except that it periodically reports not only its own location and movement but also that of its slaves along with a subscription list of cells covered by its group. A cell's master-node

aggregates all node information from its helpers and slave-nodes and exchanges the cell-information with neighbours on the backbone of the *neighbour discovery network*.

Workload threshold of a master-node or master-node helper depends on its current available capacity such as processing power, bandwidth, etc. How to make the threshold out of the factors is another subject that is out of scope of this paper. In our design, the workload threshold is the max number of slave-nodes a master-node or master-node-helper has, where the number of slave-nodes is configurable. Figure 3-11 shows the tree structure of master nodes, master-node-helpers, and slave nodes.

When the workload of a cell's master node decreases below a certain level along with a decline of node density in a cell, the master node takes the load from its helpers.

3.4.4.2 Resignation of Master-node Helper

When leaving the game or leaving its current cell for another cell, a master-node helper resigns from its master-node-helper position. The procedure of master-node-helper resignation is simpler to that of the master-node resignation.

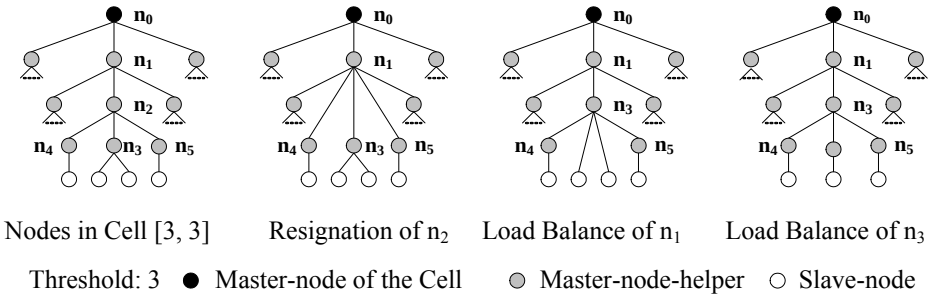


Figure 3-12 Resignation of Master-node Helper

The retiring master-node helper informs its slave-nodes its current master-node as their new master-node. It also informs its master-node its resignation. After receiving confirmation from its master node and its slaves, the master-node helper disconnects with them. With the information of the new master-node, its slave-nodes build connection with

their new master-node. The master-node erases the helper from the cell and reintroduces the other helpers into the tree structure.

Figure 3-12 gives an example of the resignation of master-node helper. When node n_2 leaves cell [3, 3], it informs its slave-node n_3, n_4, and n_5 about its resignation and introduce the new master-node n_1. Then node n_3, n_4, and n_5 try to connect to n_1. Node n_1 accepts the connections from node n_3, n_4, and n_5. Node n_3, n_4, and n_5 become slave-nodes directly under node n_1. After load adjustment of node n_1 and n_3, nodes original led by node n_2 are rearranged into the cell's load balance tree.

In case any unexpected events happen at the steps of a master-node helper's resignation, nodes can always consult the rendezvous overlay for master-node of the cell. The procedure is similar to recovery of master-node helper, which is explained in next section.

3.4.4.3 Automatic Recovery

The recovery procedure of master-node helper failure is a combination of that of master-node failure and that of slave-node failure. A master-node helper is a slave-node to its master-node and also a master-node to its slave-nodes. Both its master-node and its slave-nodes can tell if the helper fails or not from aliveness of the direct connection between them. After the helper goes down, the helper including its group member is erased by its master-node from its list. On the other hand, the slave-nodes directly under the helper resort the rendezvous overlay for master-node of their local cell. Noticing the crash node supplied by those consulting slave-nodes is not current master of the cell, the home-node provides them information about the cell's real master-node. With this information, the slave-nodes rejoin in the cell as slave-nodes directly under the master-node of the cell. Figure 3-13 gives an example of the recovery process of the master-node helper failure.

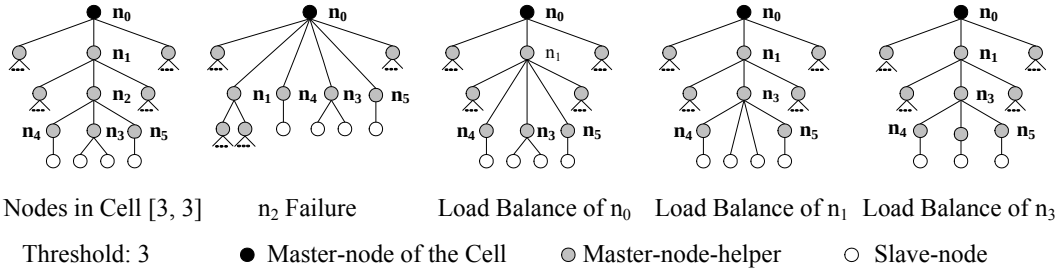


Figure 3-13 Recovery from Master-node Helper Failure

Chapter 4

Protocol and Algorithms

In this chapter, we first describe the registration and routing scheme applied on the backbone network of *Neighbour Discovery Overlay* for master-nodes to obtain information of nodes in other cells. Then, we explain the communication schemes applied on DHT to build and maintain the dynamic *Neighbour Discovery Overlay*. Finally, we summarize the key features of our system.

4.1 Registration & Routing Scheme

As mentioned in the previous chapter, master-nodes are connected, forming a backbone network of *neighbour discovery overlay*. Every master-node acts as a publisher, subscriber, and router. It publishes information about nodes in its cell, subscribes for node information of desired cells, and also routes received messages from other master-nodes. In this section, we describe the subscribing, publishing, and routing scheme used in the system.

4.1.1 Design Considerations

Issues such as scalability, cost, and efficiency have to be taken into consideration for designing the routing protocol on top of the backbone network. Obviously broadcasting information about nodes in each cell is not a good choice because it introduces extensive unnecessary network traffic and wastes the resource of nodes with irrelevant message processing. After all, messages do not need to be propagated to every master-node in the system since a master-node needs only information about nodes in the cells covered by AOIs of its cell members.

In the backbone network there exist several different paths to connect two particular master-nodes. An efficient way of routing a message from a master-node to another is through the shortest path between them. To find the shortest path, we use a rule to direct the registration and routing so that when master-node m_0 of cell c_0 subscribes information about nodes in cell c_n whose master-node is m_n, the registration goes from m_0 to m_n through master-nodes on a shortest path between m_0 and m_n and the cell-messages route back from m_n to m_0 along the same shortest path in reversed order.

Master-nodes on the backbone network are selected from ordinary player nodes, which move consistently from cell to cell in the game world and may also quit the game at any time. When a master-node leaves its cell, it resigns the master-node role and another node fills in when needed. Therefore, for each master-node to keep track of the whole topology of such a dynamic changing network would results in excessive message exchange overhead, which may be not affordable by players' machines, mostly PCs. Therefore, our design only requires each master-node to use limited topology information, namely master-nodes of its adjacent cells, and simple computation to efficiently subscribe and route information.

4.1.2 Algorithm Description

Area of a player's interest usually covers (or overlaps with) several cells. As shown in Figure 3-4, player node n_2 in cell [3, 3] covers a set of cells {[3, 3], [3, 4], [4, 4]}. To serve a player, the master-node must obtain information about nodes in cells overlapping with that player's AOI. Likewise, to serve all nodes in its cell, a master-node must subscribe to node-information of all cells covered by AOI of any node in its cell.

To reduce overhead in tracking master-nodes of cells on its subscription list, a master-node does not register directly to or obtain node-information directly from non-neighbour master-nodes. Instead, the master-node registers its subscription list only to its neighbour master-nodes, expecting to obtain all needed node-information from them.

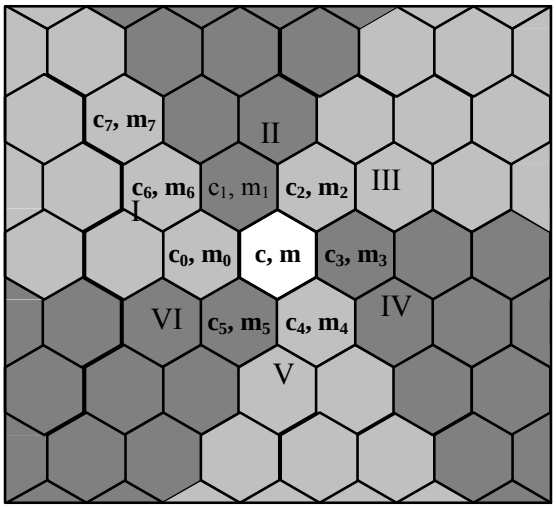


Figure 4-1 Registration of Subscription

As shown in Figure 4-1, if current master-node m needs node-information of any cell in area I, it registers to neighbour master-node m_0. Likewise, m registers to m_1 for node-information of any cell in area II, registers to m_2 for node-information of any cell in area III, registers to m_3 for node-information of any cell in area IV, registers to m_4 for node-information of any cell in area V, and registers to m_5 for node-information of any cell in area VI.

For instance, m registers to m_0 for node-information of cell c_7. m_0 then registers to m_6 and m_6 registers to m_7. As the master-node of cell c_7, m_7 periodically issues update information about nodes in its cell to m_6 (and other registered neighbours). And the message is forwarded to m via m_6 and m_0.

Since the current master-node has to take care of both the player nodes in its cell and the master-nodes of adjacent cells, its subscription list contains two parts: the cells covered by AOI of player nodes in its cell and the cells registered by its neighbour master-nodes. Except the current cell, whose nodes are known to the current master-node, other cells on the subscription list must be divided into sub-subscription list and registered to neighbour master-nodes respectively to obtain node-information of those cells. When receiving a message from a neighbour master-node, the current master-node forwards the message to neighbours who subscribed to it.

The following definitions (Table 4-1) are used in the algorithm and discussed later in this section.

Symbol	**Description**
c	current cell that current master-node *m* is responsible for
m	current master-node of current cell *c*
c_i	any cell
nc	the set of adjacent cells of *c*
n_i	any player node
pn	node set of *c,* i.e. all nodes in current cell *c*
m_i	any neighbour of *m*, $0 \leq i < 6$
nm	The set of neighbours of *m*
cells_covered_by(n_i)	the set of cells covered by n_i with its AOI
cells_covered_withAOI	the set of cells covered by nodes in *c* with their AOI
subscription_from(m_i)	a set of cells registered with *m* by m_i
neighbour_subscriptions	the set of cells registered with *m* by all neighbours *nm*
subscription	subscription list of *m*
registration_with(m_i)	A set of cells that *m* registers with m_i

Table 4-1 Definitions used in registration and routing algorithm

Figure 4-2 demonstrates some definitions listed in Table 4-1 and symbols used in the algorithm.

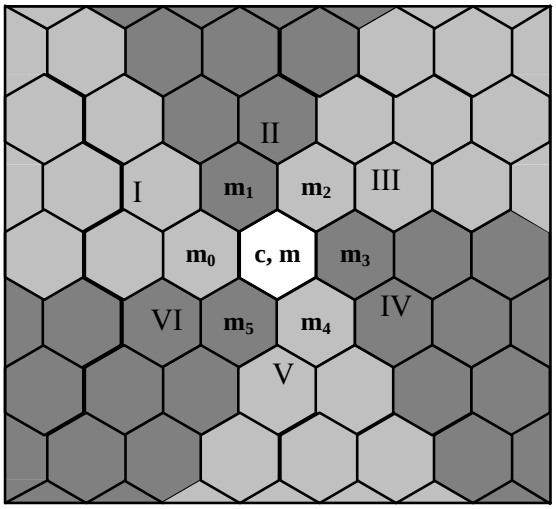


Figure 4-2 Registration and Routing

The algorithm in Figure 4-3 is used to compute $registration\_with(m_i)$.

cells_covered_withAOI = ∪cells_covered_by(n_i) where $n_i \in pn$

neighbour_subscriptions = ∪subscription_from(m_i) where $m_i \in nm$

subscription = cells_covered_withAOI ∪neighbour_subscriptions

for each cell c_i in subscription

if c_i is in area of I

add c_i to registeration_with(m_0)

if c_i is in area of II

add c_i to registeration_with(m_1)

if c_i is in area of III

add c_i to registeration_with(m_2)

if c_i is in area of IV

add c_i to registeration_with(m_3)

```
if c_i is in area of V
    add c_i to registeration_with(m_4)
if c_i is in area of VI
    add c_i to registeration_with(m_5)
```

Figure 4-3 Algorithm to Compute Registration and Sub-Registration Lists

The algorithm in Figure 4-4 describes the forwarding process for the current master-node m when it receives a message about nodes in cell c_i.

```
/* when current master-node m receives message about nodes in cell c_i, this algorithm decides
which neighbors m should forwards the message to */
for each neighbor master-node m_i in nm
  if c_i ∈ subscription_from(m_i)
    forward the message to m_i
```

Figure 4-4 Routing Algorithm

We use an example to explain the algorithm in Figure 4-3 and Figure 4-4.

Figure 4-5 depicts an instance of the virtual world, where cell [3, 3] is the current cell. We use c_{ij} to denote cell with cellId $[i, j]$, m_{ij} to denote its master-node.

As shown in Figure 4-5, nodes in current cell c_{33} cover cells of $\{c_{31}, c_{22}, c_{32}, c_{23}, c_{33}, c_{43}, c_{24}\}$ with their AOI. Neighbour master-node m_{23} registers $\{c_{33}, c_{43}, c_{34}\}$ to current master-node m_{33}, and neighbour m_{43} registers $\{c_{33}\}$. Then m_{33} combines these lists and make its own subscription list as $\{c_{31}, c_{22}, c_{32}, c_{23}, c_{33}, c_{43}, c_{24}, c_{34}\}$. These data are recorded in Table 4-2, whose last column in last row is the subscription list of current master-node m_{33}. The algorithm then divides the subscription

list into sub list as shown in Table 4-3. For example, m_{33} registers $\{c_{22}, c_{31}\}$ to m_{22} and it will receives node information about c_{22} and c_{31} from m_{22}.

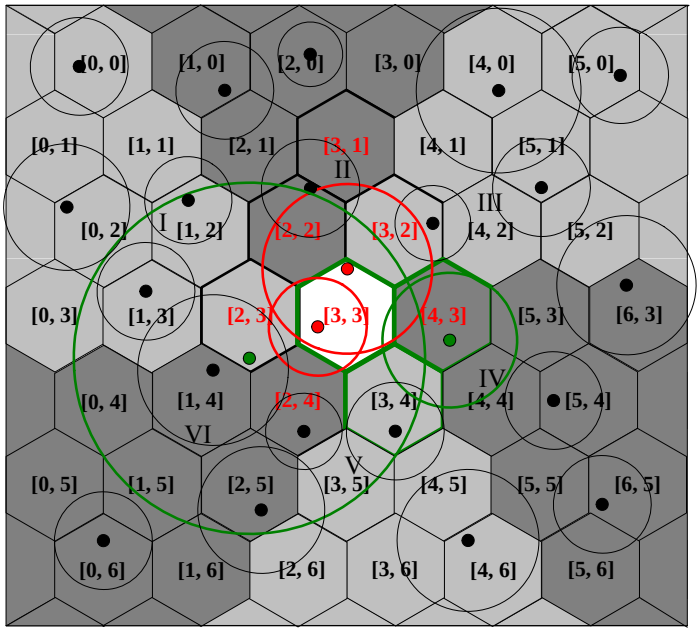


Figure 4-5 An Instance of Virtual World

master-node m_{ij}	***subscription_from(*** m_{ij} ***)***
m_{33}	c_{31}, c_{22}, c_{32}, c_{23}, c_{33}, c_{43}, c_{24}
m_{23}	c_{33}, c_{43}, c_{34}
m_{43}	c_{33}
subscription	c_{31}, c_{22}, c_{32}, c_{23}, c_{33}, c_{43}, c_{24}, c_{34}

Table 4-2 Subscription of m_{33}

neighbour m_{ij}	***registration_with(*** m_{ij} ***)***
m_{23}	c_{23}
m_{22}	c_{22}, c_{31}
m_{32}	c_{32}
m_{43}	c_{43}
m_{34}	c_{34}
m_{24}	c_{24}

Table 4-3 Registration of m_{33}

Table 4-4 shows master-nodes that the current master-node m_{33} should forward received message of nodes in cell c_{ij} to.

message of cell c_{ij}	***forwarding to master-node*** m_{ij}
c_{31}	m_{33}
c_{22}	m_{33}
c_{32}	m_{33}
c_{23}	m_{33}
c_{33}	m_{33}, m_{23}, m_{43}
c_{43}	m_{33}, m_{23}
c_{24}	m_{33}
c_{34}	m_{23}

Table 4-4 Routing of m_{33}

4.2 Communication Schemes of the Rendezvous Overlay

As stated in section 3.3, the rendezvous overlay layer helps build and maintain *Neighbour Discovery Overlay*. Information of master-nodes of *Neighbour Discovery Overlay* is stored in the rendezvous overlay as (*key, value*) pairs where the *key* is *cellId* and the *value* is information of master-node of the cell with identifier *cellId*. Master-node information includes master-node's *nodeId*, IP address, and listening port. We use Pastry [3] to implement the rendezvous overlay.

In addition to keeping master-node information of its cell, a home-node of the rendezvous overlay also takes part in assignment of master-node of its cell. Messages such as master-node enquiry, resignation, and miss in action (MIA) may intrigue home-nodes to change master-node of their cell. In this section, we describe communication schemes and procedures used to perform these functions.

Table 4-5 describes variables and functions used in the pseudo code later in this section.

Variables	Description
cellId	ID of a cell
nodeId	ID of a node
IP	IP address of a node
port	Listening port of a node
n	current node
c	current cell
homeNode	home-node of cell *c*
record	*HomeNode's* record of master-node of cell *c,* including *cellId, nodeId, IP,* & *port*
cm	Current message
rm	response message

Table 4-5 Variable Description

Table 4-6 describes functions used in the pseudo code later in this section.

Basic functions	Description	Return
GetRecord(cellId)	For home-node to get its record of master-node of cell with *cellId*	Record, including *cellId, nodeId*, *IP*, & *port*
SetRecord(cellId, record)	For home-node to set its record of master-node of cell with *cellId*	None
Route(key, message)	Route *message* in Pastry to home-node of cell whose ID is *key*	None
Send(nodeId, message)	Send *message* to node whose ID is *nodeId*	
GetSuccessor(cellId)	For master-node of cell *cellId* to select a successor from its slave-node or neighbour master-nodes	selected node
IsMaster(cellId)	If current node *n* is master-node of cell *cellId*	Boolean
GetSlaves(cellId)	Get the set of *nodeId*s of slave-nodes of master-node of *cellId*	A set of *nodeId*s
GetNeighbours(cellId)	Get the set of *nodeId*s of neighbours of master-node of *cellId*	A set of *nodeId*s
GetChiden(cellId)	Get *cellIds* of master-node of cells that directly derived from master-node of *cellId*	A set of *nodeId*s
Disconnect(cellId)	As master-node of *cellId,* disconnect with slave-nodes and master-nodes of adjacent cells of *cellId*	None
IsAlive(nodeId)	If node of *nodeId* is alive	boolean

Table 4-6 Function Description

Table 4-7 lists type and fields of messages used in the communication.

Message	Message fields
Master-node Enquiry	1. key: *cellId of the interest cell* 2. type: *ENQUIRY* 3. nodeId: *enquirer's nodeId* 4. IP: *enquirer's IP address* 5. port: *enquirer's listening port*
Master-node Resignation	1. key: *cellId of current cell* 2. type: *RESIGNATION* 3. mNodeId: *master-node's nodeId* 4. sNodeId: *successor's nodeId* 5. IP: *successor's IP address* 6. port: *successor's listening port*
Master-node MIA	1. key: *cellId of current cell* 2. type: *MIA* 3. mNodeId: *master-node's* 4. rNodeId: *reporter's* 5. IP: *reporter's IP address* 6. port: *reporter's listening port*
Master-node Information	1. Type: *INFORMATION* 2. CellId: *ID of a cell* 3. NodeId: *cellId's master-node* 4. IP: *master-node's IP address* 5. port: *master-node's listening port*
Master-node Replacement	1. Type: *REPLACEMENT* 2. CellId: *ID of a cell* 3. oNodeId: *old master-node's* 4. nNodeId: *new master-node's* 5. IP: *IP address of new master-node* 6. port: *listening port of new master-node*

Table 4-7 Message Type

4.2.1 Master-node Enquiry

Master-node Enquiry message is for player nodes to consult the rendezvous overlay for master-node information of specified cells. The enquirer supplies the *cellId* of interest cell along with its own *nodeId*, IP address and listening port with the enquiry message, which reach the home-node of interest cell through routing in Pastry. The home-node then sends back a *Master-node Information* message according to its record of master-node of the cell with *cellId* as identifier. If the cell does not have a master-node, the enquirer is assigned as master-node of the cell. Figure 4-6 shows the communication and procedures used in the function.

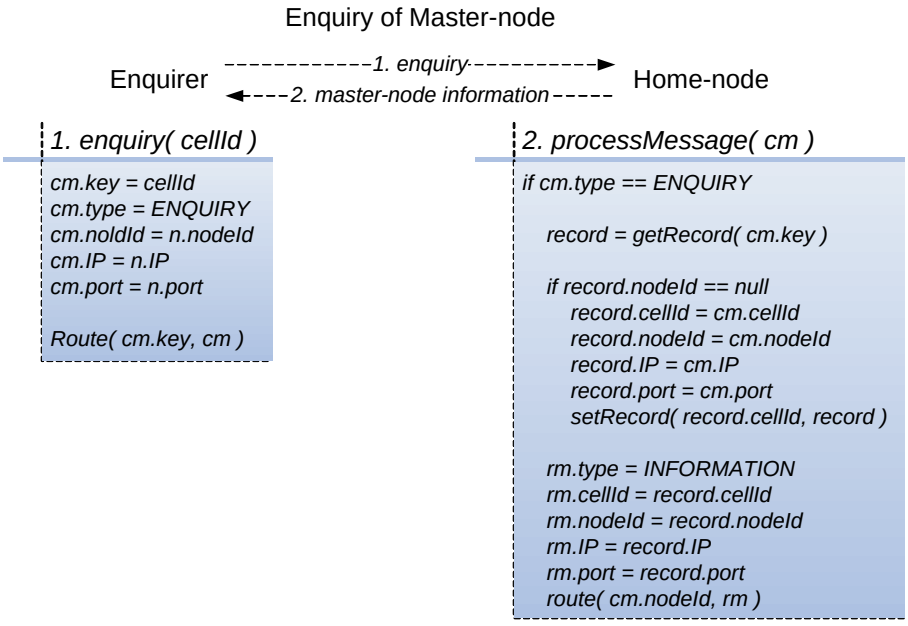


Figure 4-6 Master-node Enquiry

The enquirer uses the supplied information in the *master-node information* message to connect to the master-node of the cell. If the newly connected is a resident of an empty cell, the cell's alien master-node resigns and selects the new comer as its successor.

4.2.2 Master-node Resignation

Master-node Resignation message and *Master-node Replacement* message are the two messages used for master-node resignation. The resigning master-node of a cell sends a *Master-node Resignation* message to the cell's home-node through routing in Pastry. The home-node sends back a *Master-node Replacement* message as confirmation. The communication and procedures are shown in Figure 4-7.

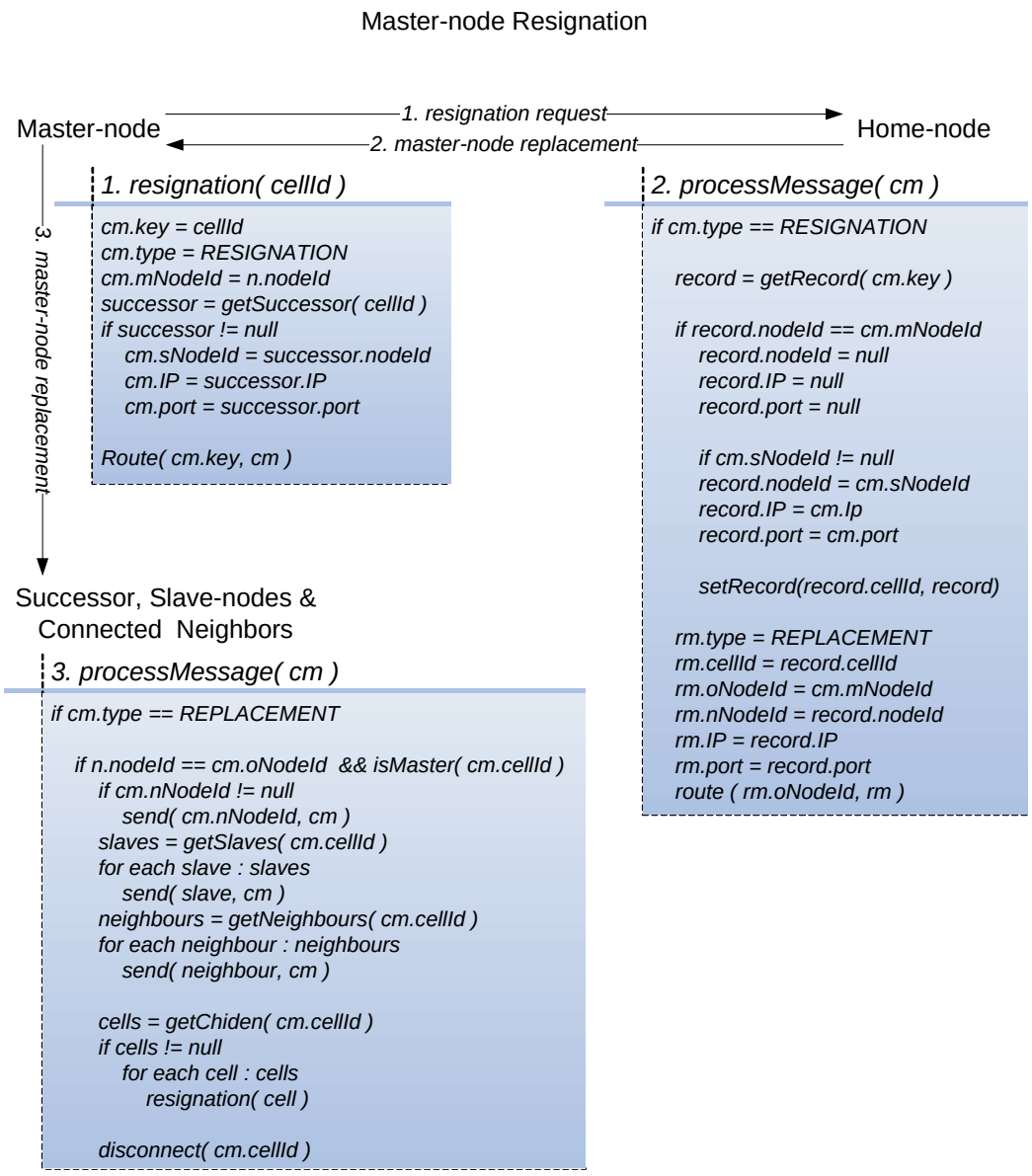


Figure 4-7 Master-node Resignation

4.2.3 Master-node MIA

Master-node missing in action is reported to home-node with *Master-node MIA* message by nodes connected with the crashed one. The home-node repairs the damage by filling in the master-node position with another node.

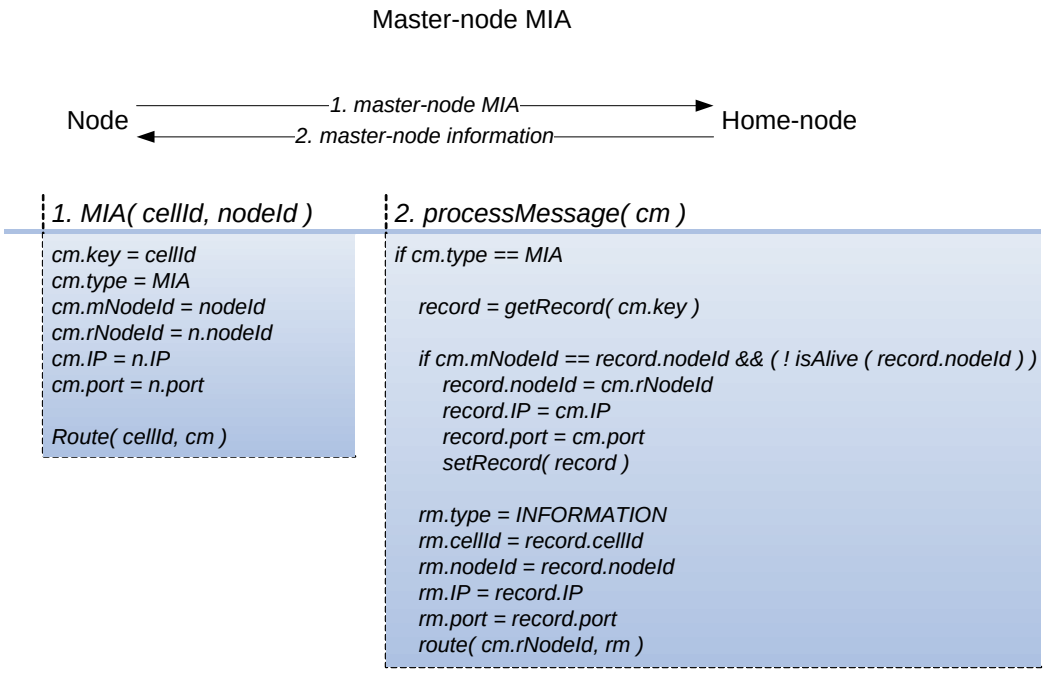


Figure 4-8 Master-node MIA

Figure 4-8 shows the communication and functions of master-node MIA.

4.3 System Analysis

The scalability and efficiency of our system relies on two assumptions: (1) Every game entity has a finite area of interest, which includes area around the entity. (2) An action at one place affects only part of the game world, mainly its neighbourhood. Our design exploits the locality feature in both assumptions. As stated before, a node concerns only events occurred in the area of its interest. The closer a node is to the location where an event occurs, the more likely its AOI covers that location. So, when master-node of cell *A* registers with master-node of its adjacent cell *B* for information of nodes in cell *C* that is closer to *B*, master-node of cell *B*, most likely, has already got that information because its slave-node's AOI also overlaps with cell *C*.

As the result of our registration and routing scheme applied on the master-node tier of *Neighbour Discovery Overlay,* each cell has a message distribution tree whose root is

its master-node. Figure 4-9 shows the distribution tree of cell C_0 used in dissemination of updates on nodes in cell C_0. The message is distributed along its registration path in reversed order. Since a parent node is always one hop closer to cell C_0 than its child nodes, for the majority of the time when the child nodes need node info of cell C_0, the parent node itself also need that info to serve its slave-nodes. The parent node forwards received node info of cell C_0 only to its child nodes that registered for cell C_0. Hence, node info of cell C_0 is disseminated to nodes through nodes that need the message, avoiding flooding the message to nodes that don't need the message. Because there is no circle in the tree and each node has only one parent that forwards the message to it, there are no duplications of messages.

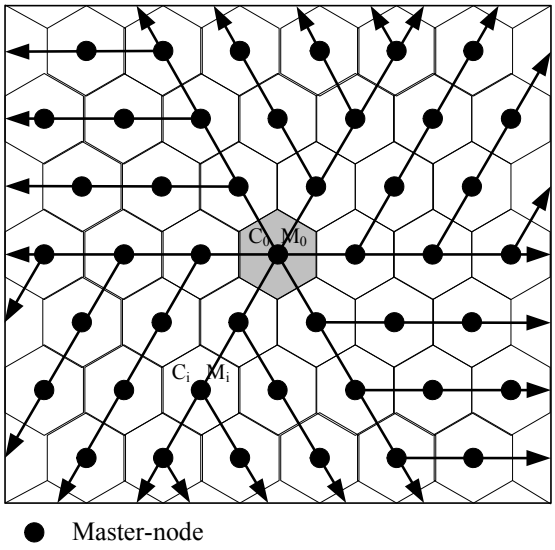


Figure 4-9 Distribution Tree of a Cell

Our system is comprised of two structured P2P networks, *Neighbour Discovery Overlay* and a rendezvous overlay. Employing a globally consistent protocol, structured P2P networks guarantee, that any node can efficiently retrieve data distributed in the network. Therefore, our rendezvous overlay, which is a structure P2P overlay, can guarantee the safe retrieval of master-node information of each cell. A slave-node can always connect with its current master-node and master-nodes can always connect with current master-node of adjacent cells at any time. The global connectivity of the topology is ensured in the sense that the whole topology is stored in the rendezvous overlay and

can be retrieved by nodes at any time when in need of connecting to any master-nodes. If a group of nodes is “isolated” from the rest of the network, it is not because there is no way for them to find each other and get connected but because both partitions don’t need any information from the other.

The artificial boundary that divides the virtual game world into cells should neither block game entities’ view nor mark the area of game entities’ interest like that in some existing systems do. The idea that game entities have same, or same size of, or fixed area of interest is also made up to simplify problem. Area of different game entity’s interest should have its own self-defined size and may change over time. It moves along with its entity, stretching across cells from time to time. Our system has no restriction on game entity’s AOI. Players can have different AOI sizes and their AOI can expand or shrink at any time during game. Our system also guarantees that each game entity has continuous view of the virtual world instead of discrete. This is because the area inside which nodes are observed by a master-node is the set of cells overlapping with AOIs of its slave-nodes that is always bigger than and contains all AOIs of the slave-nodes.

Chapter 5

System Implementation

In this chapter, we present the implementation details of the MOPAR+ prototype, and the game simulation and visualization tool built on top of it. The design presented in Chapter 3 is influenced by object-oriented technology. MOPAR+ is implemented in Java and has undergone some basic testing. The prototype system runs on Windows, Macintosh, Linux, and Sun. As mentioned, our system consists of two P2P overlays, the rendezvous overlay and the *neighbor discovery overlay*. We choose Free Pastry as rendezvous overlay implementation and include its code package in our system. The *Neighbor Discovery Overlay* is completed with success implementation of master-node, slave-node, home-node, and the communication and routing protocol applied on them.

5.1 System Design

5.1.1 Architecture

MOPAR+ is a middleware that runs on TCP/UDP and provides services to its application software running on top of it, enabling multiple processes running on one or more machines to interact. It consists of two P2P networks: the rendezvous overlay and

Neighbor Discovery Overlay. *Neighbor Discovery Overlay* is partially built on top of the rendezvous overlay as shown in Figure 5-1. The rendezvous overlay and *Neighbor Discovery Overlay* are loosely coupled. The Free Pastry used in our implementation can be replaced with other DHT products.

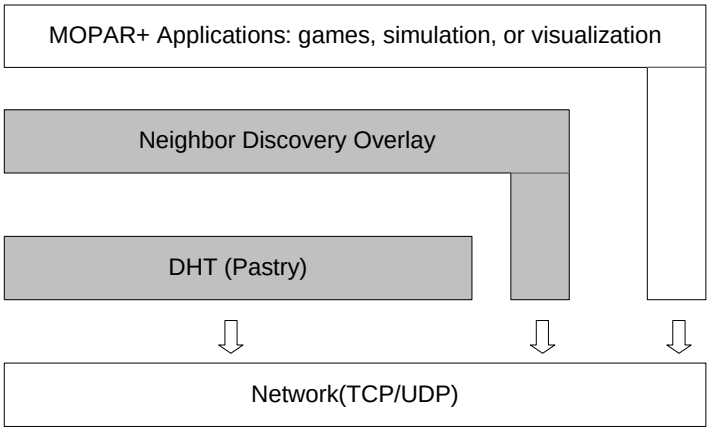


Figure 5-1 MOPAR+ Architecture

Table 5-1 describes the relationships between each layer shown in Figure 5-1.

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.
Neighbor Discovery Overlay	Neighbor Discovery Overlay is the core of MOPAR+ interest management, including implementation of the master, slave, and home nodes, and communication and routing scheme. This layer sits on both DHT and the beneath network and uses services provided by both of them.
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 Structure Layer Description

The abstraction interface between MOPAR+ and game application layer is the API provided by MOPAR+ for its applications to interact with it. The abstract interface implemented in MOPAR+ is described in Table 5-2.

API	Description
join (double[] pos, double aoiRadius)	This method joins a player node into the networked game, preparing the node for gaming. It must be called before the other methods in the API. The *pos* and *aoiRadius* are needed for MOPAR+ to provide information about nodes in the AOI.
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. *Pos* is needed for updating the node of nodes in its AOI.
getAoiNeighbors()	Gets the list of players that are currently within the node's AOI.
changeAoiRadius(double r)	The method is called when the node's AOI radius changes. The new radius r is needed for getting nodes in the node's current AOI.

Table 5-2 API of MOPAR+

5.1.2 Major Components

Table 5-3 describes the major components of the MOPAR+.

Package	Description
participant	This package includes MOPAR+ API and classes that implement the API. To use MOPAR+, the application creates an MoarParticipant object and then calls methods in API on the object.
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. It is the core package of our system since the backbone of Neighbor Discovery Overlay comprises of all master-nodes in the system and the communication and routing scheme applied on the backbone is implemented in the behaviour of master-nodes. In addition to inter-master communication, master-nodes also communicate with their slave-nodes and half of communication scheme between slave and master is completed in this package .
messaging	This package defines all types of message used in MOPAR+.
util	This package maintains utilities that are common used by all components in the system, such as MapHelper in which methods are relevant to cells in the virtual world.
properties	This package contains the configuration files for MOPAR+, configuring its working environment such as hexagon side length, virtual world dimension, work mode as subscribing expanded AOI etc.
game	This package implements a simple simulation game for demonstration purposes.
gui	This package implements a visualization tool for a game simulation.

Table 5-3 Major components of MOPAR+

5.2 Implementation Details

5.2.1 Core Classes

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

abstract master.Master

Implements the communication and routing protocol applied on master-nodes tier. Provides neighbour discovery service to slaves.

master.AoiMonitorMaster *master.Master*

Implements the abstract functions define in Master class when the system runs with subscribing AOI mode.

master.CellMonitorMaster *master.Master*

Implements the abstract functions define in Master class when the system runs with subscribing cell mode.

master.ExpandedAOIMonitorMaster *master.Master*

Implements the abstract functions define in Master class when the system runs with subscribing expanded AOI mode.

participant.MoparParticipant *IMoparParticipant*

Instantiates an object of participant.Participant according to work mode configured in configuration file. Implements functions defined in ImoparParticipant by calling the same function on Participant object.

abstract participant.Participant *ImpoarParticipant*

Implements functions defined in ImoparParticipant interface.

abstrac participant.ParticipantSubscribingCells *Participant*

Implements the abstract functions defined in Participant when the system runs with subscribing cells mode.

participant.ParticipantPeriodicPositionUpdate *ParticipantSubscribingCells*

Implements the abstract functions defined in ParticipantSubscribingCells class when the frequency of inter-slave position updates is configured to base on two nodes' distance.

participant.ParticipantSubscribingAOI *Participant*

Implements the abstract functions defined in Participant class when the system runs with subscribing AOI mode.

participant.ParticipantSubscribingExpandedAOI *participant*

Implements the abstract functions defined in Participant class when the system runs with subscribing expanded AOI mode.

homenode.HomeNodeFactory

Instantiates home nodes, and transfers home-node role between Pastry nodes, if another node becomes more qualified for the home-node of a cell than the current one of the cell.

homenode.HomeNode

Provides the services, such as master-node assignment, resignation, and enquiry, to support building and maintaining Neighbor Discovery Overlay

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 hexagon-related functions, such as get ID of neighbour hexagons, get a list of cell overlapping with an AOI etc..

game.MoparGame *participant.MoparParticipant*

Implements a simple simulation game for demonstration purposes.

game.MoparNeiworkSimulator

Simulates the TCP/UDP network communications.

GUI.gamePanel *javax.swing.JPanel*

Display the simulation game, as well as some properties of players.

GUI.ListPanel *javax.swing.JscrollPane*

Lists all player nodes and their roles in the simulation game. Provides functions, such as select a node as current node, change the AOI radius of a node etc.

GUI.PlayerConnectionPanel *javax.swing.Jpanel*

Draws a graph of current node, nodes connected to current node, and the links between them.

GUI.gameViz *javax.swing.JFrame*

GUI of the MOPAR visualization tool.

5.2.2 Collaboration Diagrams

This section describes the flow of object interactions in the MOPAR+ through the assistance of UML sequence diagrams. We choose three major services provided by MOPAR+: join, move, and leave procedures.

Join 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 with an alien master-node so that the joining node will become the master-node of the cell.

Moving Procedure

Figure 5-3 describes the operations performed in the MOPAR+ infrastructure layer when a game player makes a movement. We assume that the node is moving into an adjacent cell and that it is a master-node in the current cell and will be a slave node in the entering cell.

Leaving Procedure

Figure 5-4 illustrates class collaboration under the scenario of a game player leaving the game. We assume that the node is the only node in a cell. Therefore, it is the master-node of the cell.

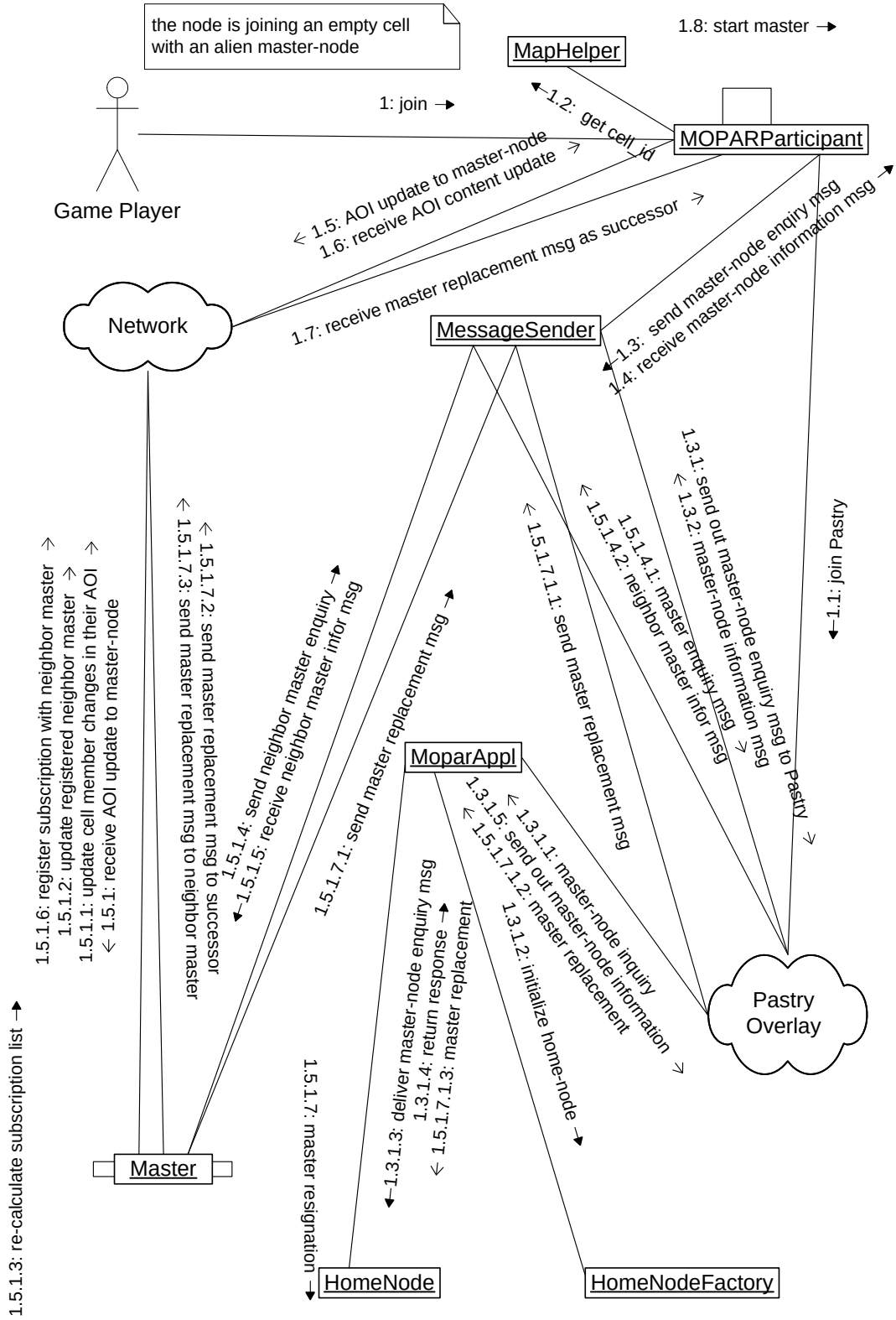


Figure 5-2 Collaboration diagram for joining procedure

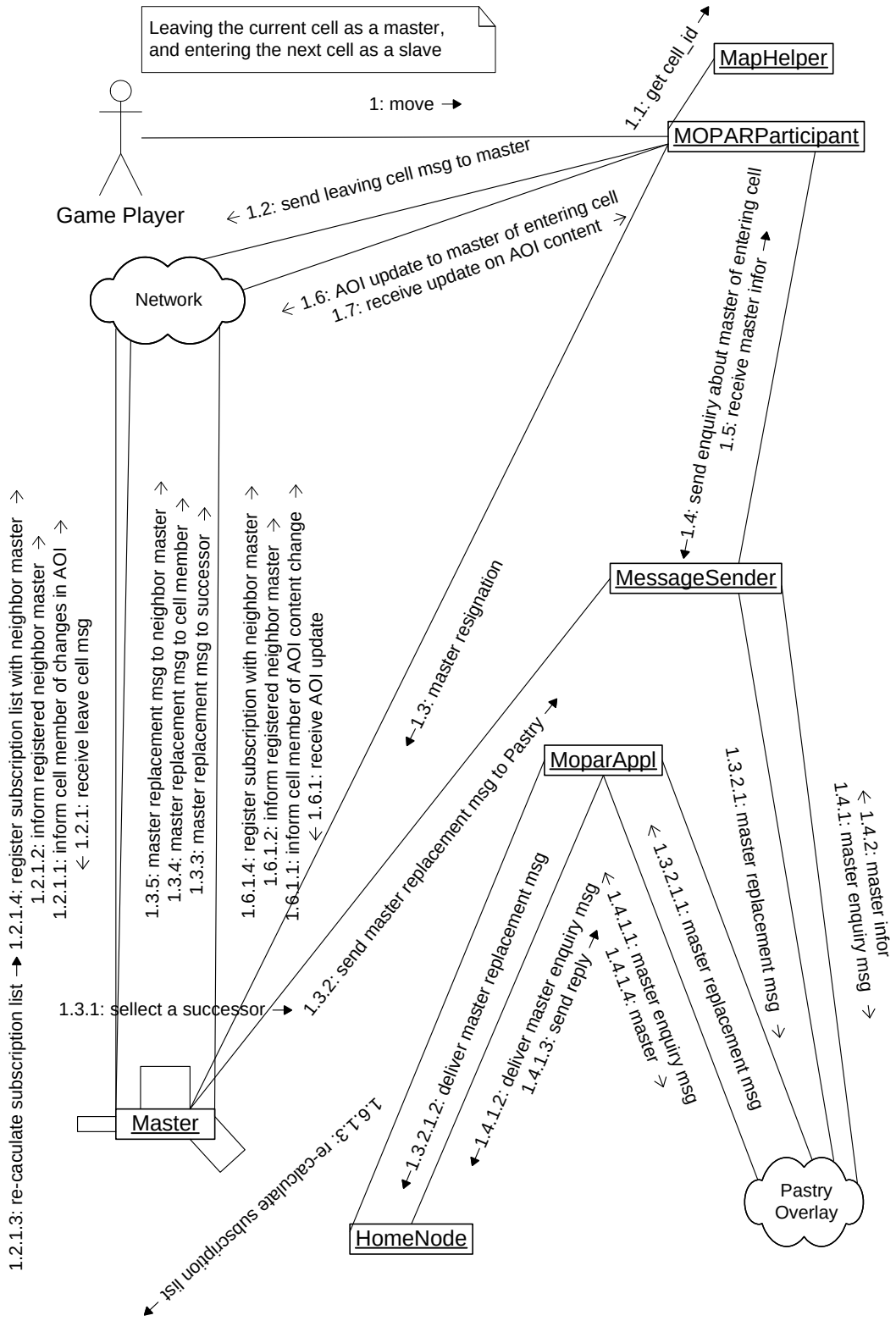


Figure 5-3 Collaboration diagram for moving procedure

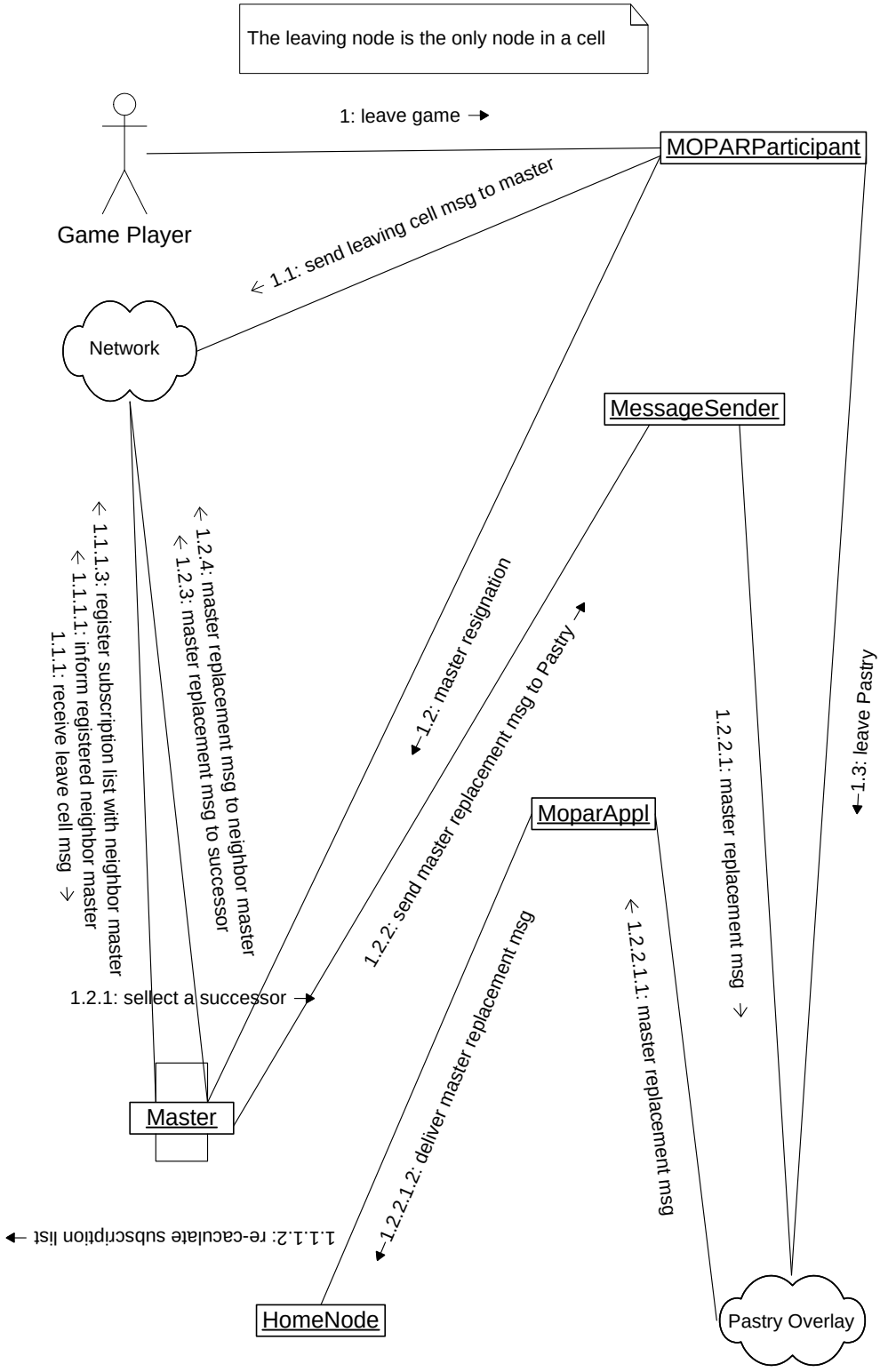


Figure 5-4 Collaboration diagram for leaving procedure

5.3 Visualization Tool

In order to demonstrate the effect of our interest management scheme, we have also implemented a simple game simulation and visualization tool. In the game, each player node has an avatar in the game and all avatars in the game wander around in the virtual game world. The GUI of visualization tool has three panels. The whole virtual world with all nodes endlessly moving in it is displayed in a game panel. Another panel, called list panel, lists all nodes and their status in the game. The connection panel displays a graph of the current node and its all connections to other nodes.

The user can select a node as the current node by mouse click on the node in the game panel or mouse double click on the node in the list panel. The current node is displayed in green to be distinguished from other nodes, which are in black. The dots that represent master-nodes are double sized compared to dots that represent non-master nodes. Dot size dynamically changes with the node switching back and forth between master-node role and non-master node role. The home-nodes are also represented by double sized dots but in color of blue while master-nodes are in color of black.

The major functionalities the visualization tool provides are listed below.

- Display all nodes on current node's AOI neighbor list.
- Display the topology of the *Neighbor Discovery Overlay*
- Display the dynamic structure of the Neighbor *Discovery Overlay*
- Change the size of the player's AOI

Figure 5-5 shows the game panel of our GUI. The red circle around current node marks the edge of its area of interest (AOI). When the current node discovers any node moving into its AOI, it sends a message to the node, requiring it to display itself in red. When the node moves out of its AOI, the current node sends another message to the node, requiring the node to display itself back in default black color. So the dots

displayed in red are nodes that are currently on current node's list of neighbors in AOI, i.e. all nodes that the current node detects and believes in its AOI. We can see that all dots inside the red circle are red and all red dots are inside the red circle. This justifies our claim that our interest management indeed enables all nodes in the system to correctly and efficiently find all nodes in their area of interest.

At any time, each cell has at most one master-node, the large dot in the cell. Change of dot size in the game panel indicates that the node is shifting from a master-node to a non-master node if the dot shrinks or from a non-master node to a master-node if the dot enlarges. It also reflects that the hierarchic structure (or topology) of *Neighbor Discovery Overlay* is undergoing some changes.

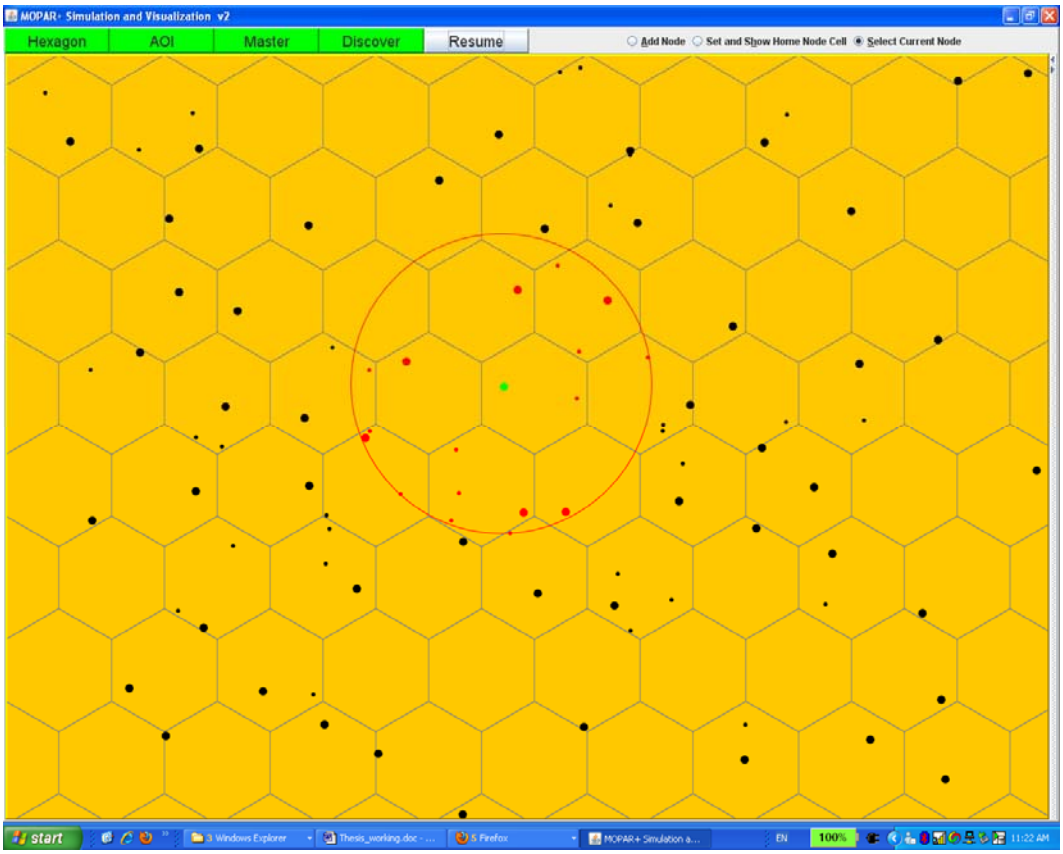


Figure 5-5 Game Panel of GUI

The upper right panel shown in Figure 5-6 is the list panel which lists all nodes in the game. The user can change AOI size of any nodes in the game from a dialog pane

popped up when mouse right clicks on the node. The user can also select a node as the current node by mouse double left clicks on the node.

Beneath the list panel is the connection panel which displays a graph of connections the current node has. There are three layers in the graph. *nodeId* of current node's master-node and *cellId* of the cell it is in charge of are shown in the middle. If the current node is a master-node, *nodeIds* of master-nodes it connects with and *cellIds* of the corresponding cells those connected master-nodes are in charge of are shown on the top layer. *nodeIds* of slave-nodes that the current nodes has (if the current node is a master node) are shown on the bottom layer. The graph shows part of structure of underneath *Neighbour Discovery Overlay*. The user can explore current topology of the *Neighbour Discovery Overlay* by pausing the game and then examining graphic connections between different current nodes.

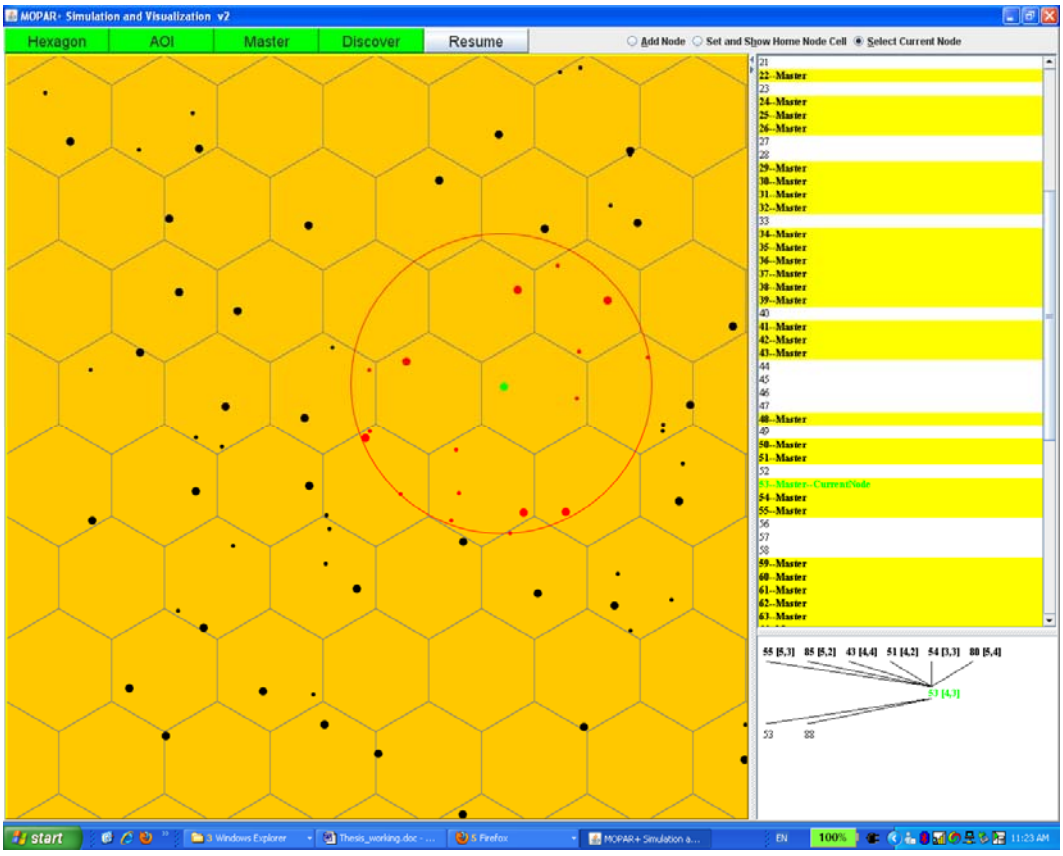


Figure 5-6 List Panel and Connection Panel

Chapter 6

Experiments

In this chapter, we present the experimental results we have obtained with a game simulator built on the prototype implementation of MOPAR+. There are various variables, such as the size of virtual world, the number of players, the size of cell, the node density, and the AOI radius and the moving speed of the avatar etc., in the system. Some of them are interrelated and may affect system performance. We first show and analyze the experimental results on scalability of our system and then the effect of node density, cell size, and AOI radius on the communication load of our system. In every instance, the results prove our hypothesis that our system has the scalability properties we claim.

6.1 Experimental Setup

All experiments are performed on a ThinkPad T60p with 2.0 GHz Intel(R) CPU and 2 GB of main memory. The machines run Microsoft Windows XP Professional. The MOPAR+ prototype and our game simulator are written in Java. All of the simulator components including the Free Pastry, run in a single Java VM. Free Pastry and the game simulator use their own network emulation environment implemented within their

software respectively. We concentrate on the networking aspect of the results because the computing load on each node is negligible compared with the bandwidth requirement. The maximum simulation size is constrained by the memory available in the testing computer. We conduct our experiment with a practical limit of about 4000 virtual nodes.

Our simulated game has a two-dimensional virtual game environment. Each player controls one avatar. The area of a player's interest is a circle centered at the position of its avatar and moves along with the entity. Each avatar moves in same direction with same speed for 100 steps. When the duration expires, its moving direction and speed is reset by randomly selecting two numbers from 0 to 1 as its moving distance in x and y coordinate in each step.

In our experiments, we configure each avatar to make a movement of one step every 100 milliseconds and measure 200 seconds of simulated game play. All measurements are taken when the simulated game is at a stable state, i.e. when no players join or leave the game.

Since the number of messages sent out by all nodes in the system equals the number of messages received by all nodes, we evaluate our system performance by the number of messages received. The results are average for each data point.

As member node of both Pastry and *Neighbour Discovery Overlay,* a player node receives messages from both P2P networks: messages from Pastry overlay and messages from *Neighbour Discovery Overlay*. The messages received as node of *Neighbor Discovery Overlay* can also be divided into two parts: 'to master' messages and 'to slave' messages. 'to master' messages are addressed to master-nodes, include AOI updates from slave-nodes to their master-nodes and cell-info updates between master-nodes. 'to slave' messages are AOI neighbor updates from master-nodes to their slave-nodes. Every node receives 'to slave' messages, but only master-nodes also receive 'to master' messages. In subscribing cell and subscribing expanded AOI work mode, a node also receives position

updates from nodes in cells overlapping with its AOI and nodes in its expanded AOI respectively. Therefore, the messages received by a player contain 4 parts: Pastry message, 'to slave' message, 'to master' message, and position updates.

6.2 Impact of Population Growth

This experiment is performed under expanded AOI mode with one position update every six movements. Side length of each hexagon is set to 200 distance units. Node density of each hexagon cell is fixed at 10. Each player's AOI has a uniform radius of 200 distance units.

nodes/cells	Neighbour Discovery Overlay		Position updates	Pastry	Total	Received by
	to master	to slave				
1000/100	3.19	0.95	39.46	0.28	43.88	node
	26.91	0.95	39.46	0.28	67.6	Master-node
1210/121	3.17	0.95	39.66	0.28	44.06	node
	27.28	0.95	39.66	0.28	68.17	Master-node
1440/144	3.2	0.99	40.78	0.29	45.26	node
	27.95	0.99	40.78	0.29	70.01	Master-node
1690/169	3.19	0.97	40.59	0.28	45.03	node
	28.04	0.97	40.59	0.28	69.88	Master-node
1960/196	3.2	0.98	41.11	0.28	45.57	node
	28.22	0.98	41.11	0.28	70.59	Master-node
2250/225	3.19	0.99	41.42	0.28	45.88	node
	28.5	0.99	41.42	0.28	71.19	Master-node
2560/256	3.2	0.99	41.07	0.28	45.54	node
	28.56	0.99	41.07	0.28	70.9	Master-node
2890/289	3.2	0.98	41.5	0.28	45.96	node
	28.79	0.98	41.5	0.28	71.55	Master-node
3240/324	3.19	0.99	41.58	0.27	46.03	node
	28.85	0.99	41.58	0.27	71.69	Master-node
3610/361	3.2	0.99	41.44	0.28	45.91	node
	29.04	0.99	41.44	0.28	71.75	Master-node
4000/400	3.2	0.99	41.37	0.28	45.84	node
	29.12	0.99	41.37	0.28	71.76	Master-node

Table 6-1 message breakdown, population growth

The purpose of this experiment is to evaluate the scalability of our system. We expect system performance for different population to be similar as long as the average node density is kept constant so that increases in players can be handled by expanding virtual world. The experiment results as shown in Figure 6-1 and Figure 6-2 justify our hypothesis that our system has the scalability properties we designed it for. We can thus

safely conclude that as long as the average cell density is kept constant, increasing in player population can be handled.

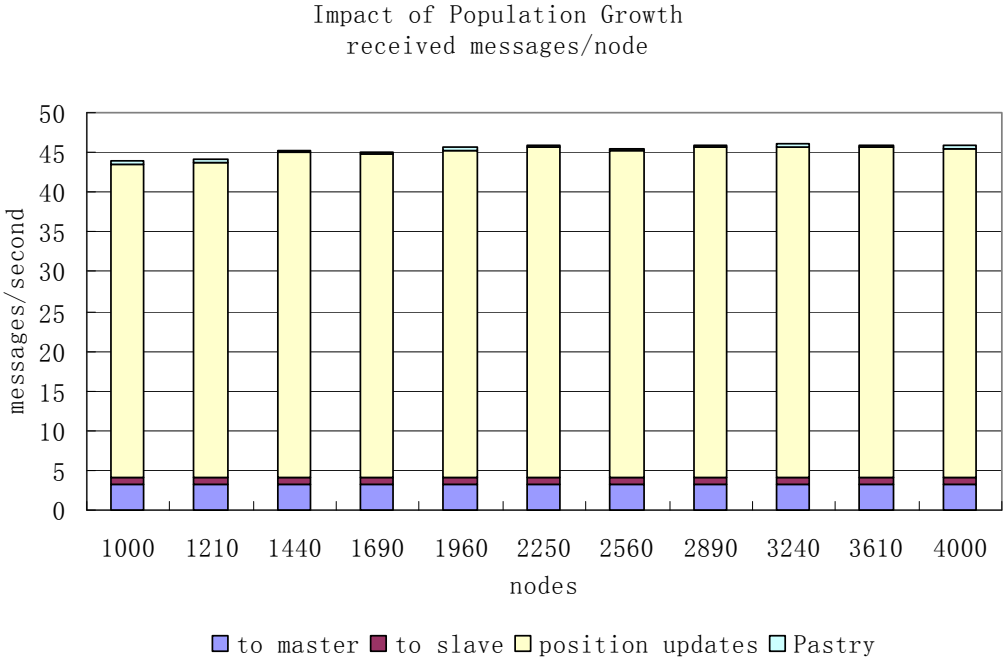


Figure 6-1 impact of population growth, received messages/node

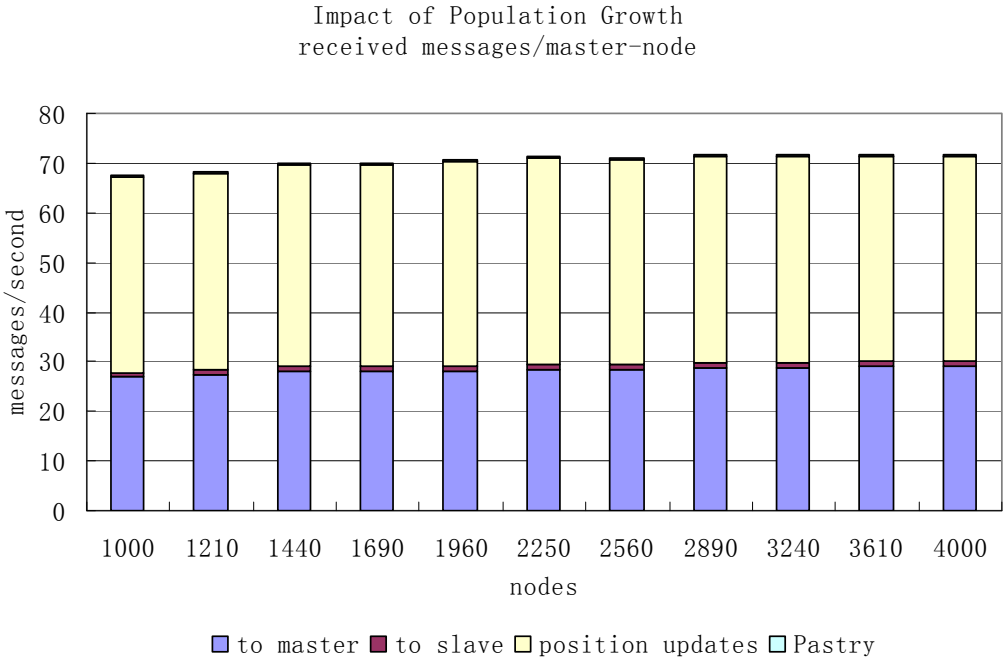


Figure 6-2 impact of population growth, received messages/master-node

Table 6-1 shows the average number of messages received per second by a player node and a master-node. Figure 6-1 and Figure 6-2 are the corresponding charts.

6.3 Impact of Population Density

We have also evaluated the effect of population density by re-running the experiment with 100 cells and 100 to 1500 players, increasing the average player density from 1 to 15 per cell. The experiment runs under expanded AOI mode with one position update every six movements. Side length of each hexagon is set to 200 distance units. Each player's AOI has a uniform radius of 200 distance units.

Nodes /cell	Neighbour Discovery Overlay		Position updates	Pastry	Total	Received by
	to master	to slave				
1	5.42	0.14	3.44	0.85	9.85	node
	7.63	0.14	3.44	0.85	12.06	Master-node
2	5.41	0.25	7.85	0.49	14	node
	11.45	0.25	7.85	0.49	20.04	Master-node
3	4.84	0.36	12.38	0.39	17.97	node
	13.88	0.36	12.38	0.39	27.01	Master-node
4	4.43	0.45	16.04	0.34	21.26	node
	15.91	0.45	16.04	0.34	32.74	Master-node
5	4.05	0.55	20.15	0.31	25.06	node
	17.88	0.55	20.15	0.31	38.89	Master-node
6	3.78	0.64	24.8	0.3	29.52	node
	20.02	0.64	24.8	0.3	45.76	Master-node
7	3.59	0.72	28.34	0.29	32.94	node
	21.52	0.72	28.34	0.29	50.87	Master-node
8	3.43	0.79	32.11	0.29	36.62	node
	23.39	0.79	32.11	0.29	56.58	Master-node
9	3.28	0.89	36.23	0.29	40.69	node
	25.41	0.89	36.23	0.29	62.82	Master-node
10	3.17	0.96	40.27	0.28	44.68	node
	27.06	0.96	40.27	0.28	68.57	Master-node
11	3.07	1.05	44.99	0.27	49.38	node
	28.82	1.05	44.99	0.27	75.13	Master-node
12	3	1.12	48.87	0.27	53.26	node
	30.52	1.12	48.87	0.27	80.78	Master-node
13	2.96	1.19	52.62	0.27	57.04	node
	32.64	1.19	52.62	0.27	86.72	Master-node
14	2.9	1.26	56.78	0.27	61.21	node
	34.08	1.26	56.78	0.27	92.39	Master-node
15	2.86	1.33	59.83	0.27	64.29	node
	36.35	1.33	59.83	0.27	97.78	Master-node

Table 6-2 Message breakdown, different node density

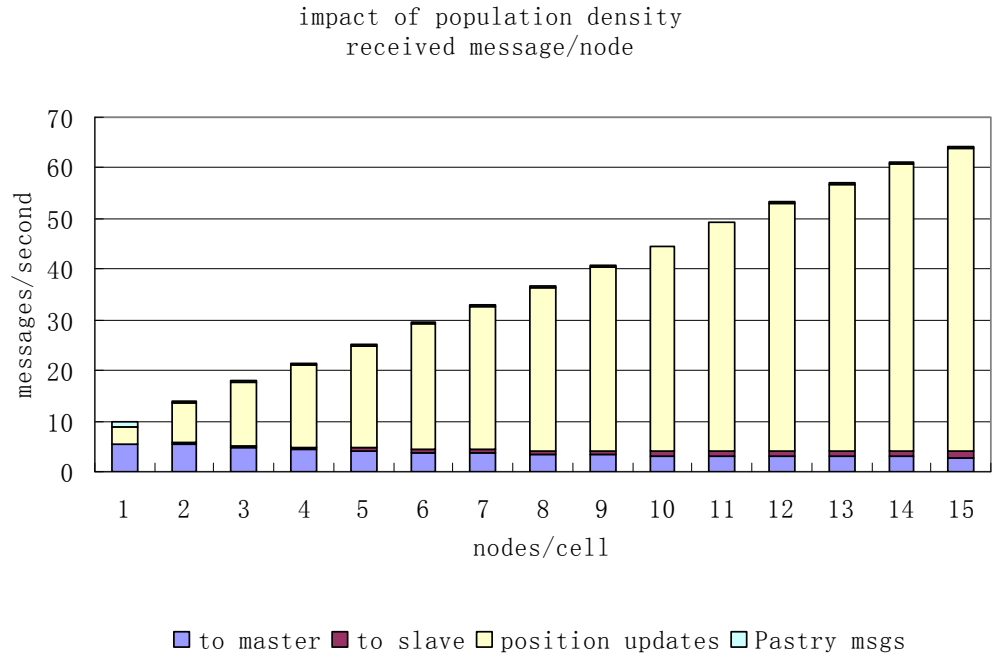


Figure 6-3 impact of population density, received messages/node

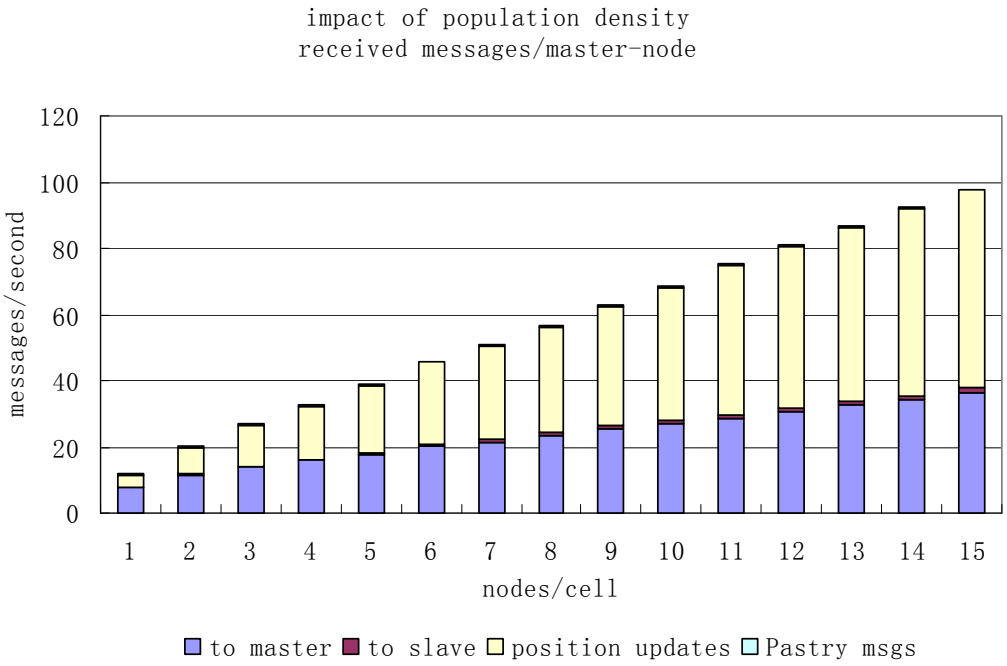


Figure 6-4 impact of population density, received messages/master-node

As we would expect, the number of messages received by each node increases linearly with the increase in node density as shown in Figure 6-3.

The number of ‘to master’ messages received by each master-node also increases linearly as shown in Figure 6-4. Then why does the average amount of ‘to master’ messages received by each player drop with the increase in node density as shown in Figure 6-3? This is because rise of the node density increases the total number of players in the experiment while the number of master-nodes is kept constant as the number of cells, which is 100 in the experiment. The amount of ‘to master’ messages received by all master-nodes is averaged over all players in Figure 6-3 and is averaged over all master-nodes in Figure 6-4. Although average of received ‘to master’ messages per node decreases, average of received ‘to master’ messages per master-node increases. The ‘to master’ messages received by an individual player node that happens to be a cell’s master-node increases with the increase in node density, not decreases.

6.4 Impact of Work Modes

The experiment results shown in the previous two sections reflect that both our player nodes and master-nodes are much more sensitive to player density than to the total number of players in the game. When crowding occurs, the workload of gathering players increases. The workload of their master-node is even higher. To deal with the problem, we design and implement two more work modes, subscribing cells and subscribing expanded AOI in addition to subscribing AOI, to offload some of master-node’s workload to its slave-nodes.

The performance of subscribing expanded AOI mode has been shown in previous section. In this section we first show the results from re-running the previous experiment with the rest two work-modes, subscribing AOI and subscribing cells. Then we compare the performances of these three modes.

The experiment settings are as same as that in previous section except their work mode. Side length of each hexagon is of 200 distance units. AOI radius of each player is

set to 200 distance units. We test system performance with node density 1 to 15 nodes per cell by raising total number of players in the experiments.

6.4.1 Subscribing AOI

With the subscribing AOI work mode, none of the master-node's workload is offloaded to others. Master-nodes detect and notify their slave-nodes whenever any node entering or leaving their AOI.

Below are the results obtained from re-running the experiment with subscribing AOI work mode.

Nodes /cell	Neighbour Discovery Overlay		Position updates	Pastry	Total	Received by
	to master	to slave				
1	37.42	0.27	0	0.84	38.53	node
	55.19	0.27	0	0.84	56.3	Master-node
2	48.56	0.46	0	0.49	49.51	node
	102.8	0.46	0	0.49	103.75	Master-node
3	56.2	0.65	0	0.37	57.22	node
	161.89	0.65	0	0.37	162.91	Master-node
4	60.52	0.85	0	0.31	61.68	node
	222.29	0.85	0	0.31	223.45	Master-node
5	62.64	1.08	0	0.31	64.03	node
	275.74	1.08	0	0.31	277.13	Master-node
6	63.69	1.29	0	0.29	65.27	node
	337.04	1.29	0	0.29	338.62	Master-node
7	64.1	1.43	0	0.27	65.8	node
	386.82	1.43	0	0.27	388.52	Master-node
8	64.86	1.71	0	0.27	66.84	node
	448.83	1.71	0	0.27	450.81	Master-node
9	65.22	1.85	0	0.26	67.33	node
	500.41	1.85	0	0.26	502.52	Master-node
10	65.48	2.13	0	0.28	67.89	node
	563.23	2.13	0	0.28	565.64	Master-node
11	64.64	2.24	0	0.27	67.15	node
	607.78	2.24	0	0.27	610.29	Master-node
12	65.41	2.42	0	0.26	68.09	node
	663.64	2.42	0	0.26	666.32	Master-node
13	65.89	2.61	0	0.26	68.76	node
	722.71	2.61	0	0.26	725.58	Master-node
14	65.74	2.88	0	0.26	68.88	node
	784.69	2.88	0	0.26	787.83	Master-node
15	65.85	3.04	0	0.26	69.15	node
	830.86	3.04	0	0.26	834.16	Master-node

Table 6-3 Message breakdown, subscribing AOI

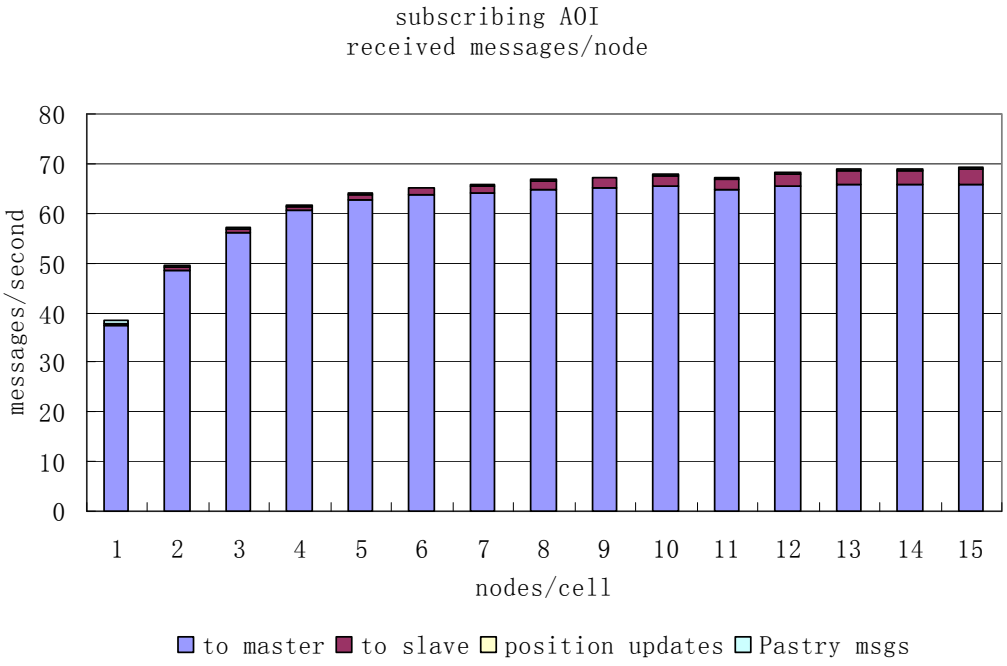


Figure 6-5 Subscribing AOI, received messages/node

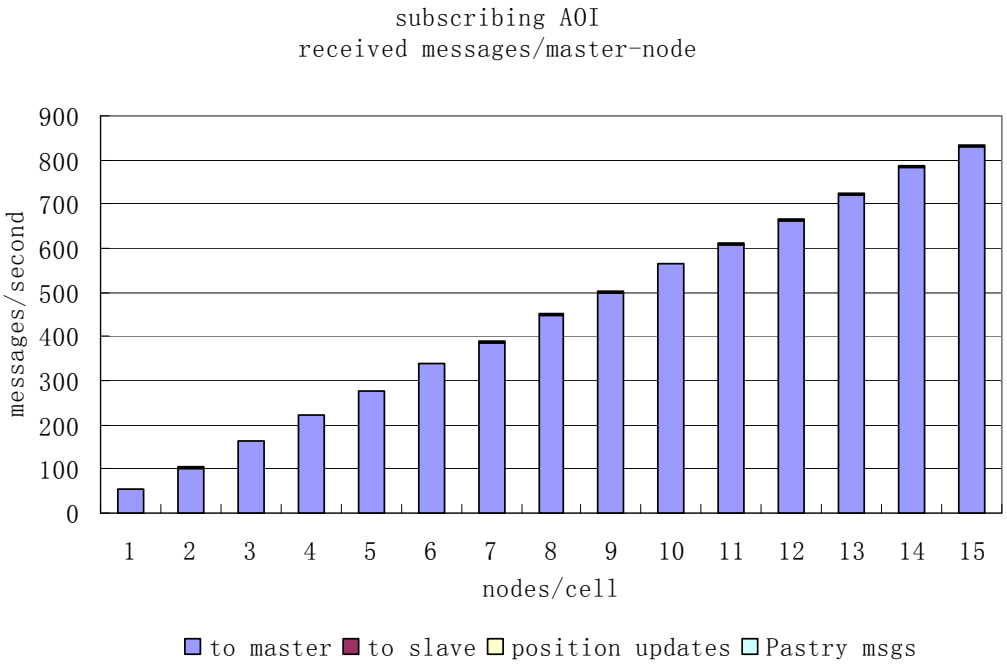


Figure 6-6 Subscribing AOI, received messages/master-node

As we would expect, the number of messages received by each master-node increases dramatically with the increase in node density as shown in Figure 6-6.

The facts revealed by experimental results are very interesting. First, the workload of master-nodes is much heavier than that of non-master nodes. Because the majority of messages in the system are 'to master' messages that are delivered only to master-nodes. The number difference between messages received by master-nodes and non-master-nodes are huge. While an average non-master node receives less than 4 messages per second, a master-node receives at least 57 messages per second as shown in Table 6-3.

Second, although the average of the messages received by non-master-nodes is low, it also increases with the increase in node density. This is because the number of nodes in an AOI increases with the increase in node density. Therefore, the frequency of nodes entering or leaving the AOI increases. The node of the AOI receives more and more notification from its master-nodes.

Finally, since the number of messages received either by master-nodes or non-master-nodes increases with the increase in node density, why does not the average of received messages by each node increase a lot as shown in Figure 6-5? This is because rise in node density also increases the total number of nodes in the experiment, but the number of master-nodes with heavy workload are fixed.

We can thus conclude that with the subscribing AOI work mode, our system performance is very sensitive to population density. Workload of the master-nodes is heavy and increases dramatically with the increase of the node density. The population density had better be set to 1 or 2 nodes per cell when running the system with this mode.

6.4.2 Subscribing Cells

With subscribing cell work mode, partial master-nodes' workload is offloaded to slave-nodes. Master-nodes provide their slave-nodes information about nodes in cells overlapping with slave-nodes' AOI instead of that in their AOI.

By re-running the experiment with subscribing cells work mode, we got experimental results below.

Nodes /cell	Neighbour Discovery Overlay		Position updates	Pastry	Total	Received by
	to master	to slave				
1	0.9	0.4	5.66	0.9	7.86	node
	1.09	0.4	5.66	0.9	8.05	Master-node
2	0.69	0.56	11.77	0.47	13.49	node
	1.38	0.56	11.77	0.47	14.18	Master-node
3	0.64	0.74	17.65	0.37	19.4	node
	1.75	0.74	17.65	0.37	20.51	Master-node
4	0.61	0.93	23.76	0.31	25.61	node
	2.13	0.93	23.76	0.31	27.13	Master-node
5	0.62	1.19	29.83	0.32	31.96	node
	2.68	1.19	29.83	0.32	34.02	Master-node
6	0.57	1.32	35.66	0.28	37.83	node
	2.97	1.32	35.66	0.28	40.23	Master-node
7	0.58	1.51	40.89	0.28	43.26	node
	3.44	1.51	40.89	0.28	46.12	Master-node
8	0.55	1.65	46.95	0.27	49.42	node
	3.77	1.65	46.95	0.27	52.64	Master-node
9	0.55	1.87	52.97	0.27	55.66	node
	4.22	1.87	52.97	0.27	59.33	Master-node
10	0.55	2.12	61.47	0.26	64.4	node
	4.68	2.12	61.47	0.26	68.53	Master-node
11	0.55	2.24	65.83	0.26	68.88	node
	5.06	2.24	65.83	0.26	73.39	Master-node
12	0.54	2.38	70.97	0.25	0.54	node
	5.46	2.38	70.97	0.25	79.06	Master-node
13	0.54	2.61	77.55	0.26	80.96	node
	5.94	2.61	77.55	0.26	86.36	Master-node
14	0.54	2.76	83.65	0.25	87.2	node
	6.29	2.76	83.65	0.25	92.95	Master-node
15	0.54	3.03	90.43	0.25	94.25	node
	6.88	3.03	90.43	0.25	100.59	Master-node

Table 6-4 message breakdown, subscribing cells

As we would expect, the number of messages received by each node increases linearly with the increase in node density as shown in Figure 6-7. By moving some workload from the master-nodes to the slave-nodes, the master-node's workload decreases while workload of each slave node increases. The majority of received messages by a node are position updates directly from nodes in cells overlapping with its AOI. Obviously, the number of nodes in cells overlapping with an AOI increases when node density of each cell increases. This increases the number of position updates received by each node.

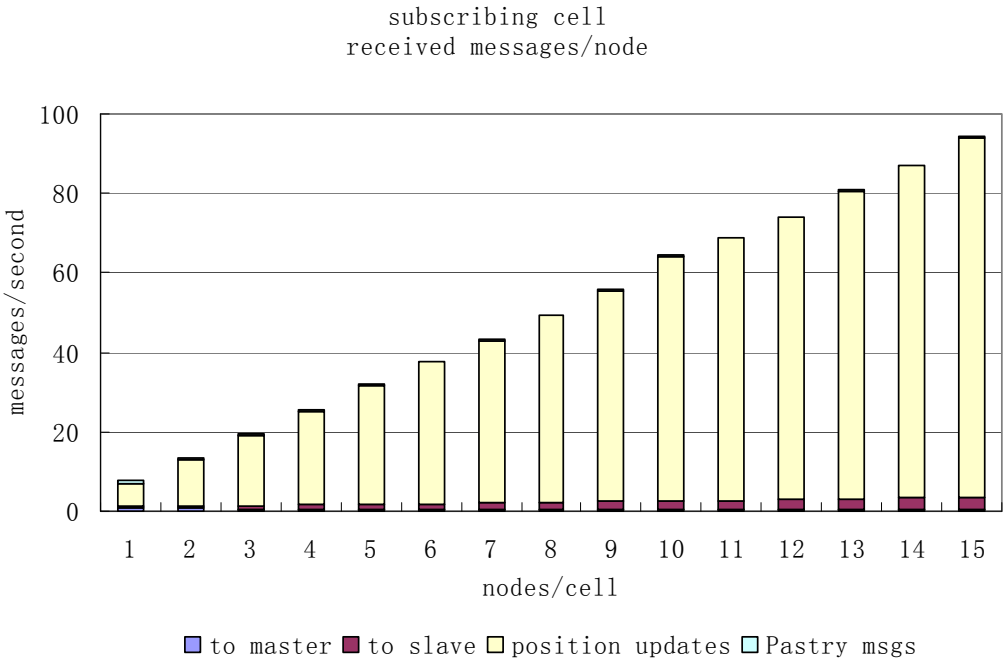


Figure 6-7 subscribing cells, received messages/node

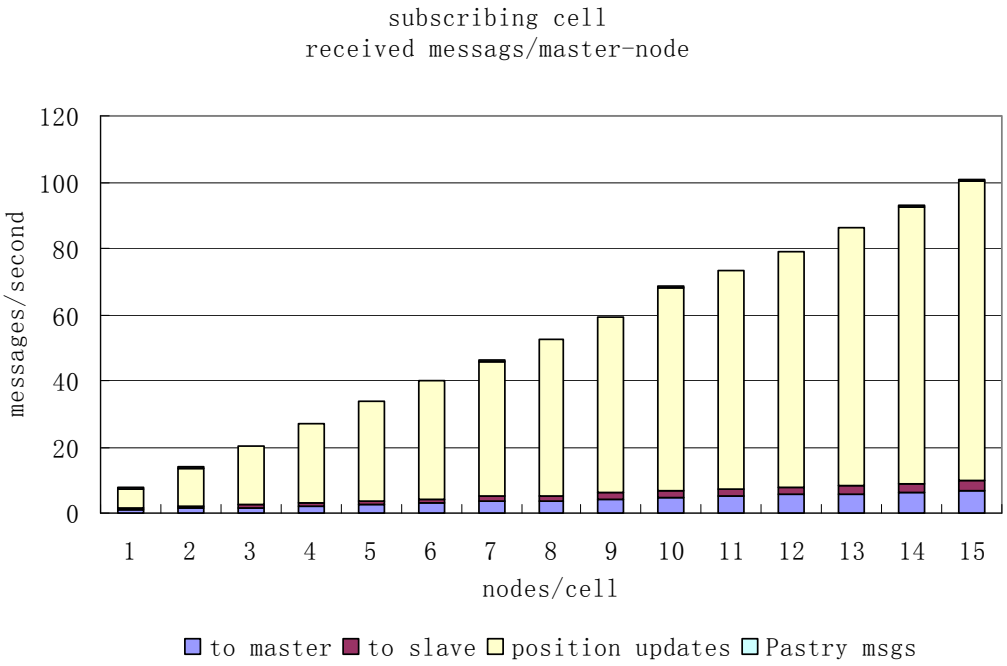


Figure 6-8 subscribing cells, received messages/master-node

6.4.3 Comparison

As we would expect, the master-node performance is improved significantly with both subscribing expanded AOI and subscribing cells work modes although all of these three modes are sensitive to node density. The performance of the subscribing expanded AOI work mode and that of the subscribing cells work mode are similar with subscribing expanded AOI mode slightly less sensitive to node density than subscribing cells mode.

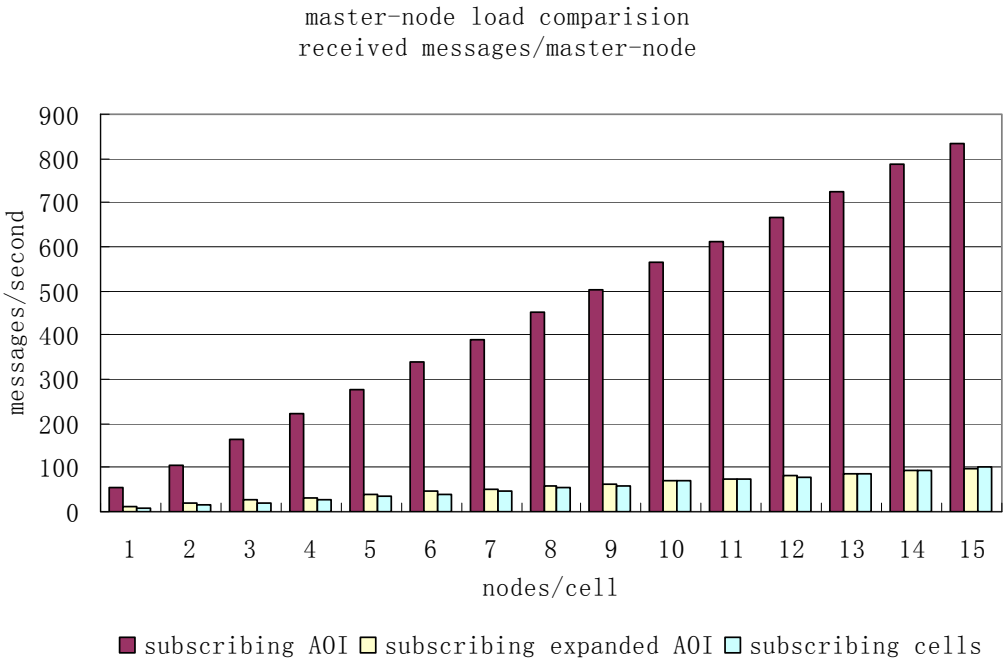


Figure 6-9 master-node load comparison

6.5 Impact of the Cell Size

Our system is sensitive to player density. When dividing the virtual world (VM) into cells, shall we choose the cell size that sets the average number of nodes in a cell lowest? Given VM size and an anticipated number of players, we explore the impact of cell size to our system performance in this experiment.

The experiment runs with expanded AOI mode with one position update every six movements. The virtual world is of 2580 distance units by 2220 distance units. The number of player nodes is 1500. AOI Radius of each node is fixed at 100 distance units.

The experiment results with different cell size are shown below.

Cell Side Length	Neighbour Discovery Overlay		Position updates	Pastry	Total	Received by
	to master	to slave				
50	17.16	1.72	50.71	2.09	71.68	node
	26.8	1.72	50.71	2.09	81.32	Master-node
60	12.27	1.76	50.92	1.44	66.39	node
	26.3	1.76	50.92	1.44	80.42	Master-node
70	9.25	1.63	50.52	1.15	62.55	node
	26.18	1.63	50.52	1.15	79.48	Master-node
80	6.85	1.55	50.48	0.88	59.76	node
	25.05	1.55	50.48	0.88	77.96	Master-node
90	5.54	1.5	50.27	0.74	58.05	node
	25.07	1.5	50.27	0.74	77.58	Master-node
100	4.46	1.42	50.08	0.6	56.56	node
	25.61	1.42	50.08	0.6	77.71	Master-node
110	3.87	1.37	49.96	0.56	55.76	node
	25.98	1.37	49.96	0.56	77.87	Master-node
120	3.49	1.35	50.19	0.48	55.51	node
	27.47	1.35	50.19	0.48	79.49	Master-node
130	3.21	1.31	50.48	0.43	55.43	node
	29.68	1.31	50.48	0.43	81.9	Master-node
140	3.04	1.26	49.8	0.39	54.49	node
	32.12	1.26	49.8	0.39	83.57	Master-node
150	2.85	1.25	50.52	0.35	54.97	node
	35.73	1.25	50.52	0.35	87.85	Master-node
160	2.79	1.21	49.99	0.33	54.32	node
	36.93	1.21	49.99	0.33	88.46	Master-node
170	2.67	1.21	50.32	0.31	54.51	node
	40.83	1.21	50.32	0.31	92.67	Master-node
180	2.61	1.17	49.98	0.3	54.06	node
	42.65	1.17	49.98	0.3	94.1	Master-node
190	2.5	1.16	50.03	0.27	53.96	node
	47.41	1.16	50.03	0.27	98.87	Master-node
200	2.46	1.14	50.35	0.25	54.2	node
	49.7	1.14	50.35	0.25	101.44	Master-node

Table 6-5 message breakdown, impact of cell size

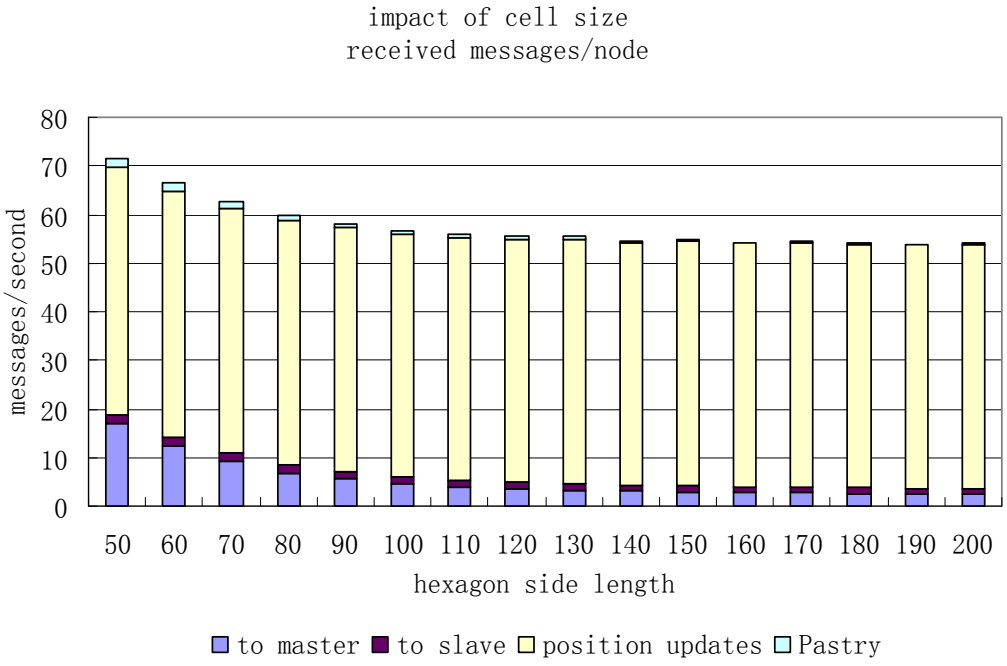


Figure 6-10 impact of cell size, received messages/node

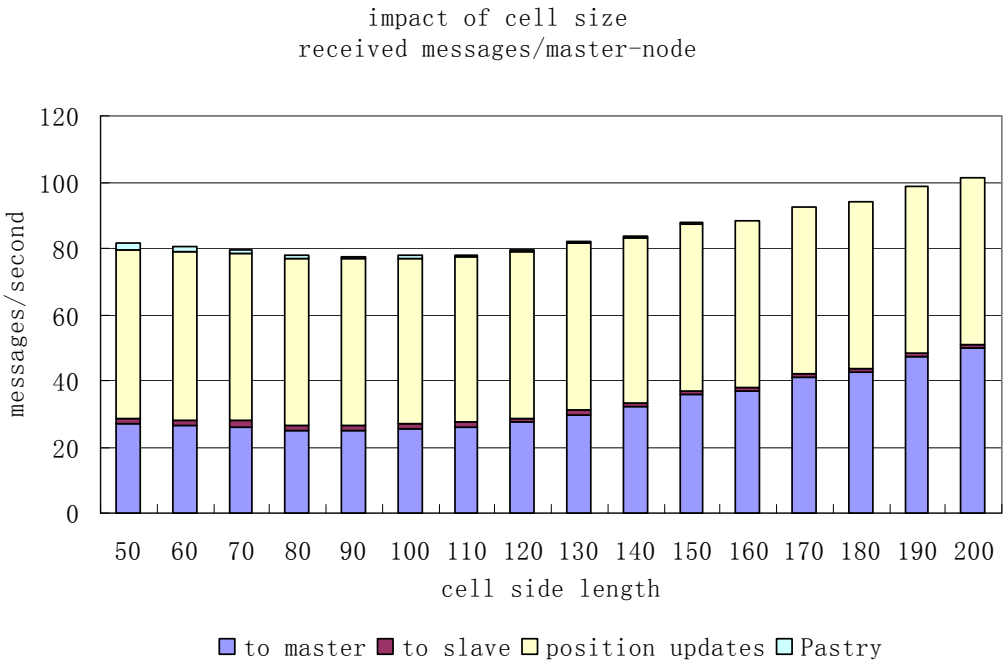


Figure 6-11 impact of cell size, received messages/master-node

Figure 6-10 shows average of received messages per node decreases with increase in cell size, misleading us to believe that our system performs better with bigger cells. Examining each component part of messages received by a node more carefully, we can

find the majority messages received by a node are position updates, the number of which is kept constant of 50 over all cell sizes. This is because the number of players in an expanded AOI does not change with cell size. But the number of cells in the experiment changes with cell size, thus the number of master-nodes changes with cell size. The big the cell size is, the fewer cells the experiment has, the fewer master-nodes are in the experiment. Each master-node has to do more work when their cell gets bigger as shown in Figure 6-11 because the number of slave-nodes each master-node takes care of increases. Although the workload of each master-node increases with the increase of cell size, the summation of their workloads decreases, resulting in the decrease of average of messages received by a node. However, cell size should be decided based on master-nodes' workload which is heavier than average load. For setting like in this experiment, hexagon side length to be of about 100 distance units is appropriate.

6.6 Impact of the AOI Radius

We also evaluated the impact of AOI radius by conducting experiments with different AOI radius for players. In the first experiment, each AOI in the experiment has a unified radius. In the second, AOI radiuses are equally likely chosen from a range. Both experiments run with same setting: 10 x 10 cells, 1000 players, hexagon side length of 200, subscribing expanded AOI work mode.

6.6.1 Unified AOI Radius

The experiment was repeated multiple times, each time with a different AOI radius, which is unified for all AOIs in the experiment.

As we would expect, messages received by both the slave-nodes and the master-nodes increases with the increase of AOI radius. Obviously, the number of nodes in an AOI increases when an AOI gets bigger. Therefore, a node connects with more nodes and

receives move position updates. Also, when an AOI gets bigger, the number of cells overlapping with the AOI increase as well. Thus its master-node subscribes for more cells and gets more updates.

aoi radius	neighbor discovery overlay		position updates	pastry msgs	total	received by
	to master	to slave				
60	0	0.36	10.54	0.27	11.17	non-master node
	25.1	0.36	10.54	0.27	36.27	master-node
80	0	0.44	14.55	0.27	15.26	non-master node
	25.57	0.44	14.55	0.27	40.83	master-node
100	0	0.52	18.29	0.26	19.07	non-master node
	25.67	0.52	18.29	0.26	44.74	master-node
120	0	0.6	22.46	0.26	23.32	non-master node
	25.84	0.6	22.46	0.26	49.16	master-node
140	0	0.69	27.08	0.26	28.03	non-master node
	25.96	0.69	27.08	0.26	53.99	master-node
160	0	0.78	31.36	0.27	32.41	non-master node
	26.16	0.78	31.36	0.27	58.57	master-node
180	0	0.87	36.4	0.26	37.53	non-master node
	26.13	0.87	36.4	0.26	63.66	master-node
200	0	0.97	40.34	0.29	41.6	non-master node
	27.3	0.97	40.34	0.29	68.9	master-node
220	0	1.04	44.38	0.27	45.69	non-master node
	28.62	1.04	44.38	0.27	74.31	master-node
240	0	1.12	47.95	0.28	49.35	non-master node
	30.73	1.12	47.95	0.28	80.08	master-node
260	0	1.24	53.71	0.28	55.23	non-master node
	32.63	1.24	53.71	0.28	87.86	master-node
280	0	1.32	58.59	0.27	60.18	non-master node
	34.07	1.32	58.59	0.27	94.25	master-node
300	0	1.36	60.9	0.27	62.53	non-master node
	35.16	1.36	60.9	0.27	97.69	master-node
320	0	1.45	65.27	0.29	67.01	non-master node
	39.25	1.45	65.27	0.29	106.3	master-node
340	0	1.52	69.48	0.28	71.28	non-master node
	41.43	1.52	69.48	0.28	112.7	master-node
360	0	1.56	73.68	0.28	75.52	non-master node
	42.84	1.56	73.68	0.28	118.4	master-node
380	0	1.63	79.03	0.28	80.94	non-master node
	44.38	1.63	79.03	0.28	125.3	master-node
400	0	1.65	82.67	0.27	84.59	non-master node
	44.68	1.65	82.67	0.27	129.3	master-node

Table 6-6 message break down, unified AOI radius

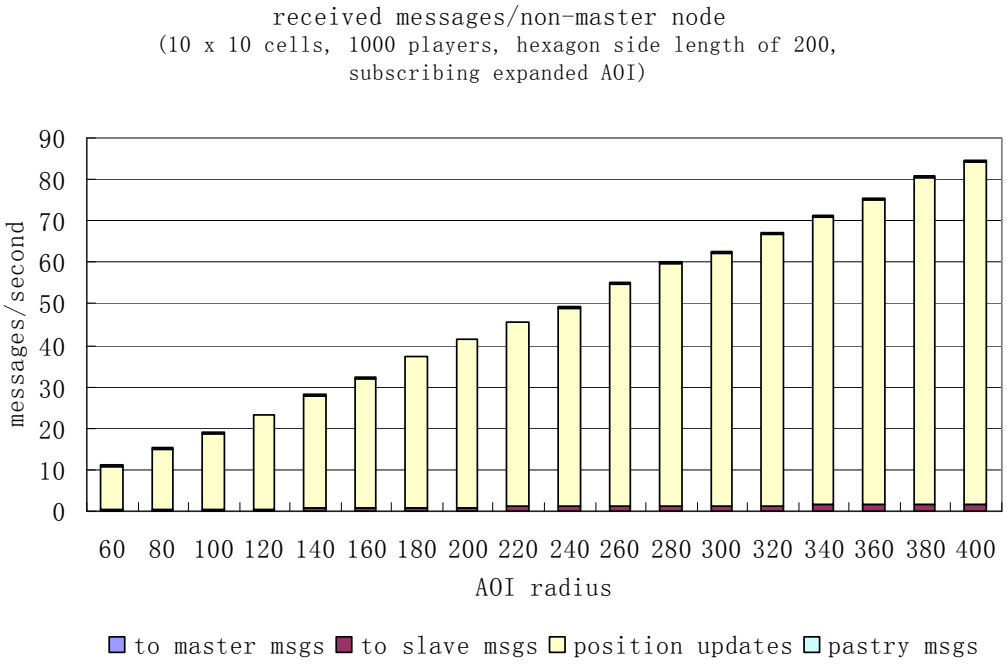


Figure 6-12 unified AOI radius, received messages/non-master node

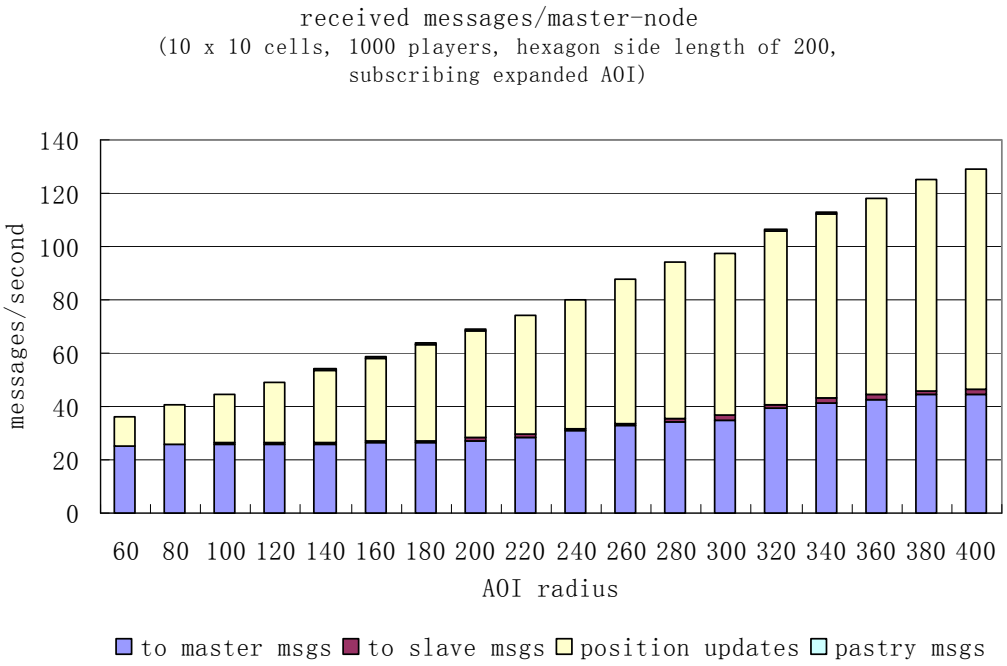


Figure 6-13 unified AOI radius, received messages/master-node

6.6.2 Uniform Distributed AOI Radius

In this experiment, the AOI radius for each player is randomly selected from a range. The experiment is executed multiple times with different AOI range for each time.

radius range	neighbor discovery overlay		position updates	pastry msgs	total	received by
	to master	to slave				
10~110	0	0.35	10.87	0.28	11.5	non-master-node
	25.12	0.35	10.87	0.28	36.62	master-node
30~130	0	0.44	14.73	0.27	15.44	non-master-node
	25.39	0.44	14.73	0.27	40.83	master-node
50~150	0	0.52	18.5	0.26	19.28	non-master-node
	25.78	0.52	18.5	0.26	45.06	master-node
70~170	0	0.62	23.23	0.27	24.12	non-master-node
	25.85	0.62	23.23	0.27	49.97	master-node
90~190	0	0.7	27.24	0.27	28.21	non-master-node
	25.98	0.7	27.24	0.27	54.19	master-node
110~210	0	0.8	31.81	0.28	32.89	non-master-node
	26.28	0.8	31.81	0.28	59.17	master-node
130~230	0	0.89	36.69	0.27	37.85	non-master-node
	26.97	0.89	36.69	0.27	64.82	master-node
150~250	0	0.96	40.63	0.27	41.86	non-master-node
	28.2	0.96	40.63	0.27	70.06	master-node
170~270	0	1.07	44.82	0.29	46.18	non-master-node
	30.2	1.07	44.82	0.29	76.38	master-node
190~290	0	1.14	48.76	0.28	50.18	non-master-node
	31.47	1.14	48.76	0.28	81.65	master-node
210~310	0	1.22	53.09	0.28	54.59	non-master-node
	33.33	1.22	53.09	0.28	87.92	master-node
230~330	0	1.27	56.23	0.27	57.77	non-master-node
	35.1	1.27	56.23	0.27	92.87	master-node
250~350	0	1.35	60.46	0.27	62.08	non-master-node
	37.27	1.35	60.46	0.27	99.35	master-node
270~370	0	1.45	64.82	0.3	66.57	non-master-node
	39.69	1.45	64.82	0.3	106.3	master-node
290~390	0	1.49	69.45	0.28	71.22	non-master-node
	41.21	1.49	69.45	0.28	112.4	master-node
310~410	0	1.54	73.38	0.27	75.19	non-master-node
	42.67	1.54	73.38	0.27	117.9	master-node
330~430	0	1.59	77.13	0.28	79	non-master-node
	43.95	1.59	77.13	0.28	123	master-node
350~450	0	1.63	80.42	0.28	82.33	non-master-node
	44.69	1.63	80.42	0.28	127	master-node

Table 6-7 message break down, distributed AOI radius

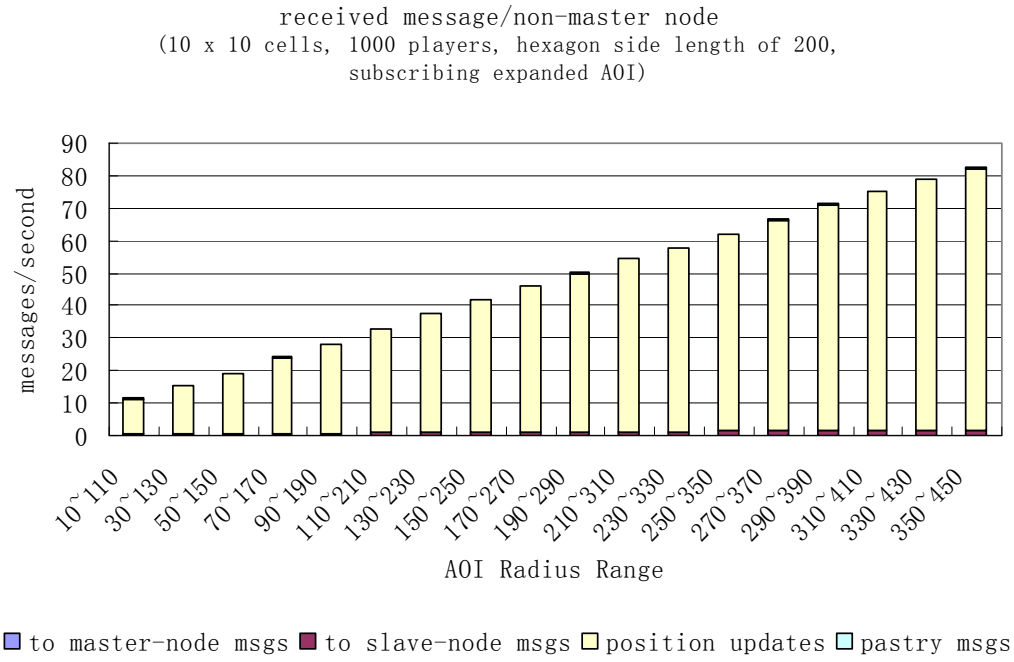


Figure 6-14 distributed AOI radius, received messages/non-master node

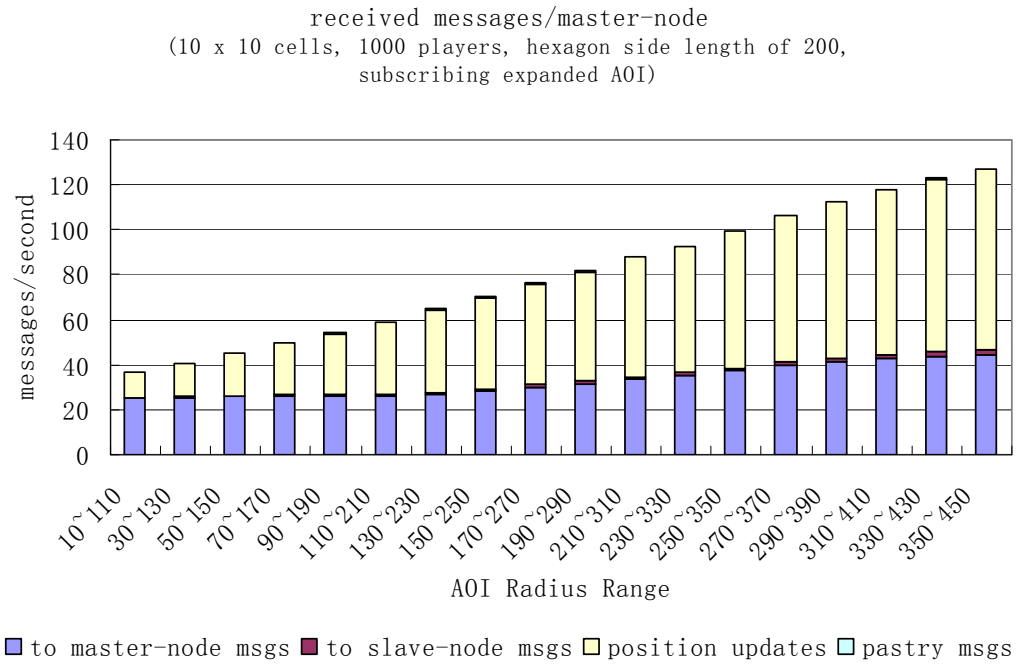


Figure 6-15 distributed AOI radius, received messages/master-node

The number of messages received by either non-master nodes or master-nodes increases with increase in mean of AOI radius. Similar to the experiment results of unified radius, the number of position updates, the majority messages received by slave

nodes, increases with increase in AOI. And enlarged subscription list causes master-nodes to receive more updates as well.

6.6.3 Comparison

From Figure 6-16 we can see, the average of the received messages by either the slave-nodes or the master-nodes is similar when the players have unified AOI radius of r or have their radius uniformly distributed in $(r - \Delta r, r + \Delta r)$. However, their distributions of received messages should be different.

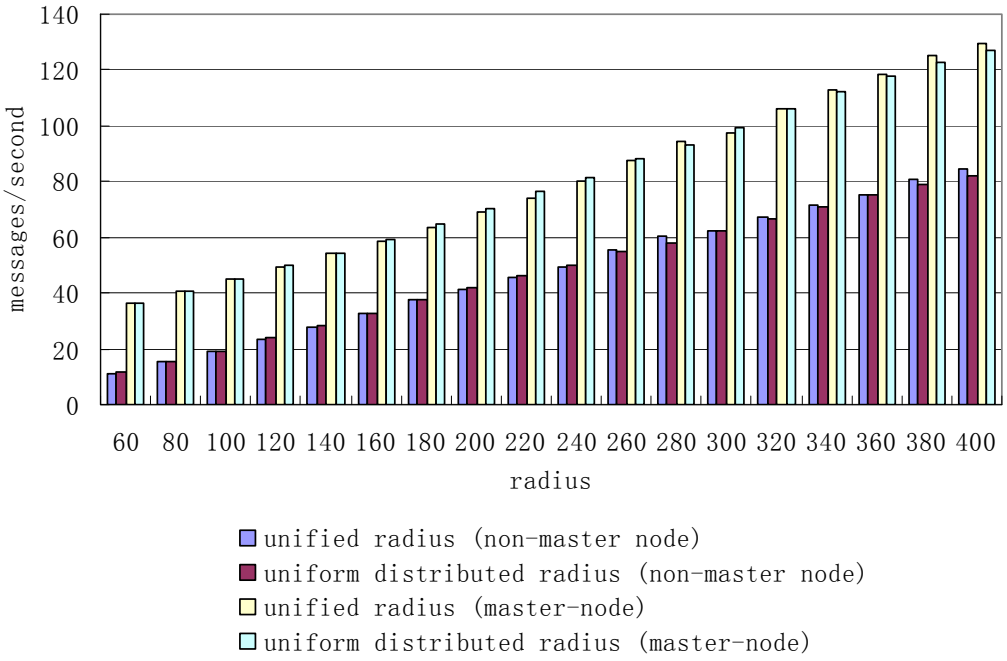


Figure 6-16 Comparison between unified and distributed radius

.

Chapter 7

Conclusions and Future Work

7.1 Summary

Massively multiplayer online games are becoming more and more popular these days. To improve the scalability of MMOGs, P2P systems have been proposed to replace the traditional client-server model as the structure for MMOGs. Although each of existing P2P solutions improves, in various degrees, the scalability of MMOGs, they still have deficiencies in interest management. Interest management, which can be easily solved in client-server model, becomes a challenge issue in P2P-based MMOGs due to the distribution of the view of the game world.

In this thesis, we propose a P2P-based middleware to achieve interest management for applications of networked virtual environment. Our system consists of two structured P2P networks, a DHT-based rendezvous overlay and a *Neighbour Discovery Overlay*. Each of these two overlays consists of all the nodes in the system.

Neighbour Discovery Overlay groups nodes into a two-level hierarchy according to nodes' location in the virtual world. The virtual world is divided into hexagon cells. Nodes in the same cell form a group and one of them is selected as the master-node of the

cell. Nodes are connected to their master-nodes. And master-nodes are connected with master-nodes of adjacent cells, forming a master-node tier.

The DHT-based rendezvous overlay is proposed to locate information of current master-nodes (which changes when any master-node leaves its cell or any node enters an empty cell) so that a node can connects with the master-node of the cell it enters or a master-node can connect with master-nodes of adjacent cells by retrieving the information.

Nodes update their master-nodes on their current AOI; in return their master-nodes inform them what nodes are currently in their AOI. In order for master-nodes to find out nodes in certain cells, a routing and communication scheme is applied on top of the master-node tier of *Neighbour Discovery Overlay* for efficient exchange of information between master-nodes.

A prototype system, called MOPAR+, is implemented in Java and experimented with a simple simulation game. In every instance, the experimental results justify our hypothesis that our system fulfills its interest management goal and has a good scalability property. A visualization tool is developed to enable users to explore the behaviour and structure of our system, by interacting with a simulation game.

7.2 Future Work

There are four major directions for further research.

First, our design assumes that every game entity has a finite area of interest that is around the entity and that an action at one place affects only its neighbourhood. The assumption holds for most cases. However, in some games there are entities whose AOI has different shape. For example, warriors in the same troop should be able to communicate with each other even when they are not in each other's sight. On the other hand, warriors should not be aware of other warriors in the enemy troop if they are not in

sight. In this case, the system should help players find out their troop members in cells that can not be seen by their avatar. How to do this efficiently calls further investigation.

Second, we realized that our interest-management scheme should not be limited to MMOGs. Any location-based application in which entities are mobile can be the potential user of our system. For example, mobile phones, laptops, or even vehicles supported by GPS system can use the proposed scheme. 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 neighbors but also profile info (metadata or attributes) of neighbors.

Third, our design can be modified and adopted to the client-server model to reduce server's workload. For example, if the bookkeeping works undertaken by the rendezvous overlay is not heavy for a single server, we could replace the rendezvous overlay with a management server. Moreover, we could replace all master-nodes in the system with a cluster of servers. Because a server can fill in for several master-nodes, the number of replacement servers is much fewer than that of master-nodes in the system.

Finally, we will improve the development of our system to let it become a complete MMOG middleware. With this middleware, game developers can concentrate on their games development without being bothered by the networking technical aspects. For

example, game updates between players could be delivered via our system by providing two more functions *send()* and *delivery()*in our API.

Eventually, we want our infrastructure to be extensively used as a network framework for easily developing and deploying user-defined MMOGs in the community.

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.

[33] An Overview of Multiplayer Gaming

[34] Comparing Interest Management Algorithms for Massively Multiplayer Games

[35] Three-Tiered Interest Management for Large-Scale Virtual Environments

[36] M. Merabti & A. E. Rhalibi. AOIM in Peer-to-Peer Multiplayer Online Games. IEEE Communications Society Globecom 2004 Workshops

[37] Interest Management Middleware for Networked Games

[38] A Federated Peer-to-Peer Network Game Architecture

[39] Interest Management in Large-Scale Virtual Environments

[40] Distributed Architectures for Massively-Multiplayer Online Games

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