uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Saurabh Ratti

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical and Computer Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Distributed Location-Aware Routing Architecture for P2P Massively Multiplayer Online Games

TITRE DE LA THÈSE / TITLE OF THESIS

Shervin Shirmohammadi

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

Chris Joslin

Amiya Nayak

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

A Distributed Location-Aware Routing Architecture for P2P Massively Multiplayer Online Games

SAURABH RATTI

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE
IN ELECTRICAL AND COMPUTER ENGINEERING

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada

Your file Votre référence
ISBN: 978-0-494-65546-7
Our file Notre référence
ISBN: 978-0-494-65546-7

Abstract

Populations in contemporary Massively Multiplayer Online Games (MMOG) continue to grow, a trend that current client-server architectures are hard pressed to sustain. Application of peer-to-peer concepts and technologies to the domain of MMOG communications can address the issues of scalability and single points of failure associated with the client-server model. A proposed system takes an existing peer-to-peer network overlay as a foundation for peer connection management, and adapts it to have location-awareness by applying the Hilbert Space-Filling Curve. The new routing architecture's location awareness is done with respect to the virtual environment, in order to achieve area-of-interest updates to interested peers with the latency of a single hop. This is done in manner that is fully distributed, thus sharing the work load as evenly as possible across peers without requiring centralized servers. This thesis reviews existing approaches, details the proposed system, and finally presents a proof-of-concept and associated evaluations.

Acknowledgements

I would like to take this opportunity to acknowledge those who have aided and supported me during my studies and the creation of this thesis.

First and foremost is my supervisor, Dr. Shervin Shirmohammadi, who provided the opportunity to pursue my studies farther than I had ever imagined. It has been his encouragement, support and tutelage which enabled me to expand and develop myself in many aspects of life, academic as well as personal. It has been a pleasure and a privilege to learn from and work with him, and I am deeply honoured to have him as my supervisor and mentor.

I must acknowledge Dr. Behnoosh Hariri for her vision and guidance with regards to this work.

I also salute all my colleagues in the DISCOVER and CARG labs at the University of Ottawa, whose companionship and encouragement I count among my blessings.

Dearest of all is my family, my parents and brother. They are my greatest source of strength, with the endless love and support they have given me in all my endeavours.

Saurabh Ratti

Ottawa, January 2010

Table of Contents

Table of Figures

Table of Tables

Table of Algorithms

List of Acronyms and Definitions

ALM	Application Layer Multicast
AOI	Area of Interest
DHT	Distributed Hash Table
DHT-ID	Distributed Hash Table Identifier
IP	Internet Protocol
MMOG	Massively Multiplayer Online Game
MSB	Most Significant Bit
MTU	Maximum Transmission Unit
QoS	Quality of Service
P2P	Peer-to-Peer
RTT	Round-Trip Time
TCP	Transport Control Protocol
UDP	User Datagram Protocol

Chapter 1 - Introduction

There is no doubt that video games have become a part of the 21st century's entertainment landscape. As a recognized form of leisure, video games have placed their mark on a number of different aspects of society. Arts and culture are an integral part of the video game industry, as a large set of creative skills needed to create a successful entry into the market. Games take inspiration from classic story-telling to present compelling narratives in a format that takes cues from film-making. With respect to social interaction, video games were initially stigmatized as being a highly solitary activity; this has changed in recent years however, with the advent of multiplayer game play. Of the different types of multiplayer game play, one of the most prevalent recent trends has been that of Massively Multiplayer Online Games (MMOGs). MMOGs, such as Blizzard's World of Warcraft [1], distinguish themselves as they allow a large, or "massive", number of players to interact and play together. The MMOG format has proven to be highly popular, as there are approximately 16 million currently active MMOG subscriptions [2].

The application for large scale simulation has tremendous potential to go beyond that of highly profitable entertainment, as evidenced by an occurrence within the online entertainment realm. In 2007, World of Warcraft players in a certain area were affected by a "spell" effect that reduced their health by small amount in consistent intervals [3]; in addition to this the spell had the characteristic of being transmissible to nearby players. Due to a bug, this spell's effectiveness was not fully contained to the initial predetermined area, and was brought to other areas and transferred to other players, resulting in an online plague-like outbreak. The resulting crisis also brought to light the various reactions shown by those infected: some players attempted to aid others despite the risk imposed; some players deferred to self-preservation and left the highly infected areas; some players ran contrary to both of these and attempted to deliberately infect the uninfected.

This accidental plague led researchers to consider large scale simulation as an avenue for research in the spread of infectious diseases, as it also allowed human behaviour to be incorporated as a contributing factor. In order to properly accomplish this and other human behaviour driven research, it is logical that a larger, or "massive", population is needed.

It seems however, that it is not only scientific research that is venturing into the domain of MMOGs. Metaverses are an extension of the MMOG ideal, where players can interact in a seamlessly persistent virtual world [4]. What differentiates the metaverse concept is that users can generate and contribute content to the world. Second Life [5], a popular contemporary metaverse, has found itself attracting an amalgam of the global community into its virtual community, as nations, businesses and academia establish their online presence alongside other individuals. The Swedish Institute has established Miramare, a virtual Swedish embassy within Second Life; global public-relations firm Edelman has created its own island. Second Life is also a popular venue for academic endeavours, as it provides the opportunity to host globally diverse students in single classroom.

This trend of online virtual presence, whether of individuals or collective entities, whether for business or pleasure, is growing to include a greater number of people. MMOGs come in wide and diverse formats and serve various purposes, so it follows that they will attract many different users; the continuing vogue of MMOGs will serve as further incitement for new players to join and establish themselves as well. The impact of this trend is that the currently desired "massive" player populations are instead becoming inevitable. Accommodating these increasing populations will require underlying MMOG infrastructure that can suitably sustain the growth.

1.1 Motivation

In any scenario of developing and deploying a large scale virtual environment, the customary use of client-server architecture entails several well known problems [6]. Perhaps the most grievous is the dependency of the entire simulation on the central server, a dependency which manifests itself in multiple drawbacks. The first is that the server becomes a central point of failure, for if it suffers any form of system failure, the shared

game world is lost and players can no longer communicate and thus no longer participate. A corollary of the server dependency is that clients are connected solely to the server, and thus if their specific network connection to the server suffers any failure, there is no alternative route for communication which results in their loss of ability to participate in the game.

Lack of scalability is another issue that plagues the client-server paradigm. Resources that are required to serve a game environment can be placed into two broad classifications: computational and network resources. The unfortunate reality is that neither of these resources is infinite. Computational resources are constrained by the limits of contemporary computing technology, and therefore a single server is insufficient when considering a "massive" number of players. Server farms can be constructed to provide sufficient computational power, but these have a hefty initial deployment cost and then incur high ongoing costs in order to remain functional. Not only does technical repair and replacement consume financial resources, but continuous power for the computing, networking and cooling systems does so as well. And yet despite these expenditures, server farms still cannot provide truly seamless game worlds where all players exist in a single reality. Rather, the world is divided up into shards and zones, akin to parallel realities and puzzle pieces respectively. Each of these realms can be assigned to separate servers, with little or no interaction and visibility between players that exists in differing realms.

With regards to network resources, infinite bandwidth is also completely unattainable in the face of the contemporary Internet, an unavoidable communications medium when considering globally diverse players. The nature of Internet means that it is not only nebulous and heterogeneous, but cannot be managed or reserved. Interestingly, the aforementioned world partitioning often has the effect of having players within a zone share a common interest in game update data. This is a prime example of a network multicast scenario, where a single packet is destined for multiple destinations and handled in such a way that there is an efficient division of labour in duplicating and routing the packet. The ideal solution for this is Internet Protocol (IP) multicast, but it is commonly known that this is a feature not widely available on the Internet.

By its very nature, peer-to-peer technology inherently addresses many of these issues that are associated with the client-server paradigm. Peer-to-peer, commonly abbreviated as P2P, is a distributed networking paradigm where participants contribute their local resources to the larger system, thus making them equal peers rather than clients. Pure P2P systems accomplish this paradigm without central coordinators or authoritative entities, such as servers. This automatically eliminates the issue of servers being a central point of failure, as they are no longer central to the task, if they exist at all. Scalability is achieved by the contribution of resources, as a greater number of peers not only means increased load but also increased resources. And lastly, though IP multicast is not an option, P2P networks have been used to create a wide-variety of application layer multicast (ALM) protocols [7], further revealing P2P as an alternative candidate to client-server.

Given the apparent advantages provided by P2P technology, it is therefore wished to employ it as a feasible alternative to the client-server architecture in supporting deployment of MMOGs.

1.2 Research Problem & Objective

While applying P2P technology will solve problems such as scalability and single point of failure, it does introduce its own set of problems which must be addressed. Accomplishing tasks in a distributed manner is often more complex without the presence of a central entity that has complete situational knowledge. As each peer has limited capabilities and resources, protocols must efficiently use local resources of multiple peers in order to accomplish tasks as optimally as possible.

Security, authentication of peers and data, and cheating are aspects that are difficult to address in P2P systems. Due to the fact that multiple peers participate in system events, sometimes anonymously, malicious behaviour can be difficult to prevent and track. Collusion between peers, man-in-the-middle attacks and hacking of peer-side software are examples of issues that are complicated without the presence a central, trustable entity. While they should be addressed before public deployment they are not however, within the present scope.

The focus of this work lies in the networking aspect of P2P systems, where each peer must maintain connections to multiple other peers. Each peer determines for itself which other peers to connect to. Since achieving mutual awareness between all active peers would devour network resources, connectivity decisions must be made with awareness of only a subset of the total set of peers. Nonetheless, the resulting network of peer links should be constructed in such a way that the overall organization is geared towards efficiently achieving the goals of the system.

In the case of MMOG networking communications, the goal of the system is for a peer to be able to disseminate game updates to other interested peers. This 'interest' is determined by which players are within each others' virtual environment locality; peers that are within each others' area-of-interest (AOI) should receive game updates from each other. These game updates should therefore be routed through the peer network as efficiently as possible with respect to virtual locality.

Specifically, the goal is to take an existing P2P network overlay as a foundation for distributed MMOG connection management, and adapt it to have location-awareness by applying the Hilbert Space-Filling Curve. The new routing architecture's location awareness is done with respect to the virtual environment, so that routing efficiency is increased for peers that have mutual awareness of each other, as these peers would have greater game update message exchange with each other.

This research objective applies P2P technology in a fully distributed manner, thus sharing the work load as evenly as possible across clients and not requiring centralized entities. The use of P2P technology alleviates problems associated with centralized architectures, such as scalability and single points of failure.

1.3 Research Contributions

This thesis presents a distributed, location-aware routing architecture as a feasible solution to managing MMOG communications. The system is based upon Pastry, a generic routing overlay, which is adapted specifically for the requirements of distributed gaming with a massive number of players.

Several contributions are included in this new system, which are as follows:

- A fast, geometric based algorithm for mapping location to the Hilbert Curve to achieve location-aware routing.
- Adaptation of the traditional Hilbert Curve to a non-uniform distribution for efficient mapping of player location to DHT-IDs.
- Handling of player movement though an adapted join protocol for DHT-ID reassignment, with mechanisms for pre-empting request contention and maintaining proper inter-peer connectivity.
- A range query algorithm with optimizations for improved memory usage and execution speed.
- AOI update scheme that achieves 1-hop game update dissemination, based on the aforementioned range query algorithm.
- Proof-of-concept and performance evaluation as validation of the design and theory.

1.4 Research Publications

The peer reviewed publications related to the field of networked gaming are given below.

Journal Papers:

S. Ratti, B. Hariri, and S. Shirmohammadi, "A Survey and Analysis of First Person Shooter Gaming Traffic on the Internet," *IEEE Internet Computing*, (accepted, to appear), 2010. [8]

Conference Publications:

S. Ratti, B. Hariri, and S. Shirmohammadi, "NL-DHT: A Non-uniform Locality Sensitive DHT Architecture for Massively Multi-user Virtual Environment Applications," in *Proceedings of 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS '08)*, Melbourne, Australia, December 2008, pp. 793-798. [9]

B. Hariri, S. Ratti, S. Shirmohammadi, and M. R. Pakravan, "A Distributed Latency-

Aware Architecture for Massively Multi-User Virtual Environments," in *IEEE Internation Workshop on Haptic, Audio And Visual Environments and Games (HAVE 2008)*, Ottawa, Canada, October 2008, pp. 53-58. [10]

R. Iqbal and S. Ratti, "A Distributed Camera Network Architecture Supporting Video Adaptation," in *Proceedings of 3rd ACM/IEEE International Conference on Distributed Smart Cameras*, Como, Italy, 2009. [11]

S. Ratti, C. Towle, P. Proulx, and S. Shirmohammadi, "FizzX: Multiplayer Time Manipulation in Networked Games," in *Proceedings of 8th Workshop on Network and Systems Support for Games (ACM NetGames)*, Paris, France, 2009. [12]

Other Conference Publications:

This last publication is not related to this thesis, but was published within the duration of the author's Masters studies.

J. Parri and S. Ratti, "Trigonometric Function Approximation Neural Network Based Coprocessor," in *Proceedings of 2nd Microsystems and Nanoelectronics Research Conference*, Ottawa, Canada, 2009, pp. 148 - 151. [13]

1.5 Thesis Outline

The majority of this thesis presents the design and evaluation of the distributed location-aware routing architecture based on adapting an existing P2P overlay network. Accordingly, the remainder of this thesis is structured as follows:

Chapter 2 - Background provides background information on MMOG communications, the adapted P2P overlay Pastry, and the Hilbert Curve which factors prominently in the location-awareness of the system.

Chapter 3 - Related Work discusses existing approaches taken by a variety of distributed MMOG networking architectures.

Chapter 4 - Proposed System specifies the design of the location-aware routing architecture, specifically regarding building location-awareness into the routing overlay, accommodating player movement and achieving AOI game updates.

Chapter 5 - Proof of Concept details the architecture's proof of concept implemented for evaluation purposes.

Chapter 6 - Performance presents the evaluation results of simulations carried out to determine the performance of the aforementioned proof-of-concept.

Chapter 7 - Conclusion & Future Work summarizes and concludes the thesis, while outlining venues for future research.

Chapter 2 - Background

2.1 MMOG Communication

With MMOGs, there are two basic classifications for entities, with the first being whether an entity is a player or an object. The concept of a player is simply the avatar (representation) of an active person playing the game at a physical computer in the real world. Objects on the other hand, are purely virtual and only exist within the virtual environment; an example of this would be a virtual rock on the virtual ground. The second classification is whether an entity is mutable or immutable, meaning whether the state of the entity can change or not. Players are obviously mutable: if they were not, the real world participant would be unable to do anything in the game. Objects can be classified as one or the other; to continue the example of the rock, a mutable rock would be identified as something that a player can kick, pick up or break apart; an extreme example of an immutable rock would be a mountain that is a permanent part of the terrain. Within the scope of this work, immutable objects are a non-issue as they tend to be part of the environmental foreknowledge included with the game application or downloaded at game session start. Players and mutable objects are both more complicated, as their state can be manipulated by multiple other entities in many cases. For this reason, every mutable entity has an 'owner' which handles applying updates to the entity state's primary, or authoritative, instance. This entity management is outside the scope of this work, as there are existing systems developed to deal with object replication, state consistency and ensuing dissemination.

The updates applied to players and mutable objects however, is the focus here. As players are active entities within the simulation, they are the main producers of "update messages". These update messages contain state information on the movement and actions of a player, the equivalent of what is perceived through sight and sound in the real world. A player's information must be disseminated to the other players whose area-of-interest (AOI) the given player is within. A player's AOI is, typically, the local area surrounding a player that

said player will perceive. If two players are in each other's AOI the perception is usually mutual, and thus they should receive updates from one another. This perception equivalence can be interpreted so that it can also be said that a player should update all entities within its AOI. This definition implicitly includes updates to mutable objects that are affected by a player, as they would be close to the player and within their AOI.

In the real world, sight and sound are transmitted through the mediums of light and air respectively. These mediums do not propagate information at the same rate, let alone instantaneously; sound is transmitted at several orders of magnitude slower than light. Yet they are still relatively quick to travel through their respective mediums and, due to humans' limited sense capabilities, they are perceived as being instantaneous and synchronized. In order to achieve perceivably smooth game play, the propagation of update messages must similarly be propagated in order to 'beat' human senses. So the most important aspect in disseminating update messages is to do so with minimum latency.

Update messages' travel medium is the physical network that connects hosts to each other. When considering hosts that are globally distributed, the primary network to consider is the Internet, which is not only nebulous and heterogeneous, but cannot be managed or reserved in order to achieve a specific quality of service (QoS). While the advantages P2P systems provide over client-server have already been touched upon, and will be expanded further below, achieving acceptable QoS is always an issue. In addition to this, peers in P2P systems are also often heterogeneous, and do not necessarily provide the same resources as each other.

It is the goal of this work to route update messages in a location-aware fashion to achieve minimum latency. Location-awareness, with respect to virtual environment locality, stems from the AOI concept deeming that the bulk of a player's update messages are destined to a player's immediate locale. In order to support location-awareness, the design of the P2P system must be such that construction of the overlay network can accommodate player movement, so that its topology consistently reflects the game environment. Secondary to latency is efficient use of peer resources, due to lack of dedicated servers and peer

heterogeneity. Bandwidth and computational cycles are to be conserved, but not at the expense of timely dissemination of update messages.

2.2 Peer-To-Peer Networking Overlays

As mentioned earlier, P2P networking is a distributed networking paradigm where participants contribute their local resources to the larger system making them equal peers rather than clients. Due to the equality of the participants within the paradigm, fully P2P systems do not have any form of central or authoritative server, which typically manages all inter-peer communication in the client-server model. Consequently, P2P networks are characterized by the fact that peers must manage their own communication to a greater degree. P2P overlay networks are named thusly as they are a network of logical links between peers over an underlying network. An overlay link between two peers corresponds to a path in the underlying network that both peers are connected to.

Unstructured P2P network overlays have links made between peers in an arbitrary fashion; a peer can create a link to any other peer it gleans awareness of. Structured P2P overlays on the other hand, have peers follow a globally standardized protocol for establishing and terminating peer links. This makes structured overlays able to route efficiently, and often deterministically, to a given peer based on some form of identifier. Just as hosts on the Internet are identified by their IP address, so do P2P network overlays assign some form of identifier to participating peers.

P2P overlays are not an individual part of the classic networking protocol stack model; being implemented on the hosts connected to the underlying network classifies them to be part of the application layer in the protocol stack. Overlay networks often provide communication capabilities that are used by other applications. Generic P2P overlay networks abstract lower layer semantics by providing a generic API to the overlying application. As this layering effect is an extension of the networking protocol stack paradigm, overlays are often termed to be an application layer communication protocol in their own right. This is illustrated in Figure 1.

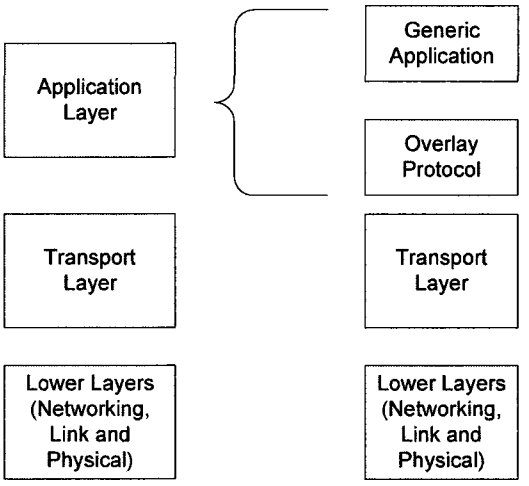


Figure 1: Traditional vs. Overlay Networking Protocol Stack

This thinking continues when applying P2P overlay networks to achieve distributed MMOG communications. The game application itself contains the engines for visual and auditory rendering, user interaction, as well as update message processing and generation. The specifics of disseminating these update messages is handled by the underlying overlay protocol, which provides an API suitable for MMOG applications. Messages are passed to and from peers in the overlay, while the actual transmission of the messages is done through the underlying network. The MMOG to overlay to network mapping is shown in Figure 2.

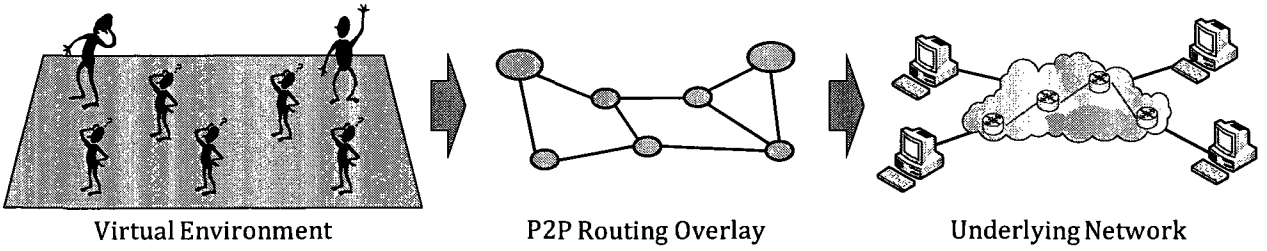


Figure 2: Mapping from Virtual Environment to Overlay to Underlying Network

Some terminology can be clarified with in this scenario. The word 'peer' has thus far been a catch-all term for a participant in the distributed networking paradigm. For each of the mapping layers in Figure 2 a specific term can be used to refer to the peer at that particular level. At the bottom, the term 'host' refers to the physical machine that is connected to the underlying network; hosts participating in the P2P overlay network are called 'nodes'. 'Player', as previously established, is the representation of the active person playing the

game at a physical host. In this system, players have a correlation with nodes, as the topology of overlay network reflects the game environment.

2.2.1 Properties

While alluded to in other sections, the general properties of P2P networking overlays which make them attractive for use are described below:

- Self-organizing – The handling procedures for arrivals and departures from the overlay network are run by each participating node. As the lower networking layers are abstracted to logical communication links, the nodes are able to organize themselves to form the network.
- Decentralized – This is a beneficial consequence of self-organization, as the network distributes previously centralized responsibilities to multiple peers. A pure P2P network can function without any centralized or coordinating entities whatsoever, thus entirely eliminating the issue of central points of failure existing in the system.
- Scalable – Scalability means that the capability of the system can grow with demand, theoretically allowing an infinite number of nodes to join. As the arriving clients' computing and network bandwidth resources are contributed to the overall pool, there should be no upper limit on the amount of users that can be handled.
- Dynamic – The peer network is able to adapt to nodes joining and leaving at random. Isolated peer failures are handled by the overlay peers informing one another of other peers in order to replace missing network links. Only a large number of simultaneous failures are able to overcome the self-repair mechanisms.

2.3 Pastry

Pastry [14] is a scalable, decentralized object location and routing peer-to-peer system. Pastry performs application-level routing and object location even in very large overlay networks, consisting of peers which are interconnected via an underlying network. Nodes participating in the Pastry overlay form a self-organizing and fault-tolerant network with deterministic object locations.

The idea of object location classifies Pastry as a form of distributed hash table (DHT), so named for its hash-table like lookup service. Where in the hash table data structure a lookup on a given key yields an associated value, a lookup on a given key in a DHT initiates the process of routing a message to the node associated with the key. As Pastry enforces a one-to-one mapping between key and node, the key becomes the node's overlay network identifier, or DHT-ID. DHTs are a well known class of structured P2P overlays.

Pastry is intended as a general substrate for the construction of P2P applications, and several applications have employed it as a foundation for their design. Examples of these applications are SplitStream [15], a multi-cast protocol for data distribution of high bandwidth data, SCRIBE [16], another multi-cast protocol for group communication and event notification, and PAST [17], a large scale distributed data storage architecture.

As a general application substrate for P2P communication, Pastry is designed to logically lie above the transport layer but under the other application protocols that use its simple API with hooks for message dispatch and reception. In routing messages from sender to destination, Pastry's employs hop-forwarding logic, where a node will forward a message to another node in the overlay that is closer to the message's final destination. This continues until the message reaches the destination node, which is illustrated in Figure 3. The application instances in the intermediate node do not need take action for this process to function, as it can be handled entirely by the underlying Pastry protocol. Pastry's API however, does allow for the application to be notified in the event of message forwarding.

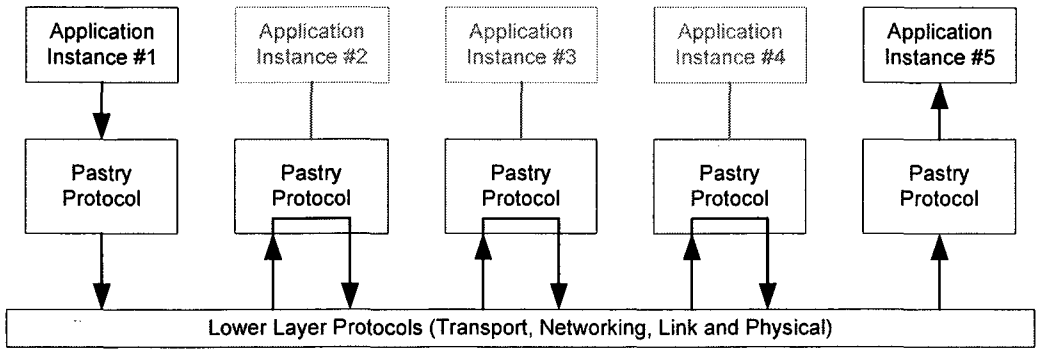


Figure 3: Illustration of Pastry Routing Though Networking Stack

Details of a traditional Pastry node's state, as well as algorithms for routing and handling churn, are presented forthwith.

2.3.1 Node State Structure

As mentioned, Pastry nodes require a DHT-ID in order to participate in the overlay network, which is traditionally a 128 bit one-dimensional identifier from a circular DHT-ID space. Generally these DHT-IDs are generated by cryptographically hashing the node's IP address or public key. This leads to a uniform random distribution of DHT-IDs, with no specific ordering or grouping, among nodes that can be diverse in location, capabilities, connectivity and other characteristics.

The core of a Pastry node state are its local state tables, which consists of a leaf set, neighbourhood set and routing table, all of whose entries are DHT-ID and network address pairs. The constructions of these local state tables are determined by the configuration parameters b, L, and M. The parameter b determines, amongst other things, the representation of the DHT-ID used in routing. The 128 bit value is converted to digits, where the number of digits is 2^b-1. The shared prefix between two DHT-IDs is the number of upper significant digits the DHT-IDs have in common with each other.

The leaf set contains the set of nodes within the overlay that have DHT-IDs numerically adjacent to the local node's DHT-ID. The leaf set is further split into two inner groups, with $\lfloor L \div 2 \rfloor$ nodes being larger DHT-IDs and $\lfloor L \div 2 \rfloor$ nodes being smaller DHT-IDs. The neighbourhood set is an undivided set of $|M|$ nodes, which are the closest to the present node with respect to a scalar proximity metric. This scalar proximity metric is traditionally a measure of network proximity, such as the number of IP hops or the round-trip time (RTT) between two nodes.

The routing table is structured to have $\lceil \log_{2^b} n \rceil$ rows and 2^b-1 columns. The entries at row n in the routing table refer to nodes whose DHT-IDs have a shared prefix length of n with the local DHT-ID, but differ at the $n+1^{\text{th}}$ digit; the column of the entry is determined by the value of the $n+1^{\text{th}}$ digit. If an entry within the routing table is empty, and the local node is unaware of an active node with a suitable DHT-ID for the entry it is left empty, though the uniform distribution of DHT-IDs ensures an even population of the routing table.

Selection of the b parameter allows for trade-off between the total number of entries in the routing table, which is approximately $\lceil \log_{2^b} N \rceil \times (2^b - 1)$ entries for an N node network, and the maximum number of hops between two nodes. For example, with a value of b=4 and N=10^6 nodes, routing tables will contain approximately 75 nodes on average and the network will have an expected hop count of 5. With N=10^9 nodes, routing table sizes increase to 105 and the expected hop count increases to 7.

Typical values for the configuration parameters b is 4, with L and M both equal to 2^b or 2×2^b. A sample local state table set is shown in Figure 4, without the associated network addresses shown. The associated parameters values are b=2, L=8 and M=8.

DHT-ID 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
1023301	10233000	10233230	10233232

Routing Table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighbourhood Set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Figure 4: Pastry Node State Tables (Associated Network Addresses Not Shown) [14]

2.3.2 Routing

As stated, Pastry employs hop-forwarding logic in order to route messages to their intended destination DHT-ID. Each node that receives message attempts to forward it to a node with a DHT-ID that has a shared prefix length with the destination DHT-ID greater than the current node's shared prefix length with the destination DHT-ID, illustrated in

Figure 5. Message forwarding continues until the message reaches an active node with either the exact destination DHT-ID, or the DHT-ID numerically closest to the destination DHT-ID. This latter case happens when there is not an active node with the exact destination DHT-ID.

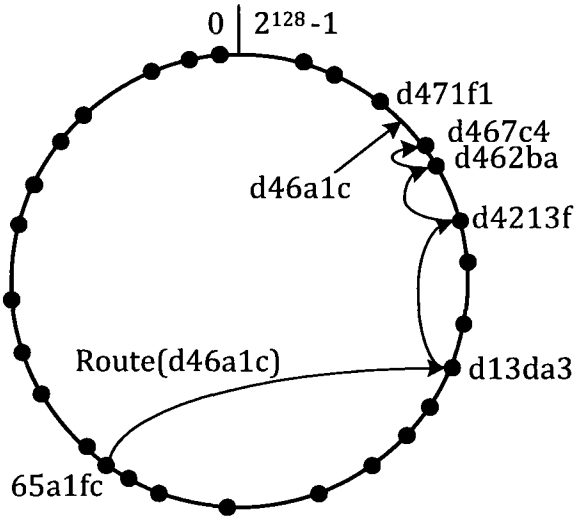


Figure 5: Example of Pastry Routing [18]

Pastry's routing algorithm, given below Table 1, is executed for every message where the direct network address of the intended recipient is unknown. Given a destination DHT-ID, the algorithm determines the node to which the message should be sent to in its next hop.

Table 1: Definitions for Pastry's Routing Algorithm

$R_l^{D_l}$	entry in routing table R at column i and row l, such that $0 \leq i \leq 2^b$, $0 \leq l \leq \lfloor 128 \div b \rfloor$
L_i	i^{th} closest DHT-ID within the leaf set L, such that $-\lfloor \lvert L \rvert \div 2 \rfloor \leq i \leq +\lfloor \lvert L \rvert \div 2 \rfloor$ negative/positive indices indicate DHT-IDs smaller/larger that the local DHT-ID
D	the *DHT-ID* destination of the message to be routed
D_l	the value of the l^{th} digit in the destination DHT-ID D
shl(A, B)	length of shared prefix between DHT-IDs A and B (in digits)

Algorithm 1: Pseudo-code for Pastry's Routing Algorithm

```
1:  if L_{-⌊|L|÷2⌋} ≤ D ≤ L_{+⌊|L|÷2⌋} then
2:      //use leaf set
3:      calculate L_i such that |D−L_i| is minimal
4:      accept locally if L_i = currentPeer, otherwise forward to L_i
5:  else
6:      //use routing table
7:      l ← shl(D, A)
8:      if R_l^{D_l} ≠ null then
9:          Forward to R_l^{D_l}
10:     else
11:         //rare case
12:         Forward to T ∈ L ∪ R ∪ M such that
13:             shl(T, D) ≥ l,
14:             |T−D| < |A−D|
15:     end if
16: end if
```

The routing algorithm determines the next node in the message's path through three main branches. The first branch, lines 2-5, checks the leaf set to see if the destination DHT-ID falls within the range of leaf set contained DHT-IDs. If the destination falls within range, the numerically closest DHT-ID from the leaf set is chosen as the next hop. If the chosen DHT-ID from the leaf set is that of the current local node, the message is accepted locally and considered to be delivered.

If the message's destination DHT-ID falls outside the leaf set, the routing table is the state table checked in the next branch, lines 6-10. The next hop is determined to be the routing table entry at the row specified by the shared prefix length between the local and destination DHT-ID, and the column determined by the next digit in the destination DHT-ID. If the routing table entry is empty, or the chosen node is unreachable, the third branch of rare case routing is considered.

In the rare case, lines 11-15, entries from all three local state tables are collected and considered. The entry that has the greatest shared prefix length with the destination DHT-ID, if it exceeds the prefix length shared by the destination DHT-ID and the local DHT-ID and is numerically closer to the destination compared to local DHT-ID, is chosen as the next hop.

A great advantage of this routing algorithm is the convergence of routing. Given an overlay network, two messages from two different nodes addressed to the same destination will always be routed to the same endpoint. An active node with the destination DHT-ID, or the node with the numerically closest DHT-ID should the former not exist, will receive both messages.

It should also be noted that when the neighbourhood set is used by the routing algorithm, it is in the rare case that both the leaf set and routing table were unable to provide a node with a DHT-ID closer to the destination, compared to the DHT-ID of the local node, for the next hop. This means that while individual DHT-IDs within the neighbourhood set are not of great consequence, the distribution those of DHT-IDs should not be overly clustered.

2.3.3 Churn Handling

Within the context of P2P routing overlays, churn is the phenomenon of nodes joining and departing the network. Pastry includes procedures to automatically handle node churn, thereby making the network both self-organizing and self-adapting.

2.3.3.1 Node Joining

Pastry's standard node arrival protocol is for the joining node to send a "join request" message, addressed to DHT-ID 'X', to joined node 'A'. 'A' forwards the join request according to the aforementioned routing algorithm, and will reach node 'B' whose DHT-ID is numerically closer to the DHT-ID 'X'. The message eventually reaches 'Z' who cannot forward along due to one of two reasons: 1) 'Z''s DHT-ID is equivalent to 'X'; 2) 'Z''s DHT-ID is the closest to the destination 'X' in the message, without equalling it. Should the reason be the former, 'X''s join request is denied and must restart the join process with another DHT-ID. In the latter case, the join request is accepted, and the joining becomes part of the overlay with DHT-ID 'X'.

Each node that forwards the join message also sends their state tables directly to the joining node. This received state table information is used to initialize the local state tables of the joining node. Mutual node awareness is achieved after the joining node's join request is granted, as it then sends its state table information to all the nodes within its tables; these nodes update their local state tables accordingly.

Pastry's design handles concurrent node arrivals optimistically, assuming that this issue is a rare case for two reasons. Firstly, the number of arriving nodes at a given time is relatively small compared to the number of nodes that form the overlay; secondly, the probability of requests for the same DHT-ID is low, due to the collision resistance properties of the cryptographic hashing employed for DHT-ID generation. If two nodes do happen to join the overlay concurrently and request the same DHT-ID, they must handle the contention themselves through the use of time stamped state updates, with one node finally leaving and rejoining the network with a different DHT-ID.

2.3.3.2 *Node Departure*

Pastry's design is such that nodes are generally expected to be persistent in the network, and thus expects all nodes departing from the overlay network to do so due to faults that lay elsewhere. As a result, graceful and ungraceful departures from the overlay are not distinguished from each other and are handled in the same manner. Detection of node departure, a responsibility that must be fulfilled locally by each node, is done through either keep-alive messages or failure to communicate with neighbouring nodes. In the event a departed or failed node is detected, the procedure for repair depends on the state table that contains the failed node entry.

If the departed node exists within the leaf set, the local node contacts the overlay peer with the largest DHT-ID in the leaf set within the same inner group as the failed node. It requests that peer for its leaf set, and updates its own leaf set accordingly. If the failed node exists within the routing table at entry R_l^d, the local node contacts the routing table entry R_l^i, where $i \neq d$, and requests that node's entry for R_l^d. If no valid entries can be provided, the value of l is increased by 1 and another node is contacted until a replacement is found. In order to repair the neighbourhood set, the neighbourhood sets for the nodes within the local neighbourhood set are requested, and a suitable replacement is determined.

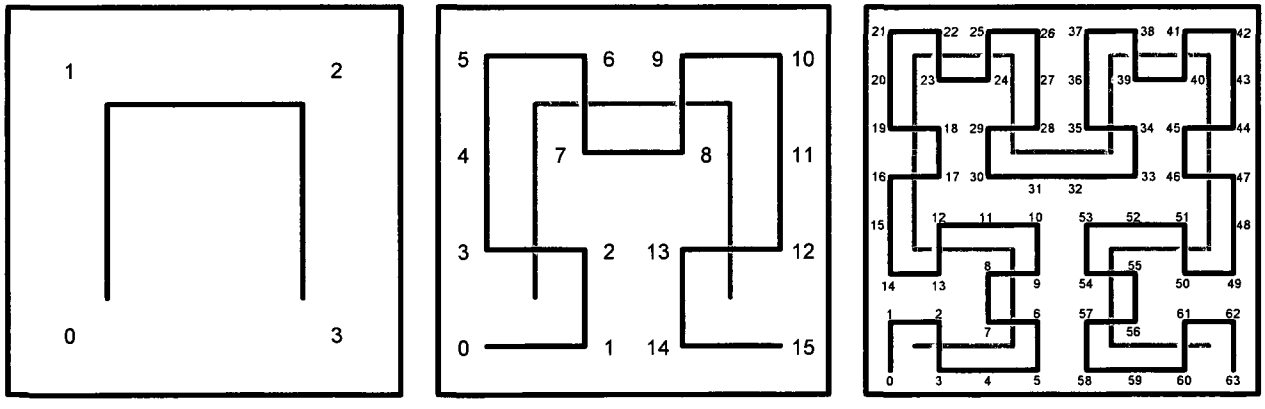
2.3.4 Applicability for MMOGs

Though Pastry is designed as a general routing substrate for P2P applications, it is obvious that the traditional design is not inherently suitable for use in distributed MMOG communications. This is evidenced by the applications developed on Pastry, where the

focus is placed on long term persistent storage and/or dissemination of data in a relatively static network. Nodes are expected to persist online and only depart due to underlying network failures, an assumption supported by a suggested 20 minute keep-alive message interval [18]. The lag between node departure and its subsequent detection is of course dependent on this interval. In terms of location aware routing, Pastry's inability to natively change a node's DHT-ID means that location based generation of DHT-IDs is bothersome, as players in MMOGs are expected to change location. It follows that the traditional Pastry design must be adapted in several ways in order for the protocol to be used in distributed MMOG communications.

2.4 Hilbert Curve

Space-filling curves are continuous and symmetrical geometric fractals that pass close to every point in the space they occupy. The Hilbert Curve is a space-filling curve built from individual components called cups. The first order curve consists of a single cup, and subsequent orders consist of cups that are rotated and translated, as illustrated in Figure 6.



(a) 1st Order Curve **(b) 2nd Order Curve** **(c) 3rd Order Curve**

Figure 6: Progression of 2D Hilbert Curve with Vertex IDs

The illustrations presented in this work are often limited to the two dimensional paradigm for the sake of clarity. The concepts of two dimensional Hilbert Curves readily apply to the three dimensional scenario however, which is of use in the three dimensional environments often used in MMOGs. Figure 7 shows a base three dimensional Hilbert cup, with Figure 8 being it and the two subsequent curve orders superimposed on the same cubic space.

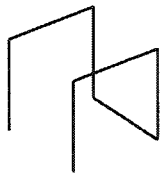


Figure 7: Base 3D Hilbert Cup

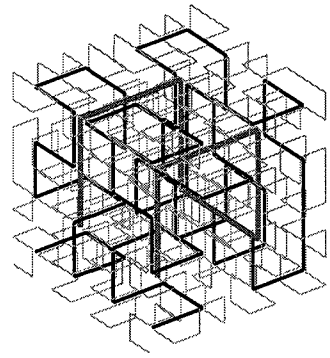


Figure 8: 3D Hilbert Curve of Orders 1 to 3

The Hilbert Curve, as with other space-filling curves, is often used to map multidimensional spaces to a single dimensional space, due to the fact that it passes close to every point in space. The single dimensional space is the sequence of index numbers assigned to vertices according to their sequence on the curve. Mapping the multiple dimensions to the single dimension, or linearization, is achieved by determining the index number of the vertex (vertex ID) on the curve closest to the multidimensional coordinates being mapped. Equation 2.1 states that the number of vertices V, on a given curve is dependent on the curve order K and the number of dimensions D.

$$V = 2^{K \times D} \qquad (2.1)$$

Of the many variations of space-filling curves, the Hilbert Curve has been proven to have the best locality preserving properties [19]. This means that the mapping of two points in space to the Hilbert Curve will yield vertex IDs that approximate the spatial relationship that exists in the multiple dimensions. In addition to this, examining an area over a two dimensional curve, or volume over three dimensions, will yield fewer index range fragments, which occur due to the curve exiting and re-entering the area or volume of interest.

Referring back to Figure 6, the fractal nature of the curve can be visually confirmed, as the K^{th} order curve is used as subcomponents in order to construct the $(K+1)^{\text{th}}$ order curve; this indicates a hierarchical pattern between Hilbert Curves of different orders. In addition

to this, vertex IDs are also based on the hierarchal relationship. Specific properties regarding this hierarchy are enumerated:

Property 1) Each cup in K^{th} order curve will generate 2^D cups in $(K+1)^{th}$ order curve, where D is the number of curve dimensions. Each generated cup is centred on a different vertex of the K^{th} order cup. The overall pattern of the $(K+1)^{th}$ order cups, relative to the K^{th} order cup, resembles a 2^{nd} order Hilbert Curve. The 2^{nd} order curve generated from the 1^{st} order is the base case of this property.

Property 2) A Hilbert Curve of order $K>1$ can be considered a combination of one or more 2^{nd} order Hilbert Curves which have been scaled, rotated and translated. This extends the first property, in that each of these 2^{nd} order "curve components" in the $(K+1)^{th}$ order curve can be generated from a single corresponding cup in the K^{th} order curve. As long as the geometric transformation factors are taken into account, the relatively simple 1^{st}-to-2^{nd} order curve transformation can be applied to any cup, in any order curve, to generate cups in the next order.

Property 3) Vertex IDs in the $(K+1)^{th}$ order curve are base 2 hierarchical from the K^{th} order curve. This signifies that vertex IDs can also be calculated without the need for generating the entire curve. The most significant bits (MSBs) of a cup's IDs are the bits of the vertex ID that the cup is centred about from the previous order. For example, in a three dimensional curve, the vertex with ID 5 {101} in the K^{th} order will lead to IDs 40 {101000} through to 47{101111}, in the $(K+1)^{th}$ order.

2.4.1 Tree Representation and Generation

Early works for space-filling curve generation include diagrammatic [20] and byte-oriented [21] approaches. Later efforts by researchers were made in generating lower order curves through recursive algorithms [22]. While these algorithms can be expressed in simpler and more efficient non-recursive forms, the issue lies in doing so for the generation of the required higher order curves. A table driven framework for generating space-filling curves is presented in [22], but the underlying generation algorithms still rely on recursive methods. An apparent middle ground to all these approaches is taken by Lawder in [23], where the Hilbert Curve is applied to the storage and retrieval of multi-dimensional data.

Lawder's representation is based on conceptualizing a curve of order K as a tree structure with K levels, and embodies the Hilbert Curve properties enumerated in the previous section. The ascending orders of the curve equate to the descending levels of the tree. The tree is comprised of nodes with each node comprised of 2^D sub-spaces, where D is the number of dimensions. Nodes and node sub-spaces are equated to first-order cups and cup vertices respectively. Each vertex sub-space contains that vertex's ID in binary and the coordinates at which it is located. As a single cup's dimensional length is a single unit, 0 to 1, vertex coordinates in a node sub-space are stored as binary sets with the size of the set equating to the number of dimensions. Each node in the tree has its sub-spaces ordered according to vertex ID, but the associated coordinates are not paired in an identical pattern for each node. This is because the various orientations of Hilbert cups are achieved by the different permutations of vertex ID/coordinate pairs. Figure 9 illustrates a two dimensional Hilbert Curve of the third order in the tree representation.

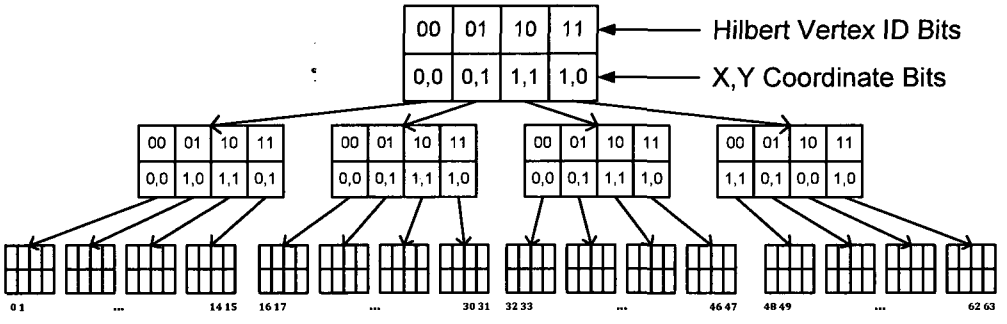


Figure 9: 2D Hilbert Curve Tree Representation

As can be seen from the figure above, it is the vertex sub-space that leads to the node in the tree at the lower level. As the orientation of the cup of the next higher order depends on the orientation of the current cup and the vertex of interest in the current order, the tree representation also follows in that the current node's pairing permutation, along with the sub-space followed, determines the permutation of the node in the next level lower. Hence if specific node permutations are identified as states, given inputs of the current state and sub-space, a state machine can be constructed to determine the next state as necessary. This is advantageous as it is prohibitive for an entire tree of higher orders to be stored in its entirety. The state machine for a three dimensional Hilbert Curve is given below in Table 2.

Table 2: 3D Hilbert Curve Tree Representation State Machine

State	Next State and <X,Y,Z> Vertex Coordinates for each Vertex ID in State							
	0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)	6 (110)	7 (111)
0	1 <0,1,0>	2 <0,1,1>	2 <1,1,1>	3 <1,1,0>	3 <1,0,0>	4 <1,0,1>	4 <0,0,1>	5 <0,0,0>
1	2 <0,1,0>	0 <1,1,0>	0 <1,0,0>	6 <0,0,0>	6 <0,0,1>	7 <1,0,1>	7 <1,1,1>	8 <0,1,1>
2	0 <0,1,0>	1 <0,0,0>	1 <0,0,1>	9 <0,1,1>	9 <1,1,1>	10 <1,0,1>	10 <1,0,0>	11 <1,1,0>
3	10 <1,1,1>	8 <1,1,0>	8 <0,1,0>	0 <0,1,1>	0 <0,0,1>	9 <0,0,0>	9 <1,0,0>	6 <1,0,1>
4	11 <1,0,0>	6 <1,1,0>	6 <1,1,1>	8 <1,0,1>	8 <0,0,1>	5 <0,1,1>	5 <0,1,0>	0 <0,0,0>
5	9 <0,0,1>	7 <1,0,1>	7 <1,1,1>	10 <0,1,1>	10 <0,1,0>	0 <1,1,0>	0 <1,0,0>	4 <0,0,0>
6	4 <1,0,0>	11 <0,0,0>	11 <0,1,0>	1 <1,1,0>	1 <1,1,1>	3 <0,1,1>	3 <0,0,1>	9 <1,0,1>
7	5 <0,0,1>	9 <0,0,0>	9 <1,0,0>	11 <1,0,1>	11 <1,1,1>	8 <1,1,0>	8 <0,1,0>	1 <0,1,1>
8	3 <1,1,1>	10 <1,0,1>	10 <1,0,0>	4 <1,1,0>	4 <0,1,0>	1 <0,0,0>	1 <0,0,1>	7 <0,1,1>
9	7 <0,0,1>	5 <0,1,1>	5 <0,1,0>	2 <0,0,0>	2 <1,0,0>	6 <1,1,0>	6 <1,1,1>	3 <1,0,1>
10	8 <1,1,1>	3 <0,1,1>	3 <0,0,1>	5 <1,0,1>	5 <1,0,0>	11 <0,0,0>	11 <0,1,0>	2 <1,1,0>
11	6 <1,0,0>	4 <1,0,1>	4 <0,0,1>	7 <0,0,0>	7 <0,1,0>	2 <0,1,1>	2 <1,1,1>	10 <1,1,0>

If the tree representation of a given order Hilbert Curve is traversed from its root node to a leaf node, and a specific sub-space of the leaf node is chosen, it is possible to determine two items of information regarding the vertex represented by the chosen sub-space. Following Hilbert Curve property 3, the vertex ID can be determined by the concatenation of the vertex ID bits of the sub-spaces in the tree traversed to reach the current one, with the root node's bits as the most significant. The second item is the dimensional coordinates for the vertex, which can similarly be determined by the concatenation of the coordinate bits of the traversed sub-spaces. This hierarchical property of vertex coordinate is similar to that of

vertex IDs, but is constrained to representations that enforce an integer based coordinate system.

Chapter 3 - Related Work

There has been much research done in recent years in the area of scalable MMOGs, in which P2P based virtual environments have been discussed as an option for their advantageous properties. These works contain varying approaches in applying the P2P concepts and technologies.

3.1 Client-Server Hybrid

Hybrid architectures attempt to combine the properties of client-server with P2P in order to cultivate the advantages from both, but do so in a manner that requires actual servers instead of assigning server-like responsibilities to peers. This hybridization means that this type of architecture has aspects that resemble the predominant client-server model. For this reason, the following works place focus on creating an API recognizable by contemporary game developers, as well as on integrating the system into first-person shooter style games.

Colyseus [24] is a distributed architecture whose object query interface is based on Mercury [25], a custom DHT that supports range queries. Players publish their environmental coordinates on the DHT and locate peers in their AOI to request updated information on objects by using the range-query. Objects, specifically players and mutable items, have an authoritative copy residing on a single node and replicas that can reside on multiple nodes. Colyseus also proposes the addition of a speculative pre-fetch to exploit locality to compensate for DHT delay while updates are propagated from primary copies to replicas directly. While Colyseus's goals are very close to this work's in some regards, community servers must be utilized to provide dedicated computing power to the overall system, in order to obtain the efficiency and overall scalability touted.

Chan et al. present Hydra [26], a client-server/P2P hybrid where each node has both client and server functionality, but assumes only one of the roles. Client nodes are comprised of a game client and client proxy. The game client is oblivious to the communication specifics

and simply reads/writes to the "network" through the client proxy; the proxy is aware of the underlying communication architecture. Server nodes hold "slices" or regions of the MMOG environment; the primary slice on a server is reflected in a backup slice on another server node. When sending updates to a server with the primary slice, client proxies will duplicate the update to the backup slice; this is in case the server node with the primary slice fails. While the distribution of server load is split among several nodes, which solves the central point of failure issue, scalability is still limited within this architecture. It is up to the servers to store, process and disseminate updates to the clients and each other; the last is required to keep the backup slice fully up to date.

While valid for established MMOGs that wish to apply P2P concepts without a drastic changeover in paradigm, the client-server hybrid approach still requires dedicated server resources to accomplish. Thus while decentralization is achieved to a degree, true scalability is not. In circumstances where resources are scarce, such as non-profit/academic research, an alternative and more distributed approach is favoured.

3.2 Environment Zoning with Super-Nodes

In this approach the MMOG environment is logically split into defined and non-overlapping zones/regions. This is a highly popular approach as player AOI can be quantized to one or more zones. This simplifies the logistics in dealing with AOI and game updates. Zone management is accomplished by separate resources employing the same architectural scheme. Each zone has a small subset of "super-nodes": peers to which server-like responsibilities are assigned. A single zone owner super-node is customary, but may be accompanied by active backup node(s).

Limura et al.'s Zoned Federation of Games Servers [27] is an approach that applies DHTs and environment zoning to MMOGs. As DHT based communication is said to incur higher latency due to increased message hop count, the DHT layer is not used for bulk node communication. It is used instead for backup game state storage and initial "rendezvous". The zone owner keeps direct connections to all interested parties, and updates those parties when zone data is requested. While the zone owner keeps the zone data in its own local cache, the same data is also backed up to another node in the DHT. This backup node

also holds the network address of the zone owner. When a zone member wishes to get zone data, the DHT backup node is queried and the zone owner is then contacted directly.

Knutsson et al. describe SimMUD [28], a peer-to-peer approach for MMOGs that uses Pastry [14] and SCRIBE [16] for game state handling and communication respectively. Game states are classified into 3 categories: player state, object state and the map (which is considered preloaded and effectively static). Player and object state information are aggregated by region, each of which is assigned an ID. The active overlay node with the numerically closest DHT-ID to the region ID is deemed the region coordinator. This overlay node handles updates to that region's information, and the next numerically closest node(s) is/are the data backup. The super-nodes chosen are not necessarily within the given region. Relevant player and object data is multicast to all nodes within a region with Scribe, which uses the underlying Pastry network; the region coordinator acts as the root of the Scribe multicast tree.

Hampel et al. [29] propose another Pastry-based MMOG (PastryMMOG), a similar idea to SimMUD that uses Pastry [14] as the base network overlay and SCRIBE [16] multicast for update dissemination, but also includes PAST [17] for object management. The purpose of this is to address other game design aspects, such as object and resource storage, as well as cheating hindrance. Again the environment is split into regions, but the assigned controller for each zone changes every game tick to prevent cheating.

The issue with these zoned architectures is the assignment of mass responsibilities to the zones' super-nodes, as there is the potential that crowding within in a zone can overwhelm the super-nodes' resources. This centralization also leads to dependence on the zone owner, which incurs additional overhead for repair should the owner disconnect ungracefully; in the case of PastryMMOG, the owner switching overhead is a constant. Moreover, the random nature of zone owner selection does not consider any form of proximity between the owner and the members interested in the zone. Updates may subsequently be subject to overall higher latency due to this.

3.3 Unstructured Overlay Networks

In the zoned environment works discussed above, the P2P overlay networks used are primarily DHTs, a type of structured network. DHTs are often deemed unsuitable for use in virtual environments as their randomized DHT-ID generation does not take locality into account. Unstructured networks on the other hand can be used to form an overlay that is based on locality as they allow connections to be formed between nodes in a less formalized manner. This is the approach taken by Hu et al. with VON [30], which creates connections between nodes based on the Voronoi diagram, a mathematical construct which partitions the game environment depending on node location. Each node maintains connections with other nodes in the same area according to a relationship between their AOI and the partitions of the environment. As the overhead in maintaining appropriate inter-node connections is unregulated and depends on the number of nodes in an area, highly concentrated populations can result in unmanageable scenarios with regards to message exchange.

3.4 Peer-to-Peer Hybrid

Another variation on the hybridization concept is to combine P2P overlay network of varying natures. Yu et al.'s MOPAR [31] attempts to combine DHT and unstructured P2P networks, along with environment zoning to quantize player AOIs. Each zone is associated with a home node, a node whose DHT-ID is numerically closest to the zone's hash generated ID. This home node is used for master node (zone owner) tracking and registration. The master node forms an unstructured network with the other nodes that have an interest in the particular zone, directly feeding them update data. Despite the hybridization of P2P systems, the super-node concept is reintroduced with its issue of centralization and single point of failure. In addition to this, slave nodes' resources remain unexploited as they are connected to the master nodes, and not each other.

Chapter 4 - Proposed System

4.1 Requirements

As a summary of the MMOG communication requirements discussed in section 2.1, this system must address the following:

- Location-aware routing of update messages in order to achieve minimal latency of game update delivery.
- Construction and maintenance of the overlay network in a manner that the topology reflects the game environment.
- Delivery of updates messages to players' AOI.
- Minimization of network and computational load, but not at the expense of timely delivery.

4.2 System Overview

This proposed system aims at building a structured overlay on the top of the Internet in order to allow MMOG users to exchange their updates efficiently, while trying to get the highest possible QoS. An existing P2P DHT network, Pastry [14], is adapted with the application of the Hilbert Space-filling Curve in order to achieve locality awareness. Use of the DHT provides the advantages of structured networks, but the traditional use of cryptographic hash based DHT-ID generation is forgone in favour of location-aware DHT-ID generation. These location-aware DHT-IDs are obtained by mapping three dimensional locations onto the Hilbert Curve. The classic symmetrical Hilbert Curve is modified with non-uniformity to better fit the distribution of players and objects in MMOG environments.

In order to ensure that the topology consistently reflects player location, player movement is handled by having nodes re-join the overlay with their updated positions. The overhead of re-joining is reduced by adapting the join protocols, and consistency is maintained with additional mechanisms.

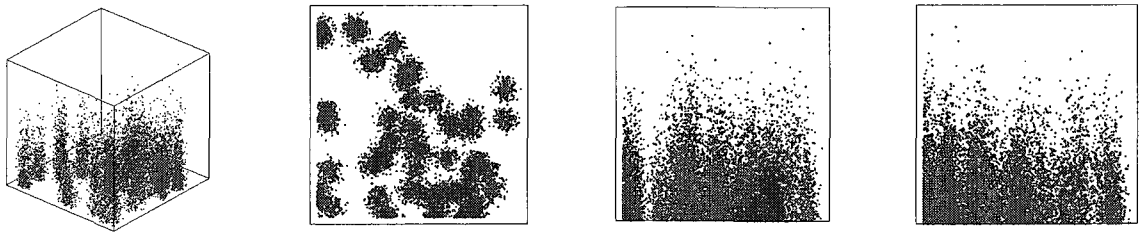
Use of the Hilbert Curve provides a touchstone upon which knowledge of potential nodes within an area can be determined, which allows the system to support AOI updates. The potential DHT-IDs are calculated by a range query algorithm optimized for memory usage and computational load. Coupling the range query algorithm with peer discovery and update dissemination processes achieves AOI game updates to interested parties with 1-hop latencies.

The following sections provide in-depth design detail for these three aspects of the system. In addition to this, a discussion on the selection of the underlying transport protocol is given.

4.3 Location Aware DHT-ID Assignment

Location aware DHT-ID assignment is realized by using a space-filling curve for linearization of the game environment's three dimensions. The Hilbert Curve is the selected space filling curve due to the fact that its locality preserving properties are the best for all space filling curves [19]. This means that two points with proximity in three dimensions will have DHT-IDs that are also 'close'. The definition of close in the single dimensional space does not specifically mean that the difference between the two values is small, but also that the order of magnitude of the difference is small. The definition of close is expanded in this manner due to the loss of granularity when mapping multiple dimensions to one. As routing in Pastry is based on the DHT-IDs, which now reflect locality, use of Hilbert based DHT-IDs achieves location-aware routing.

Mapping collisions often occur with lower order curves as nearby points are mapped to the same vertex on the Hilbert Curve, resulting in identical DHT-IDs. Increasing the curve order results in a higher vertex density; however a greater number of bits to represent IDs are required and the associated computational load is also increased. Use of a standard uniform Hilbert Curve seems logical when users are distributed evenly throughout game world. In real scenarios where the Z-axis stretches skywards however, objects are distributed in clusters all over the XY plane. The non-uniformity case is much more severe over the XZ and YZ planes, as the majority of objects are located close to XY plane. Figure 10 demonstrates an example user distribution in a city-style game world.

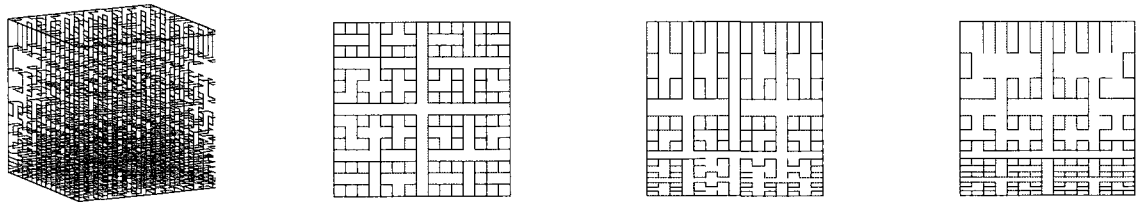


(a) 3D (b) XY plane (c) XZ plane (d) YZ plane

Figure 10: MMOG Player Distribution

In this case, object location mapping to a standard uniform Hilbert Curve will result in the waste of a large portion of the Hilbert ID space. While a Hilbert Curve order should be chosen to provide sufficient density to cover the concentrated regions near the XY plane, the same curve density is practically useless high in the Z direction where there are few points to be mapped.

This system introduces a non-uniformly distributed curve that can achieve increased vertex density where it is required. Figure 11 demonstrates the proposed non-uniform Hilbert Curve. The resulting curve will be non-symmetrical, but will still have the same number of vertices and require the same number of DHT-ID bits, resulting in fewer mapping collisions.



(a) 3D (b) XY plane (c) XZ plane (d) YZ plane

Figure 11: Non-Uniform 4th Order Hilbert Curve

4.3.1 Fast Computation of the Hilbert Curve

As discussed, high order Hilbert Curves provide increased vertex density, allowing a massive number of player locations to be mapped with fewer collisions and minimal location/vertex differentials. The challenge lies in dynamically generating and mapping to this curve, as storing an entire high-order curve and traversing it linearly is infeasible. This

solution exploits the properties of the Hilbert Curve enumerated in 2.4, reiterated here for convenience, in order to achieve an iterative and fast dynamic mapping algorithm.

Property 1) Each cup in K^{th} order curve will generate 2^D cups in $(K+1)^{th}$ order curve, where D is the number of curve dimensions. Each generated cup is centred on a different vertex of the K^{th} order cup. The overall pattern of the $(K+1)^{th}$ order cups, relative to the K^{th} order cup, resembles a 2^{nd} order Hilbert Curve. The 2^{nd} order curve generated from the 1^{st} order is the base case of this property.

Property 2) A Hilbert Curve of order $K>1$ can be considered a combination of one or more 2^{nd} order Hilbert Curves which have been scaled, rotated and translated. This extends the first property, in that each of these 2^{nd} order "curve components" in the $(K+1)^{th}$ order curve can be generated from a single corresponding cup in the K^{th} order curve. As long as the geometric transformation factors are taken into account, the relatively simple 1^{st}-to-2^{nd} order curve transformation can be applied to any cup, in any order curve, to generate cups in the next order.

Property 3) Vertex IDs in the $(K+1)^{th}$ order curve are base 2 hierarchical from the K^{th} order curve. This signifies that vertex IDs can also be calculated without the need for generating the entire curve. The most significant bits (MSBs) of a cup's IDs are the bits of the vertex ID that the cup is centred about from the previous order. For example, in a three dimensional curve, the vertex with ID 5 {101} in the K^{th} order will lead to IDs 40 {101000} through to 47{101111}, in the $(K+1)^{th}$ order.

The following sections describe in steps the major components of the mapping algorithm, which begin with fast mapping to a standard Hilbert Curve.

4.3.1.1 Fast Geometric Mapping to the Standard Hilbert Curve:

The objective of the fast mapping algorithm is to find the vertex on the Hilbert Curve of a specified order that is closest to a given point, and to determine the ID of the final vertex. The basic idea is to only generate cups of interest instead of generating the whole curve.

During the process of mapping a point to the Hilbert Curve, finding the K^{th} order vertex closest to the point leads to generating the cup of interest in $(K+1)^{th}$ order curve that is

centred on the K^{th} order vertex. After starting with a predefined 1^{st} order curve, the algorithm iteratively goes through increasing orders. In each order only the single cup of interest that is closest to the point is generated. The final ID of the point mapping is calculated using the hierarchical property of the vertex IDs.

Previous work on Hilbert Curve generation put forward the use of lookup tables to store all possible cup to cup transformations [22]. The second property is exploited in order to avoid this, as generation of all cups of interest start with a single predefined base cup centred about the origin. Applied to this base cup are rotation, scaling and translation transformations in order to achieve the correct cup of interest in the $(K+1)^{th}$ order.

Rotational transformation on the base cup is two fold. The first rotation applies the 1^{st}-to-2^{nd} order transformation to the base cup as indicated by the vertex chosen in the K^{th} order. The second rotation changes the base cup to match the orientation of the K^{th} order cup. The scaling factor on the base cup is dependent on K, as cup sizes are halved through subsequent orders. The translation on the base cup is done so it is centred on the K^{th} order vertex, as stated in the first property. Applying these transformations together generates the cup of interest in the $(K+1)^{th}$ order, which is then used in the next iteration of the algorithm.

4.3.1.2 Scaling to the Virtual Environment

Traditional Hilbert Curves are symmetrical and typically centred about the origin. The dimensions of the space filled by the curve are dependant on the order of the curve and size of the first order cup, both of which are known values. For use in MMOGs, the curve will ideally be scaled to fill the entire environment space. In order to achieve this, modifications on the basic mapping algorithm are required. The first modification is to calculate a scaling factor for each dimension so that the final curve would entirely fill the space in that dimension. This scaling factor is applied when generating each cup of interest, in addition to the other geometric transformations.

The second modification is to translate the predefined 1^{st} order curve so that it is located in the middle of the environment space. This need only be applied once, as the translation

carries through when generating all subsequent cups of interest due to the fact that they are centred on the previous order vertex.

4.3.1.3 Achieving Non-uniform Distribution

The objective in non-uniform distribution is to increase the density of Hilbert Curve vertices in a specified area, deemed the "cluster centre". Densification is performed on a dimension of the curve as illustrated in Figure 12, where the cluster centre is at the 0 value of the x-axis. If non-uniform distribution is applied simultaneously to multiple dimensions, the effect is a "clustering target", shown in Figure 13.

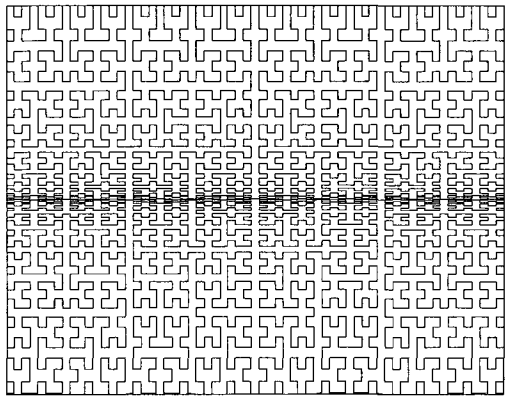


Figure 12: Single Dimension Non-Uniform Distribution

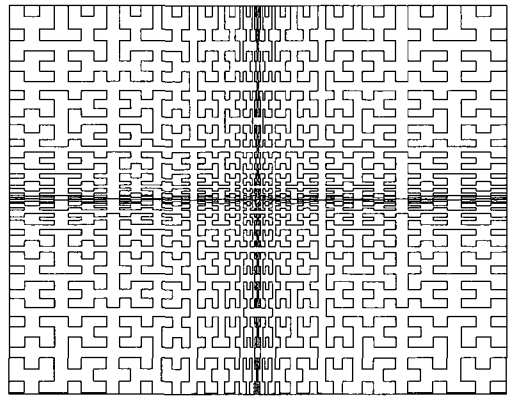


Figure 13: Multiple Dimension Non-Uniform Distribution

Since cups of interest are generated individually and iteratively, the ability to redistribute vertices must therefore rely on data that is either already known or has minimal overhead when calculated. In applying non-uniform distribution, each vertex on the cup is scaled individually on each dimension. Application of non-uniform distribution must be on the points of the $(K+1)^{th}$ order cup after all other aforementioned geometric transformations have been applied. As previously discussed, this includes the translation to centre the cup on the K^{th} order vertex. However, the K^{th} order vertex must be located at the position it was before non-uniform scaling was applied to it. This means that the K^{th} order cup must be copied before applying non-uniform distribution to it, so that the copy can be used in determining the $(K+1)^{th}$ order cup.

The scaling factor that is applied to vertices is dependant on the ratio between two distances. The first is the distance in the given dimension between the vertex location and the cluster centre. The second is the distance, again in the same dimension, between the clustering centre and upper boundary of non-uniform distribution. Any function of this ratio can then be used as the non-uniform scaling factor. The choice of a proper function depends on the knowledge regarding population distribution. For this system however, it is important to note that the ratio function employed requires a computationally achievable inverse function. This is due to the specifics of performing range queries over the curve, which are discussed in 4.5.1.

This system defaults to applying non-uniform distribution only on the Z-dimension, as this dimension is usually interpreted as reaching skywards in many contemporary game environments. Hence the lower and upper-boundaries of the non-uniform distribution range are the lowest and highest values of the Z-dimension, with the cluster centre near the lower boundary or "ground level" where the majority of objects and players are located. This produces results analogous to Figure 11. Other domains and applications of this non-uniform Hilbert Curve may require more complex distributions, and so this technique allows multiple regions to be defined, each with their own particular distribution, as in Figure 14.

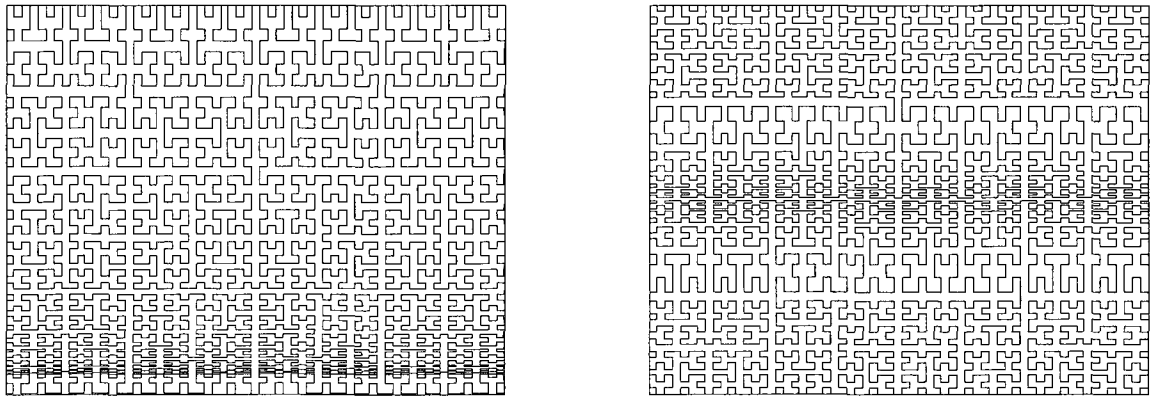


Figure 14: Multi-region Non-Uniform Distributions

The specific ratio function employed for non-uniform distribution in the evaluated proof-of-concept attempts to emulate the curvature of the normal distribution's probability

density function. A distribution based on a given mean and deviation is normalized against the standard normal distribution to determine scaling factor for each point. The inverse for this ratio function is achieved through the application of Newton's method of numerical analysis.

4.3.2 Hilbert Curve Mapping Algorithm Pseudo-Code

The components of the location-to-Hilbert mapping algorithm discussed thus far are combined into the mapping algorithm presented below. Notation relevant to the algorithm is first detailed in Table 3.

Table 3: Definitions for the Hilbert Curve Mapping Algorithm

P	the point in space to map to the Hilbert Curve
K	order of the Hilbert Curve being mapped to
baseCup	coordinate array of a 1st order Hilbert cup
scaleFactors	set of XYZ curve scaling factors
shiftFactors	set of XYZ curve translation factors
rotationMatrix	a homogeneous identity matrix
nuZScaling(*H*)	performs Z-axis non-uniform distribution on the coordinate array *H*
closePoint(*H*, *P*)	find the array index of the coordinate in *H* closest to the coordinate *P*
shift(*I*, *B*)	bitwise shift left the integer value *I* by *B* bits
calRotation(*I*, *M*)	calculate the combined rotation matrix *M* and the 1st-to-2nd order rotation denoted by *I*, a Hilbert first order cup vertex number
appRotation(*H*, *M*)	apply rotation *M* to coordinate array *H*

Algorithm 2: Pseudo-code for Mapping to the Non-Uniform Hilbert Curve

```
1:  DHT_ID ← 0
2:  currentOrder ← 1
3:  i ← 0
4:  for each coordinate_XYZ C in baseCup
5:      tCup[i] ← (C × scaleFactors) + shiftFactors
6:      i ← i + 1
7:  end for
8:  repeat
9:      saveCup ← tCup
10:     tCup ← nuZScaling(tCup)
11:     vertexIndex ← closePoint(tCup, P)
12:     DHT_ID ← shift(DHT_ID, 3) + vertexIndex
13:     if (currentOrder ≠ K)
14:         rotationMatrix ← calRotation(vertexIndex , rotationMatrix)
15:         rCup ← appRotation(baseCup, rotationMatrix)
16:         scaleFactors ← scaleFactors × 0.5
17:         i ← 0
18:         for each coordinate_XYZ C in rCup
19:             tCup[i] ← (C × scaleFactors) + saveCup[vertexIndex ]
20:             i ← i + 1
21:         end for
22:     end if
23:     currentOrder ← currentOrder + 1
24: until (currentOrder > K)
```

Lines 3-7 create the 1st order curve in the variable tCup. The loop between 8 and 23/24 is the main iterative component of the algorithm.

Lines 9-10 save the currently uniform tCup for future use, and then performs Z-axis scaling on it. Lines 11-12 find the vertex coordinate in tCup closest to the point and uses it to calculate the final ID. Lines 13-21 calculate the next order cup, if it is needed. First the cumulative rotation is calculated and applied to baseCup (lines 14-15), with a scale and a shift then performed on the newly rotated cup (lines 16-22).

To provide a rough estimate on the complexity of calculating DHT-IDs, this algorithm is of $O(K)$ complexity, with K being the order of the curve being mapped to. However, it should be noted that the active data set upon which calculations are performed is very small. An array of eight vertex coordinates, a single 3D Hilbert cup, is operated on instead of the entire curve of each given order.

4.4 Player Movement

Though location aware DHT-ID assignment can be accomplished, the system must now take into account the fact that players within the game will not remain stationary. This means that nodes within the overlay must continually ensure their assigned DHT-ID is updated to reflect their current position in the environment, resulting in an overlay topology that is fluid over time. The rather static network constructed by traditional Pastry now becomes just a snapshot in time of the MMOG network as game time progresses and the overlay adapts accordingly. The following subsections discuss the mechanisms developed to handle player movement.

4.4.1 DHT-ID Threshold

The first step in handling player movement is to create a DHT-ID changeover threshold which determines when players within the environment should initiate the DHT-ID change process. The threshold area is centred about the location that the player occupied when the current DHT-ID was issued. The logic applied is that those that remain within the radius of their original DHT-ID location are still well represented in the overlay; those that pass beyond the threshold are eligible to request a DHT-ID change. This is illustrated in Figure 15.

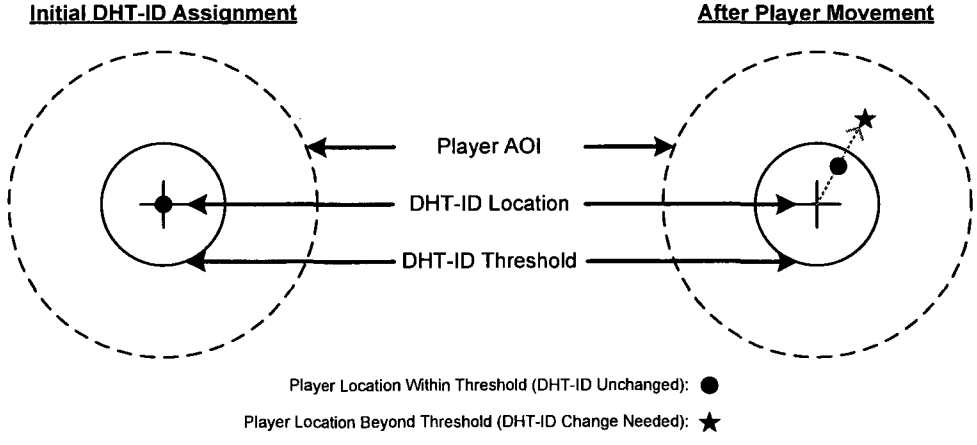


Figure 15: Visual Representation of Player AOI and DHT-ID Threshold

As player locations must be well approximated within the overlay by DHT-IDs, the defined threshold is cannot be overly large. Thus smaller values are preferred in order to achieve DHT-IDs that updates according to the player's movements. On the other hand however, if

the value of the threshold is overly small and players have larger displacements in small intervals of game time, the result is nodes continually changing their DHT-IDs. As an extreme case this is undesirable, because the peer connections within the overlay are in constant flux and the communication overhead may become unacceptable. As the value of the DHT-ID threshold is affected by various game specific parameters, such as player AOI, average player movement and game network throughput, the threshold value cannot explicitly be defined here for all MMOGs.

4.4.2 DHT-ID Reassignment

4.4.2.1 New Overlay Node Creation

As Pastry's join procedure is comprehensive in building the local state tables of a node for routing, reusing it for DHT-ID reassignment is a logical step. The base case for assigning a new DHT-ID to a game client is to create a completely new node in the overlay network, as shown in Figure 16. In a naive approach, there is no intersection between the new and old instances, and the new DHT-ID node is initialized without knowledge of the existing connections to other game clients. This procedure is required if no changes are made to the Pastry API, as traditional Pastry overlay nodes do not change their DHT-IDs.

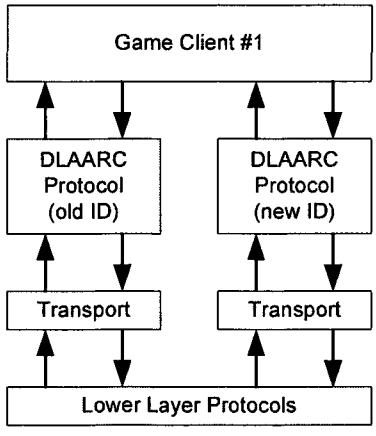


Figure 16: Basic Overlay Node Duplication

Naive creation of new overlay nodes for a single game instance is wasteful, as it does not consider the overhead by the underlying layers in creating peer communication links. Given the fact that DHT-IDs are meant to reflect the players' location as they move through the environment, it is possible for there to be little numerical difference between the old

and new DHT-IDs. In this case, as the new DHT-ID node receives state table information from other nodes during its join procedure, it is possible that it will form communication links with peers that are already peers of the old DHT-ID node. Regardless of the specific transport protocol chosen, which is discussed in section 4.6, creation of a network communication link requires some overhead.

For greater efficiency, the new DHT-ID node should assume command of the established links held by the old DHT-ID node if said links are required by the new node. In addition to this, if the communication links are application level due to use of a connectionless transport layer protocol, the old node can act as the new node's proxy to the network. In this case, the old node needs only to inspect incoming messages' application header for their intended destination and handover those addressed to the new node; thus the new node need not open its own port(s). While this handover of communication links reduces overhead incurred by the lower layer protocols however, the number of messages exchanged by the overlay protocol itself is not diminished.

4.4.2.2 Abbreviating the (Re) Join Procedure

In an attempt to reduce the overall number of messages required by a DHT-ID change, the idea of reusing prior links is revisited. Peer connections already established by the old DHT-ID overlay node may be useable for the node with the new DHT-ID. In fact, if the DHT-IDs differ only by a few lower significant bits, many of the peer connections in the upper rows of the routing table, and perhaps a subset of the leaf set, will be equally valid for both old and new overlay nodes. In addition to this, as both nodes are on the same physical host, the neighbourhood set remains completely useable. Thus the first step to abbreviating the join procedure is to have the new node merge the old node's state tables into its own upon initialization.

Recalling Pastry's standard node join protocol, one of the purposes of the initial join request message is to request state table information from the nodes forwarding the join request itself. This state table information received by the joining node is used to initialize its own, initially empty, state tables. The initial join request is first sent to a node is in close network proximity (e.g. low RTT) to the joining node. As traditional Pastry's DHT-ID

generation is hash-based, there is no guarantee of shared prefix between the DHT-IDs of the joining node and the node to first receive the join request. This is purposefully done in order to fill the upper levels of the joining node's routing table. These upper level entries are those with small, or no, shared prefix with the joining node's DHT-ID.

In the case of DHT-ID reassignment, this lengthy join request routing is unneeded as the new local state tables have taken the applicable entries from the old DHT-ID state tables. In fact, the greater the shared prefix length between the old and new DHT-IDs, the fewer the number of entries within the new state tables that will be empty. In order to fill these empty entries, a DHT-ID change request message still needs to be routed through the overlay, a message type semantically identical to the standard join request message. The first node to receive this message can now be much closer to the end destination, which reduces the number of hops and, by consequence, the amount of overhead the DHT-ID change incurs.

Determining the first hop of the DHT-ID change request is done by making use of the old DHT-ID node's state tables in the standard routing algorithm. This is due to the fact that the change request message is literally addressed to the new DHT-ID, similar to the standard join request message. As the new DHT-ID node has not yet joined the overlay, routing using the old DHT-ID node's state tables will yield a known node with the closest possible DHT-ID to that of the message destination. This skips as many intermediate hops as is possible. It should be noted that while the state tables of the new node can theoretically be used, this is avoided as the standard routing algorithm would recognize that the new local node is the destination of the change request. Doing so defeats the other purpose of the request, which is to discover if the requested DHT-ID is available and not currently in use. Thus leveraging the old node's state tables for routing avoids requiring a change in the existing routing algorithm.

It is possible that the DHT-ID change request is simply routed to the local overlay node with the old DHT-ID. This is again due to the possibility that the old and new DHT-IDs are numerically very close due to a small and well chosen DHT-ID threshold. In this case, network communication is eliminated altogether, making the abbreviated DHT-ID

reassignment very efficient. On the other hand, if the player's old and new DHT-IDs are based off of locations that map to completely different lower order Hilbert vertices, the shared prefix between old and new DHT-IDs may be non-existent. In this case, the base case performance experienced is expected to be equivalent to that of section 4.4.2.1, in terms of number of messages required to be sent.

4.4.3 Consistency Mechanisms

There are several consequences to adapting Pastry's design for use in MMOGs, specifically here as a result of the above movement handling protocol, and allowing hosts to change their DHT-IDs. These specific issues are a higher rate of state table entry invalidation, a paradigm shift in detecting said entry invalidation, and increased DHT-ID request contention. Thus, in order to maintain consistency across the peers in the network, a suite of consistency mechanisms are required to accompany the DHT-ID reassignment procedure.

4.4.3.1 Peer Notification and the Peer List State Table

As discussed in 2.3.3.2, Pastry's design takes into account that nodes may depart or fail from the network without warning. As graceful and ungraceful departures are not distinguished from each other however, the window of opportunity that exists in graceful departures is not exploited for peer notification of upcoming node departures. Thus a node's invalidated state table entries are only repaired when said node detects the departure of the nodes associated with the entries by a lack of response to either keep-alive messages or application communication.

By allowing hosts to change their DHT-IDs, the rate of state table entry invalidation across the overlay is much higher than in traditional Pastry, as every DHT-ID change requires the old DHT-ID node to leave the network. The departure of the old DHT-ID node from the overlay network can be classified as graceful, due to the fact that the physical host represented by the old DHT-ID node is still connected to the underlying network and can be reached at the same IP address. This means notification messages can be sent to notify peers, in order for them to initiate repair procedures for the state table entries invalidated

by a node's departure, without the detection lag induced by the keep-alive message mechanism.

Accomplishing peer notification requires a new locate state table called the peer list, whose purpose is to keep track a node's 'routing peers'. A node's 'routing peers' are the nodes in the overlay network that can route a message directly to the node in question. For the peer list to be of use however, a 'routing peer notification' message type must also be defined and used. This notification message is sent whenever an entry is added or removed from the original Pastry state tables: the routing table, leaf set and neighbourhood set. As the message indicates whether the entry was added or removed, the node receiving a routing peer notification accordingly adds or removes the sender from their peer list. The routing peer model is clearly exemplified by the following: If node A's routing table, leaf set or neighbourhood set contains node B as an entry, node A is a routing peer of node B and thus node B's peer list should contain an entry for node A. When node B is added to node A's state tables, node A sends a routing peer notification message to node B indicating the addition; node B, upon receiving the notification, adds node A to its peer list. If node B is, for any reason, removed from node A's state tables, a corresponding notification is sent to node B; upon receiving this 'removal' notification, node B will remove node A from its peer list as node A no longer routes to node B directly.

The peer list itself is used when a DHT-ID change operation occurs, as it is a node's routing peers that need to be informed of the change. When a node carries out a DHT-ID change, it sends another newly defined message type, the 'DHT-ID change notification', to its routing peers held in its peer list; as a comprehensive measure, all nodes within the other local state tables are also notified of the change if they are not already included in the peer list. The routing peers, upon receiving a DHT-ID change notification, initiate the state table repair procedures for the entry representing the old DHT-ID node. This is because the old DHT-ID node has gracefully "departed" from the overlay network and the associated state table entry has become invalidated.

It is of note that although the primary usage of the peer list and peer notification described here is in the event of a DHT-ID change, they can also be employed in a similar manner

when a player wishes to stop playing, initiates a graceful game shutdown and fully disconnects from the network.

4.4.3.1.1 Selective DHT-ID Change Notification

The described peer notification process is necessary in reducing the time taken to initiate repair of invalidated state table entries. When a node changes its DHT-ID however, not all of its routing peers need perform the original Pastry repair procedures to their fullest extent for the state table entry associated with the changing node. So while peer notification reduces the time taken by a routing peer to detect an "invalid" state table entry caused by a DHT-ID change, the priority of the notifying a given peer can be determined in order to reduce the number of DHT-ID change notifications sent to maintain the state tables. The priority for notifying a given routing peer depends primarily on which state table of the routing peer the changing node is placed in.

The highest notification priority is for routing peers whose leaf set contains the node changing its DHT-ID. It is imperative that these routing peers be notified by the changing node so that full repair procedures may be initiated, regardless of whether the new DHT-ID will be within or without the range of the leaf set. This is due to the nature of the leaf set, as it maintains DHT-ID proximity awareness between nodes in the overlay and is the first table to be checked in the routing algorithm. In addition to this, the leaf set is of vital importance in the proposed AOI update scheme, discussed in detail in section 4.5, and thus must be maintained as best as possible.

In contrast to the leaf set, the neighbourhood set's purpose is to maintain network proximity awareness between nodes in the overlay, and is used only in the event of rare cases in the routing algorithm. As the specific DHT-IDs within the neighbourhood set do not impact the majority of routing decisions, the priority of notification for the neighbourhood set is low.

The priority of notification for the routing table is a more complex issue, one that is dependent on the length of the shared prefixes between the old DHT-ID, new DHT-ID, and the DHT-ID of the routing peer itself. If the length of the shared prefix between the old DHT-ID and new DHT-ID is termed $Prefix_{ON}$, and shared prefix between the routing peer's

DHT-ID and the old DHT-ID is termed $Prefix_{RO}$, there are two cases to consider: when $Prefix_{ON}$ is greater than $Prefix_{RO}$; and when $Prefix_{ON}$ is less than or equal to $Prefix_{RO}$.

In the first case, when $Prefix_{ON}$ is greater than $Prefix_{RO}$, the entry within the routing peer's routing table is not truly invalidated. In this situation, the new DHT-ID will occupy the same location in the routing peer's routing table as the old DHT-ID, because the local DHT-ID shares the same prefix length and digit after the shared prefix is the same for both old and new. The only invalidation with regards to the entry is that the DHT-ID is not up-to-date, and this will not affect routing except in the rare case branch of the routing algorithm; even then the effect is minimal because the old and new DHT-IDs are relatively the same numerical distance from the routing peer's DHT-ID. Thus it follows that no extensive state table repair is immediately required and the priority of notification is low.

In the latter case, when $Prefix_{ON}$ is less than or equal to $Prefix_{RO}$, the routing table entry is completely invalidated because the new DHT-ID will not occupy the same place in the routing table as the old DHT-ID, according to the routing table structure discussed in 2.3.1. In the case that $Prefix_{ON}$ is less than $Prefix_{RO}$, the new DHT-ID will be placed in a different row of the routing table. In the case that the values are equal, the column of the new entry will differ. In both of these scenarios, the routing peer has an invalid entry which must be repaired, and thus the notification priority is high.

Having determined that there are high and low DHT-ID notification priorities, selective peer notification can be carried out in order to reduce the amount of messages exchanged to simply maintain the overlay. In the case of high priority routing peers, a DHT-ID change notification message must be sent out without delay. In the case of low priority routing peers, DHT-ID change notification messages can be sent lazily in response to messages addressed to prior DHT-IDs. If strict maintenance of the overlay is required, low priority routing peers should also be notified but these messages can be delayed in order to meet network throughput constraints.

As a result of the notification priority depending on which state table of a given routing peer a node is placed in, some design changes must be highlighted. First and foremost, the 'routing peer notification' message definition must include a field that indicates which of

the routing peer's state tables the node being notified is within. The peer list structure must also track this information for each of its entries and make it available for selective notification. If a routing peer holds a node in more than one state table, the highest priority table is chosen for the notification dispatch. To summarize, the notification priority for the state tables from highest to lowest, is as follows: leaf set, routing table, neighbourhood set.

4.4.3.2 DHT-ID Time Stamping & State Table Scrubbing

While the peer notification mechanism reduces the latency in detecting invalid state entries, it does not remove it altogether. That resulting window of time can be problematic if a peer with an invalid, but yet undetected, entry sends its local state tables to another peer as part of the join or repair procedures. When the receiving peer merges the incoming state tables, a given host can be present multiple times in a state table with different DHT-IDs. This means there are multiple overlay connections from one peer to another which is an unacceptable scenario, as it violates the structured nature of DHTs and leads to loops in the network. It is especially problematic in this architecture as state tables are continually exchanged between peers with movement induced re-joins and repairs.

The first step in addressing this is to have all DHT-IDs time stamped with their creation time. This does not need to be a globally synchronized value, but simply the time local to the node creating the DHT-ID. The reason for this is that timestamps are only compared to other timestamps that originate from the same host. In addition to this time stamp, each DHT-ID in the state tables is already associated with a host network address, such as an IP address and port number, a combination which should be unique to each host.

With this combined DHT-ID information, it is possible to properly and safely 'scrub' each local state table to remove host duplication. Scrubbing involves checking all entries within a particular state table to ensure that each host, identified by its network address, is associated with only a single DHT-ID. If multiple DHT-IDs are found to be bound to a single host, the entry with the latest time stamp is kept. In order to maintain the network structure as best as possible, each state table is scrubbed after any change is made to it.

4.4.3.3 Buffering Prior DHT-IDs

Another issue arising from the time between a node assuming a new DHT-ID and the associated notifications reaching its peers is that there may be messages en-route to the node, addressed to the old DHT-ID. In traditional Pastry, if a message arrives at a node who's DHT-ID differs from the one the message is addressed to, it is assumed that the message is destined for another node and forwarded along in the overlay network. This assumption is no longer valid in the scenario of player movement and changeable DHT-IDs, and so messages are no longer simply discarded without consideration if they are addressed to a DHT-ID that differs from the local one. In order to circumvent this, a mechanism for nodes to buffer their prior DHT-IDs is required.

When a message is received by a node, the nature of the message must be considered. Should the message type be one that is to be forwarded, such as a join request, the message is forwarded along without issue. If the message type is one that is directly sent to the host in a single hop, such as a peer notification or a state table request sent by repair procedures, the message's destination DHT-ID is checked. If it differs from the node's current DHT-ID, but is present in the list of prior DHT-IDs, the message is accepted locally; otherwise it is assumed that the message has been improperly routed.

The node may then send a DHT-ID change notification to the original message sender in order to ensure that they are up-to-date. This particular notification dispatch trigger is a lazy mechanism for detecting and notifying peers of network inconsistency is employed as a rear guard against peers that did not receive DHT-ID change notifications, whether due to network failures or aggressive application of selective notification.

The number of prior DHT-IDs buffered should not be infinite. The reason for this is clear when considering scenarios where an old DHT-ID has been assumed by another node and the local node has moved away from the location represented by the old DHT-ID. If a direct message intended for the node currently at the old DHT-ID is accidentally routed to the local node by an out-of-date routing peer, the message should not be considered for local acceptance. In order to prevent prior DHT-IDs from being buffered for an overextended period of time, the entries within the prior DHT-ID list are time stamped with when they

were discarded for a new DHT-ID. The prior DHT-ID list is then inspected periodically, and entries expired according to a game defined timeout are removed. In addition to this, all incoming messages should be checked against the list in order to detect whether any prior DHT-IDs have been assumed by another node. If a message's originator DHT-ID is found to be within the prior DHT-ID list, that entry is also removed from the list.

4.4.3.4 DHT-ID Request Contention

As discussed in 2.3.3.1, traditional Pastry's design handles concurrent node arrivals optimistically, assuming this is a rare case. If a DHT-ID is in contention between two joining nodes, the issue is handled through the use of time stamped state updates and having one node leave and rejoin the network with a different DHT-ID. There is however, no defined mechanism to inform routing peers, those with the DHT-ID/IP address pair for the rejoining host in their state tables, that the DHT-ID is held by another host. This results in inconsistencies within the overlay that must first be detected in order to repair them.

With location based DHT-ID generation and accommodation of player movement, there is an increased potential for DHT-ID contention. This is especially so at environment hotspots, where many players are in close proximity to each other, rendering the optimistic method of handling DHT-ID request contention insufficient. One facet in handling DHT-ID contention is passive avoidance, which is to first ensure that there is a sufficient density of vertices available for use as DHT-IDs. Appendix A outlines the criteria and logic for determining an appropriate Hilbert Curve order, which is a determining factor of vertex density. Moreover, the densely populated region(s) of the environment are further provided with a greater number of vertices through the use of the non-uniform Hilbert curve, discussed at length in section 4.3.1.3.

Passive avoidance is supplemented with an active mechanism to deal with DHT-ID request contention. This mechanism leverages the convergence property of Pastry's routing algorithm, which ensures that if a node with a given DHT-ID does not exist, any message destined for that DHT-ID will be delivered to the node with the numerically closest DHT-ID. If two separate nodes issue requests for the same DHT-ID not already in use, both request messages will be routed to the same node. The node that receives these messages simply

grants the first request to arrive, while denying the second. The node that receives the denial can then rejoin with another DHT-ID close to its location in the environment. This gracefully prevents many of the issues with the approach in traditional Pastry. There is now minimal overhead, as the nodes need not send each other state information because only one node is granted use of the DHT-ID, which means that overlay consistency is also maintained.

In order to implement the active mechanism, each node must buffer the DHT-IDs for which it grants requests. Thus if another request for the same DHT-ID is received, the node checks the buffer, becomes aware of the contention and denies the second request. Once the initial requesting node joins the overlay with the DHT-ID, it should send a notification to the node that granted its request. The node being notified can then remove the sender's DHT-ID from its buffer, as future requests for said DHT-ID will be routed to the now existing node that holds it. In addition to the notification, entries in the buffer can be time stamped and removed after a timeout.

4.5 Area of Interest Updates

The final component of the architecture allows players to send game updates about their actions and movements to other players within the environment. This is possible as the overlay is capable of efficiently adapting itself to match the player positions over time. By placing this capability into the architecture, game development is simplified as the mechanics of game updates are abstracted away from the game's design. This is the purpose in adapting Pastry to include location-aware DHT-ID assignment and accommodating player movement.

In order to compensate for the discrepancy between a DHT-ID and player location, it is recommended that the range query area should be larger than the actual game update AOI. These discrepancies can occur due to the DHT-ID threshold slightly delaying the reflection of player movement within the overlay, and the time taken by the overlay itself to reconfigure itself due to movement.

In brief, there are two aspects to AOI updates: peer discovery and game update dissemination. Peer discovery is the process of a player discovering other participants in an area of the environment, usually within the local vicinity as those participants will be interested in the player's status. Update messages are then disseminated to the interested participants that have been discovered. There is however, more than one approach that can be taken in carrying out player discovery and update dissemination, which are discussed in the following sections.

4.5.1 Peer Discovery

Without yet speculating on game update dissemination, first consider the aspect of player discovery, the basis of which is performing a range query over the Hilbert Curve. A range query is the process of determining the vertices within a given area or search space. Figure 17 demonstrates a range query on a two dimensional 3rd order curve that yields 9 vertices found to be within the search space, which equates to 9 potential DHT-IDs.

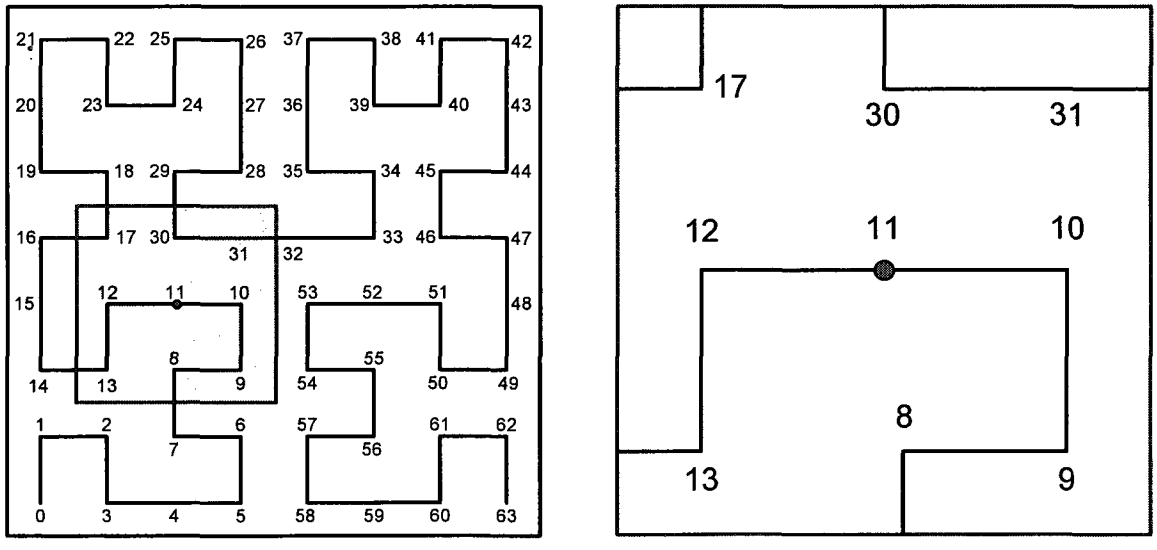


Figure 17: Hilbert Curve Range Query about a Specified Location

In itself a range query over the curve is not sufficient for peer discovery, as there is no assurance that every vertex is an active DHT-ID in the overlay at the time of peer discovery. Thus the goal of the peer discovery process is to contact the set of active nodes with DHT-IDs based on the vertices found within the range, which equates to finding the players in the area. To accomplish this, the range query calculations must be coupled with network

communication. General approaches for peer discovery are discussed and chosen here, with the specifics of the range query algorithm following in the subsequent section.

In a fully distributed approach to peer discovery, the node initiating the process starts by calculating a single DHT-ID within the search space, such as the numerically smallest. A peer discovery request message, which specifies the search space, is then dispatched by this node to the calculated DHT-ID. Due to routing convergence, the message is delivered to the active node in the overlay with the DHT-ID numerically closest to the one the message is addressed to. This receiving node makes use of its leaf set to further propagate the peer discovery process, as the leaf set of a node holds links to nodes with numerically adjacent DHT-IDs. This receiving node calculates DHT-IDs subsequent to its own within the specified range, until a DHT-ID that matches a known active node in is found and is used as the next destination of the peer discovery request message. If the next DHT-ID is beyond the awareness of a node, the message is propagated forward by the normal routing algorithm. Essentially in this approach, each node is responsible for notifying the next active node in the query range and continuing the process. The advantage of this approach is that both the computation and communication responsibilities are shared amongst many nodes in the overlay. The drawback however, is the time required in completing the process; simply summing the time required for DHT-ID computations and network delay against the potential number of nodes in an AOI results in an unacceptable delay between request initiation and fulfillment.

As the fully distributed approach violates the primary goal of achieving low latency, an alternative approach is taken. Here, all the potential DHT-IDs within the range are calculated by the initiating node. With the entire set of potentially active DHT-IDs, multiple peer discovery request messages are sent out through the overlay to DHT-IDs within the search space. Each request specifies a range of DHT-IDs numerically adjacent to the message recipient. Ideally the message recipient is an active node that would use its leaf set to directly contact the other DHT-IDs within the range specified in the request message. If the size of the DHT-ID range is beyond its leaf set, the responsibility for propagating the request further is given to other nodes within the range.

This alternative approach is relatively more expensive with regards to computation per node and bandwidth consumed; the initiating node must calculate all DHT-IDs within the area of interest and dispatch multiple peer discovery request messages. These messages are further propagated by the receiving peers, who must carry out further calculations on the message's DHT-ID range. Duplicate request messages may also converge on the same node in sparsely populated regions, which is an undesired side-effect. On the other hand, the time taken to find and contact the active peers within the search space is considerably reduced. This is due to the configurable binning of DHT-IDs in each peer discovery request sent out. The methods for binning and propagation can vary slightly, perhaps changing the balance between duplicate delivery, bandwidth usage and propagation delay.

The first propagation method is single-sided, which propagates the peer discovery request by ascending DHT-ID ranges alone. Binning of DHT-ID ranges is done in multiples of half the leaf-set size, $\lfloor L \div 2 \rfloor$, and request messages are sent to the DHT-ID at the start of each binned range. Further propagation by nodes receiving a request with an embedded DHT-ID range is simple: they forward the request to the nodes in their leaf set with numerically larger DHT-IDs that are within range, and since these messages are one shot they are not propagated further. If the range extends beyond the current node's leaf set, responsibility for further propagation is given to the node at the edge of the leaf set.

Alternatively, both sides of the leaf set can be used in the peer discovery propagation, with binning size now a multiple of the maximum leaf set size L. In this double-sided propagation, the starting peer for initial request dispatch is now chosen from the middle of the range. The receiver of a request now propagates to both sides of their leaf set for DHT-IDs that are numerically both higher and lower, as long as they are within the range specified in the message. Further request propagation for DHT-IDs beyond the leaf-set range is similarly done for both edges of the leaf set. In terms of implemented logic, this approach is somewhat simpler due to the lack of special cases that occur in single-sided propagation when a node outside the range receives a request to propagate.

In both methods, once a peer discovery request sent by the initiating node is received, the theoretical maximum hop count value required to reach any subsequent DHT-ID within the

request range is that of the multiplier. In the common scenario where all DHT-IDs within the search space are not active nodes, it is highly likely that the maximum number of hops may not be reached. This is especially true when considering curves of higher orders, as the vertex density will far outweigh the It is important to remember that the total hop count for the peer discovery message in this example is the addition of this "maximum" with the number of hops required to initially route the request message from the initiating node to the receiver through the overlay. The effect of the two methods on the number of messages sent and the presence of duplicate delivery of a request to nodes are evaluated in the proof-of-concept.

4.5.2 Hilbert Curve Range Query

Having determined the approach to take for peer discovery, the Hilbert Curve range query algorithm can be designed accordingly. As stated, the goal of the range query algorithm is for a single node to calculate all the available DHT-IDs within a defined search space. As discussed in Chapter 2, existing work on applying the Hilbert Curve to data storage is presented by Lawder in [23]. The body of work, using a state table generated tree representation of the Hilbert Curve, presents a range query algorithm that also operates on the aforesaid representation. This work is leveraged and optimized for the purposes of peer discovery in this MMOG routing overlay architecture.

4.5.2.1 Non-Uniform to Standard Hilbert Curve Conversion

The non-uniform Hilbert Curve employed in this architecture for DHT-ID generation is not comprised of identical cups. This is the reason for the geometric nature of the mapping algorithm in 4.3, as the geometric floating-point representation of cups allows for geometric transformations to be applied. A drawback of the tree representation is that it does not have this geometric flexibility. The nodes that comprise the tree represent the 1st order curve components of the Hilbert curve. They must be a bounded set in order to be generated by a state machine. Lawder's tree representation also employs bit based integer values for vertex indices as well as vertex location. These properties of the tree representation mean that it is unable to inherently handle non-uniform distribution of vertices. In order to employ Lawder's tree representation and associated range query logic, the first step is to reverse the effect of the non-uniform distribution applied to the curve.

This is the reason it is stated that the ratio function used for non-uniform distribution must have a computationally achievable inverse function.

The inverse non-uniform distribution function is applied to the range query's search space in order to normalize it for a standard Hilbert Curve. The reasoning is that a range query over a standard Hilbert Curve with the normalized search space will yield identical results to a range query over a non-uniform Hilbert Curve with the original search space; this effect is illustrated in Figure 18. With the normalized search space, a range query algorithm based on the tree representation of the Hilbert Curve can be utilized.

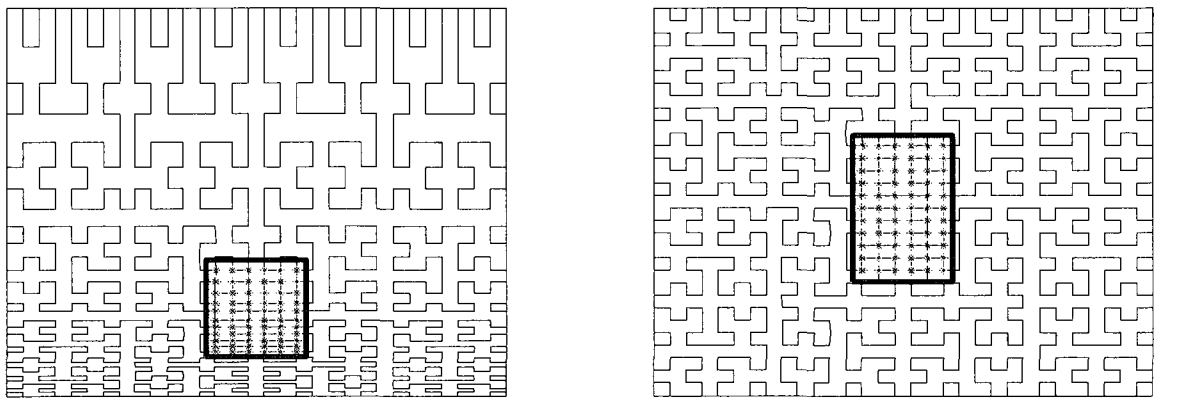


Figure 18: Conversion of Range Query over Non-Uniform to Standard Hilbert Curve

4.5.2.2 Tree-Based Range Queries

Lawder's unmodified algorithm is unsuitable for the chosen peer discovery approach, as it determines a single subsequent curve vertex ID within the search space given an initial starting point. What it provides instead, for range query purposes, is baseline traversal logic for the Hilbert Curve tree representation.

As the tree representation is incapable of inherently handling floating point values, the first step in the range query algorithm is to express the normalized search space in terms of coordinates that can be related to the tree representation. This is necessary because the initial search space is relative to the game environment, while the tree representation coordinates in a given dimension range from 0 to $2^K - 1$, where K is the given curve order and the coordinate values differ by 1. The final search space boundaries for each dimension must fall within this coordinate value range. It is possible in this conversion that vertices

previously included or excluded from the game environment relative search space are now excluded or included due to the change in precision. As these vertices fall at the boundary of the search space, and it is already suggested to have the space exceed the actual game AOI, this discrepancy is not overly concerning. It is for this reason however, that the tree representation is not used in DHT-ID generation.

Once the search space boundaries are expressed as integer values that fall within the dimensional coordinate ranges, the tree is traversed in order to find the vertices and associated DHT-IDs within the search space. Essentially, the tree is searched in a divide and conquer manner; a branch of the tree is eliminated from traversal if it can be determined that the space represented by the branch does not intersect with the search space. This eliminates the need to visit each vertex individually, and is accomplished by exploiting the hierarchical properties of vertex coordinates that exists in this tree representation of the Hilbert Curve.

In order to carry out the checks that eliminate branches from traversal, bit arrays are required. These bit arrays have a fixed length of K, the order of the curve and depth of the tree; they are also initialized with zero values in every index. For each dimension, three bit arrays are required with each associated to one of the following values: the current vertex coordinate, the lower search space boundary and the upper search space boundary. For clarification, consider first the bit array associated with the current vertex coordinate. Recall that at a given a sub-space within a leaf node, the vertex coordinate of the sub-space can be calculated by concatenating the coordinate bits of all the sub-spaces traversed from the root node up to, and including, the current sub-space. Essentially, the proposed coordinate bit array collects the sub-space coordinate bits as the tree is traversed for this purpose, so that the coordinates of a vertex can be determined. This applies to the search space boundary arrays as well, except the boundary arrays collect bits from the binary expression of the integer search space boundary values. The array index of the bits to currently replace in these bit arrays, as well as the array index to copy from the boundary values, is equal to the current level of the tree. While the coordinate bit array is updated every time the current sub-space changes, the boundary arrays need only be updated the tree is ascended or descended. Whenever the traversal algorithm is ascends the tree, the

bit arrays are filled with a zero at the index of the current tree level before actually moving to the higher level.

These integer values represented by these bit arrays are used in determining the truth of Equation 4.1.

$$boundary_{XYZ_{min}} \leq coordinate_{XYZ} \leq boundary_{XYZ_{max}} \quad (4.1)$$

Equation 4.1 is the expression of checking whether the integer value of the coordinate bit array falls within between the ranges of values respectively bounded the integer values of the two boundary bit arrays. In the subsequent paragraphs, referring to the result the sub-section check signifies the truth of Equation 4.1.

The range query algorithm itself, having described the required background, becomes a relatively simple loop that is executed once for every sub-space encountered during traversal of the tree. The algorithm is initialized to start at the top level of the tree, at the first sub-space of the root node. The traversal ends when the traversal returns from a lower level to the last sub-space of the root node with no further sub-spaces available. At each node sub-space, the result of the sub-space check is used to determine the next action.

If the sub-section check result is false, the traversal algorithm moves horizontally to check the next sub-space. If the current sub-space is the last in the node, the traversal algorithm goes back up the tree, reinstating the node state and sub-space index that was buffered from the level up.

If the sub-section check result is true, and the current node is not a leaf node, it indicates that the branch extending from the current sub-space contains vertices of interest. The traversal algorithm descends one level of the tree to first sub-space of the next node. The specific node is determined by the state machine described in 2.4.1. When going down a level of the tree, the current node state number and sub-space index is stored in a buffer, which is required for backtracking up the tree.

If the sub-section check result is true, and the current node is a leaf node, the algorithm has arrived at a curve vertex within the search space, and must determine the corresponding

vertex ID. This requires another bit array to collect the vertex ID bits of all the sub-spaces traversed from the root node up to, and including, the current sub-space. The size of this array is $D \times K$, the number of dimensions multiplied by the curve order of the tree. The vertex ID bits of a sub-space are simply the sub-space index value in base 2 representation.

4.5.2.3 Performance Optimizations

In the following subsections two optimizations to the base algorithm that yield significant performance gains in the execution of the range query are discussed. These optimizations are generated from analysis of the range query data usage and the Hilbert Curve properties. It is acknowledged that further performance gains can be attained by generally established methods that achieve greater computational efficiency, such as multi-threading or dedicated hardware usage.

4.5.2.3.1 Range Representation

The first optimization stems from the fact that the peer discovery approach chosen for this architecture does not deal with individual DHT-IDs calculated by the range query. Rather, it is ranges of continuous DHT-IDs that are manipulated when creating and sending peer discovery request messages. While ranges of continuous DHT-IDs naturally occur in range query results from the curve exiting and re-entering the search space, ranges can be further divided by the peer discovery process. Yet it is still desired that the range sizes finally sent out are not simply a single DHT-ID. In addition to this, when a node receiving a peer discovery request message check its leaf set peers and then propagates the discovery message, it need not deal with individual DHT-IDs in the request range. This is because the DHT-IDs within its leaf set are simply checked for whether they are within the boundaries of the range of DHT-IDs within the request message.

In view of how continuous DHT-ID ranges have greater significance than individual DHT-IDs, it follows that the results of the range query algorithm should be tailored to represent ranges rather than DHT-IDs. The original returned algorithm result of an array holding calculated DHT-IDs is replaced with an array of 'range representation' objects. The structure of the range representation object is the first DHT-ID in the range and an integer indicating the length of the range, including the first. When the first vertex within the

search space is found in a range query, a new range object is created with the vertex's DHT-ID with a length of 1. The range object's length parameter is incremented for each subsequent vertex until a DHT-ID is found to be outside the search space, at which point the range object is stored in the array of ranges to be returned by the algorithm. When the next vertex within the search space is found, another range object is created and the pattern is followed.

The primary benefit of incorporating this range representation format is a reduction in the memory required by the algorithm. Previously each DHT-ID within the search space would be calculated and stored. Now only one full fledged DHT-ID for the start of the range and a value for the length of the entire range itself is stored. The final DHT-ID at the end of the range, which is of potential interest, is easily calculable.

4.5.2.3.2 Enveloped Branch Detection

In terms of the algorithm's computational load, it is the tree traversal and the checks performed multiple times per node that are the significant contributors. Enveloped branch detection requires an additional check within the algorithm. If activated, it circumvents the need to traverse an entire branch of the tree to the bottom level which results in many saved calculations which outweigh its cost. As an example of enveloped branch detection, consider the range query on the 3rd order curve illustrated in Figure 19 . The range query's search space covers many vertices at its edges, but also fully envelopes the 2nd order curve component at the top right of the filled space. Enveloped branch detection detects that this 2nd order curve component lies fully within the search space, and use knowledge of Hilbert Curve properties to quickly calculate the DHT-IDs associated with the highlighted formation.

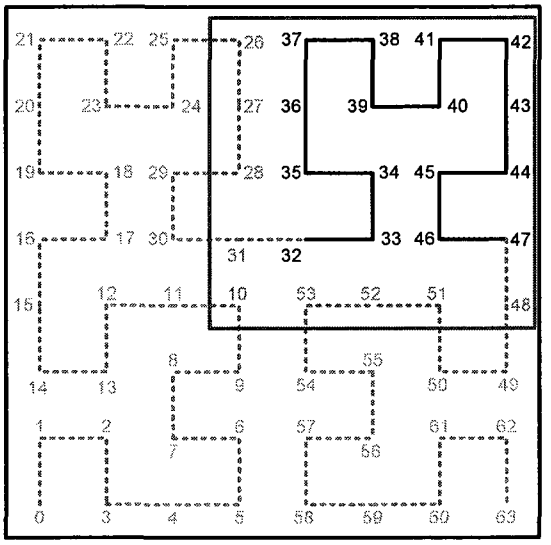


Figure 19: Example of an Enveloping Range Query

Enveloped branch detection's name is derived from the structure of a curve component within the tree representation of the Hilbert Curve. In the figure above, the curve component shown to be within the search space is a 2nd order curve in its own right. If the 2nd curve component is similarly highlighted in the tree representation of a 3rd order Hilbert Curve, as in Figure 25, it can be seen that an entire branch of the tree represents the 2nd order curve component.

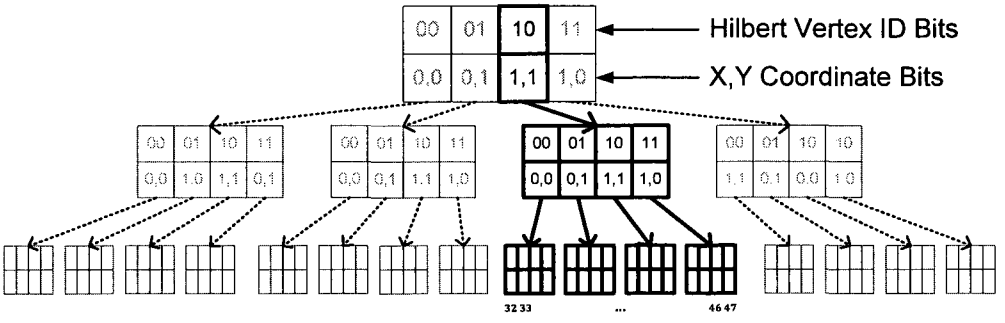


Figure 20: Highlighted 2nd Order Curve Component in 3rd Order Hilbert Curve

The name is derived from the fact that an entire branch of the tree is enveloped by the range query search space. Any time an entire branch is found to be enveloped within the search space, no matter what level said branch stems from, it is known that all the DHT-IDs represented by the leaves of that branch will be included in the range query results. Equation 4.2 states *N*, the number leaves in a branch, can be determined by the number of dimensions *D* and the number of levels *L* that the branch is composed of.

$$N = 2^{D \times L} \qquad (4.2)$$

Detecting whether a branch is enveloped is done before the branch is entered by the tree traversal algorithm and so the logic check is performed for every node sub-space in the tree, similar to the logic check that determines whether to descend or ascend the tree. This new logic check requires an additional vertex coordinate bit array for each dimension. Unlike the previous coordinate bit array however, the new array is appended with 1's instead of 0's for the tree levels lower than the current one. In order to distinguish between these coordinate bit arrays, the former 0-filled array is termed the minimum coordinate bit array, while the new 1-filled array is termed the maximum coordinate bit array. In order to determine if a branch is completely enveloped by the range query search space, the corresponding integer value of these maximum and minimum coordinate bit arrays are checked to see if they are both within the boundaries of the search space. Note that this does mean the boundary bit arrays, but the original integer boundary values of the search space. If this proves true for all dimensions, the entire branch is found to be within the search space.

Once a branch is found to be enveloped, it is then a matter of extending the length of the current DHT-ID range object with the number of DHT-IDs in the branch, which can be calculated by Equation 4.2. If a new range object needs to be created, the first DHT-ID can be determined by filling the remainder of the DHT-ID bit array with zeroes and calculating its value. Having updated the current range representation object, the tree traversal does not need to continue down this branch of the tree. It can instead continue horizontally to the next entry in the node, or vertically up a level if this was the final entry within the node.

Although Lawder includes a similar check in his algorithm, its purpose is to curtail only the special case of finding the first vertex in the enveloped branch. Every subsequent vertex in the branch however is calculated by fully descending the tree. Thus in 4.5.2.2's baseline range query algorithm that calculates all DHT-IDs within the search space, Lawder's use of enveloped branch detection is of little use. Due to the inclusion of the aforementioned range representation optimization however, enveloped branch detection is reinstated and

exploited with the additional range object calculations in order to achieve greater computational efficiency.

4.5.2.4 Range Query Algorithm Pesudo-Code

The baseline range query and its optimizations discussed thus far are combined into the range query algorithm presented below. Notation relevant to the algorithm is first detailed in Table 4, which outlines variables and simple functions used in the algorithm. As the range query algorithm is quite extensive in full, sub-procedures have been outlined in the algorithm discussion and are fully detailed with additional definitions in Appendix B.

Table 4: Definitions for the Hilbert Curve Range Query Algorithm

K	given Hilbert Curve order upon which range query is performed
$ssMax_{XYZ}$	given upper boundary values of the search space
$ssMin_{XYZ}$	given lower boundary values of the search space
$coorBMax_{XYZ}$	bit array of size *K* to hold coordinate bits up to the current level with trailing 1s
$coorBMin_{XYZ}$	bit array of size *K* to hold coordinate bits up to the current level with trailing 0s
$boundaryBMax_{XYZ}$	bit array of size *K* to hold upper boundary bits up to the current level with trailing 1s
$boundaryBMin_{XYZ}$	bit array of size *K* to hold lower boundary bits up to the current level with trailing 0s
curRange	range representation object that holds a DHT-ID and range size integer value
arrRanges	array of range representation objects returned by the algorithm
toInt(*B*)	converts the bit array *B* to an integer value
areBounded(*J,K,L,M*)	determines if values *J* and *K* are both bounded by *L* and *M*
stateCoorBits(*S, I*)	returns $coordinate_{XYZ}$ bits for sub-space with index *I* from state *S*

As many aspects of the algorithm are repeated for each dimension, the algorithm definitions have been simplified so that all dimensions are dealt with at the same time. If an item has the *XYZ* subscript notation, it signifies that the variable and/or associated operation is repeated separately for each of the three dimensions in the exact same

manner. It is also assumed that array indices and sub-space indices both start at 1, and that if an array of bits is interpreted as an integer value, the first index is considered the MSB.

Algorithm 3: Pseudo-code for Hilbert Curve Range Query Algorithm

```
1:  normalizeSearchSpace( )
2:  initializeAlgorithm( )
3:  repeat
4:      curIndex ← curIndex + 1
5:      if (curIndex > 2^3 and curLevel ≠ 1) then
6:          moveUpTree(curLevel)
7:      else if (curIndex ≤2^3) then
8:          coorBMax_XYZ ← stateCoorBits(curState, curIndex)
9:          coorBMin_XYZ ← stateCoorBits(curState, curIndex)
10:         coorMax_XYZ ← toInt(coorBMax_XYZ )
11:         coorMin_XYZ ← toInt(coorBMin_XYZ)
12:         bdMax_XYZ ← toInt(boundaryBMax_XYZ)
13:         bdMin_XYZ ← toInt(boundaryBMin_XYZ)
14:         if ( areBounded(coorMax_XYZ, coorMin_XYZ, ssMin_XYZ, ssMin_XYZ) ) then
15:             computeEnvelopedBranch( )
16:         else if (areBounded(coorMax_XYZ, coorMin_XYZ, bdMax_XYZ, bdMin_XYZ)
            then
17:             if (curLevel ≠ K) then
18:                 moveDownTree( )
19:             else
20:                 computeLeafWithinRange( )
21:             end if
22:         else if (curRange ≠ Ø)
23:             rangeCount ← rangeCount + 1
24:             arrRanges[rangeCount] ← curRange
25:             curRange ← Ø
26:         end if
27:     end if
28: until (curIndex > 2^3 and curLevel = 1)
```

Line 1 in the algorithm calls the sub-procedure for normalizing the search space to the tree based Hilbert Curve representation. Not only does this apply the inverse of the non-uniform distribution function on the Z-dimension, but also quantizes the search space to integer values in accordance with the tree representation. Line 2 calls the sub-procedure for initializing the various bit arrays, as well as other variables.

Lines 3-28 are the main loop that traverses the Hilbert Curve tree, and has the condition of stopping only when the traversal returns to the root node, specifically to the sub-space index that is beyond the number of sub-spaces within the node.

Line 4 increases the current sub-space index, which achieves horizontal traversal of the tree. Line 5 detects whether the current sub-space index is beyond the current node's limit, which initiates the tree traversal to the node above in Line 6. Line 7 ensures that the traversal is in a valid position within the current node, and if so, gives way to lines 8-26 which represent the calculation of DHT-IDs within the search space.

Lines 8-13 simply keep the necessary arrays up to date with information of the coordinates at this level and convert them to comparable integer values. The **if** statement at line 14 represents the enveloped branch detection which, if successful, initiates the computation in line 15 that circumvents the necessity of full descending the tree. If enveloped branch detection is unsuccessful, line 16 checks if the current branch has any DHT-IDs within the search space. If this check returns true, the tree is either descended further (line 18) if the traversal is not already at the bottom of the tree (line 17). If the traversal is already at the bottom of the tree, the current leaf sub-space is added the current range representation object. Both functions for enveloped branch and leaf computation will or update the current range representation object, or create a new one if none exists.

If there is no indication from the previous checks that this portion of the tree should be pursued further, lines 22-25 store the current range representation object in the results to return if such an object exists.

4.5.3 Game Update Dissemination

The manner in which game update dissemination is carried out is determined by how peer discovery is incorporated into this process. Again, there are two approaches to be weighed: update push and update request.

The concept of update push is essentially having game updates disseminated by the peer discovery process itself. Here, the peer discovery request message would be combined with an update message; the updates are "pushed" by a player onto the nodes in its AOI. At each

game update cycle, a peer discovery process of the player's defined AOI is initiated, and the peers discovered would not only propagate the discovery message but also extract and apply the game update included within the message. While this simplifies the overall scheme of game update dissemination, the disadvantage is that the game updates suffer the same latency as peer discovery. While the number of hops taken by peer discovery messages can be influenced by the configured DHT-ID range bin size, the number of hops taken in initially routing the message cannot be directly controlled. Thus it is possible, even with location aware routing, that game updates disseminated by the peer discovery process will not reach interested peers in a timely manner. In addition to this, the execution of the Hilbert Curve range query algorithm at every game update cycle, which occurs multiple times a second, would consume significant computational resources. Another inconvenience is related to game mechanics; for update push to be effective, all players must have equivalent AOI ranges.

The update request approach attempts to resolve these issues changing the responsibility given to nodes. In the previous update push approach, each node is responsible for determining which peers are interested in its game updates along with pushing its updates towards them. In this new approach, each node is responsible for requesting updates from peers within its AOI, as well as sending its own updates to the nodes that have requested its game updates. To achieve this, the peer discovery process is now used to propagate a request for game updates from the initiating node. Each node that receives a request message stores the requesting node's address in a local list, and all addresses within this list are then sent a game updates at every cycle. The primary advantage of this is that game updates can now be sent directly to interested peers in a single hop. Another advantage of having nodes responsible for request updates from other nodes is that this scheme also allows for variable sized AOIs. For these reasons, update request is the preferred approach for game update dissemination in this architecture.

As players are assumed to be consistently moving, update requests are sent out periodically by a node to ensure receipt of game updates from the players entering its AOI. While this does require the range query algorithm to be executed regularly, the frequency that the update request process is initiated can be much smaller than that of game updates,

thus saving computational resources. Similarly, the entries within a local update list are expired after a time, so that players no longer interested in another player's updates will not receive them unnecessarily. When an update request arrives for a node that already exists in the local update list, that entry's timestamp is simply refreshed.

With the update request able to support variable sized AOI, a player can have multiple AOIs from which it requests game updates. This allows the advantageous ability of an MMOG to implement pre-emptive AOI, which is an AOI that that looks ahead to where the player will require updates from in the near future. This can be determined by current player's behaviour and destination, and implementation is achieved by the game engine simply initiating the update request process for multiple AOIs. The efficacy of a given pre-emptive AOI configuration is dependent on the nature of the MMOG itself, and thus is game specific.

4.6 Transport Protocol Selection

This architecture's primary focus is location aware routing in order to achieve timely communication between peers for a fully distributed MMOG, and is designed to be deployed at the application layer of the networking model. A thought however, must also be given to ensuring that each peer is furnished with game updates not only in a timely fashion, but reliably as well. Both of these aspects, timely and reliable delivery, are influenced by the chosen underlying transport protocol. The protocols under the transport layer are outside the scope of selection, as the prevailing networking layer is Internet Protocol (IP) and the nature of the Internet abstracts the link and physical layers.

The Transport Control Protocol (TCP) is a connection-oriented transport layer protocol designed to accomplish reliable and ordered packet delivery over IP, and is used in commercial MMOGs such as World of Warcraft and Lineage II. The nature of TCP is that of a byte stream tunnel established between two endpoints, where the application layer at each endpoint may write and read data to and from the stream. Use of TCP in P2P MMOGs however, does have the potential for issues with regards to scalability and game play. The first issue is that of scalability, as a unique TCP connection must be established between hosts in order for them to communicate. This may be problematic if the number of connections needed for a network node to participate in the overlay exceeds the available

resources. In addition to this, TCP connections introduce significant delay in several fashions. The first is through buffering, as multiple small data packets written to a connection stream may not be dispatched immediately, but may instead be buffered until the MTU size is reached and then dispatched altogether. Additionally, TCP employs flow and congestion control, and so does not maintain high throughput in the face of deteriorating network conditions. Computational overhead is also increases, as resources are also consumed in connection maintenance, sequence number processing and duplicate packet detection, all of which are required by TCP's design for reliable and ordered packet delivery. While TCP's initial 3-way handshake ensures mutual awareness and connectivity, it also requires a greater number of messages to perform.

Alternatively, the User Datagram Protocol (UDP) is a potential connectionless transport layer protocol candidate, as it is considered to be relatively lightweight compared to the connection-oriented TCP. UDP's purpose is to simply identify datagrams' source and destination application endpoints through port binding. In terms of network behaviour, UDP is aggressive due to its lack of flow and congestion control. Network conditions are not taken into account, and thus datagrams are not buffered or held back but immediately sent out over the network. Computationally UDP is also simpler, as sequence number processing, duplicate packet detection and content checksum calculations are nonexistent. These characteristics are advantageous for networked games where maximized performance is desirable, but come at the cost of total lack of reliable communications.

A middle ground between the two transport protocols can be found if UDP is combined with an application layer networking component. While the former handles delivery to the application layer, the latter takes responsibility for providing the reliability required in MMOG communications. This is an attractive solution over the full-fledged TCP, as the degree of reliability chosen can be customized for game communications. This approach is existent in various available networking libraries, such as ENet [32], Raknet [33], Replicanet [34], HawkNL [35] and OpenTNL [36].

Chapter 5 - Proof of Concept

Testing the performance of P2P network overlays in a real world network of peers is usually highly impractical. This stems from the fact that networks with a sufficient number of peers to be considered "large scale", such as the Internet, are awkward for evaluating performance. The underlying networks, and perhaps peers themselves, are not under the control of the researchers and can be prone to failure and other unforeseen events. While this may be acceptable in testing connectivity repair mechanisms, determining baseline performance becomes problematic. Conversely, having a true large scale network as a fully contained and controlled environment is infeasible, usually in terms of cost.

As an alternative to live deployment, simulation is favoured in research as a means to test the performance of P2P systems. Simulation provides the advantage of allowing the network environment to be tailored, allowing for varying network conditions to be modeled and tested. Within these controlled conditions, the relative baseline performance of different approaches can be measured in order to determine overall feasibility. The drawback to simulation is that it is computationally expensive, as multiple processes are executed on a limited set of processors. This places a cap upon the overall complexity that can be simulated.

In order to evaluate the proposed architecture, it is implemented and tested in a simulated environment. The proof of concept includes both network and game play simulation in order to gain insight on the performance of the adapted network overlay while being used for MMOG communications. Both aspects of the simulation are discussed in detail in the ensuing sections.

5.1 Network Simulation

5.1.1 PeerSim

PeerSim [37] is the open source, general purpose P2P simulator chosen for this proof of concept. It offers both cycle-driven simulation and discrete-event simulation using separate

simulation engines, with the latter chosen here as it provides a more realistic evaluation basis. As a general purpose simulator, the protocol stack instantiated at each node is entirely configurable. This is advantageous as the lower layers of the networking protocol stack can be abstracted away, allowing focus to be placed on the application and transport layers. The protocols at these layers can be those included with the simulator, or defined by the user.

PeerSim provides an API that allows users to extend several base classes in order to customize their simulation. Along with protocols and nodes, users may also define "Initializers" and "Controls". Initializers are objects created and run after the node and protocol objects are created, but before the actual simulation itself is executed. As the name suggests, their purpose is to initialize the state of the network nodes and their protocol stacks as needed. Controls are objects that can access any part of the network during the simulation itself. They can be defined to accomplish any number of tasks: monitor the network to gather metrics and evaluation data; change the network nodes' state to simulate churn; call node and protocol functions to imitate external stimuli.

5.1.1.1 *Adaptation of Peersim-Pastry*

Since the architecture proposed here adapts the existing Pastry [14] protocol for MMOG communications, an open source and publicly available implementation of the Pastry protocol, called Peersim-Pastry [38], was used as a template for this proof of concept. The Peersim-Pastry implementation was thoroughly analysed and brought back to traditional Pastry specifications before being refactored and extended with the components of this system. The final implementation is used for the final evaluations of the routing architecture.

5.1.2 Protocol Stack Configuration

The protocol stack defined for nodes in the PeerSim based proof of concept are a basic network layer, basic transport layer, and two user defined application layer protocols. The user defined protocols here refer to a game simulation protocol and the proposed MMOG overlay protocol. While the game simulation is explained at length in Section 5.2, in brief its purpose is to simulate MMOG game play and initiate overlay communications. Messages

sent by the MMOG routing overlay protocol are first passed to the underlying transport protocol, which is chosen to be connectionless but ensures reliable delivery of packets, similar in concept to the networking libraries discussing in Section 4.6. From the transport layer, messages go to the network layer protocol which introduces uniform random delay as it delivers them to their destination node.

5.1.2.1 *Effect of Reliable Transmission on Overlay Messaging*

As stated, the transport layer protocol underlying the MMOG communications protocol in the simulation is assumed to be to a distinctly separate layer that provides reliable transmission of messages. The messages represented in the evaluations thus represent those solely generated by the MMOG communications protocol that implementing this architecture. Optimizations have been made with regards to this, such as routing peer notification messages piggybacking on state table messages that are sent to the same destination at the same time. In cases such as these, the piggybacking notification is not logged as a separate message. On the other hand, if the underlying transport layer and connection session/link handler is integrated closely with the overlay implementation, the optimizations that can be made in such a scenario may differ and thus show a slightly different evaluation footprint. As example of this would be when a peer tears down a transport level connection, it could be interpreted as a routing peer notification of peer removal and node disconnect, and thus these notifications would not be sent.

5.1.3 FreePastry

Though the PeerSim based simulation is the main vehicle for performing simulations and gathering performance evaluation data on this architecture, initial research in location aware routing was performed with FreePastry [39], a Java based, open-source implementation of the Pastry protocol. FreePastry is a heavy-weight implementation of Pastry that can be deployed in real networks. It also includes a simulator specific to its design and protocol, which was used to evaluate the effect on routing efficiency when using the Hilbert Curve to generate location aware DHT-IDs.

5.2 Game Simulation

The game simulation protocol at the application layer of the PeerSim simulation is implemented according to the following specifications. The game simulation protocol integrates closely with the MMOG routing protocol, similar to a game engine, in order to initiate DHT-ID changes and AOI updates according to the proposed architecture.

5.2.1 Environment

The virtual environment emulated within the simulation is defined by several parameters. The first item is the size of the three dimensional environment; each dimension is assumed to have a lower bound of zero, while the upper bound is defined by the user. A number of cluster centers, or hotspots, are uniformly randomly distributed in the XY plane of the defined space. A probability parameter is then used to define how many of these hotspots on the XY plane will have object distribution in the Z-dimension as well. If a given hotspot is selected for this, a normally random number is used to define the height of the hotspot in the Z-dimension. These Z-dimension hotspots are called building hotspots.

5.2.2 Player Distribution

For player distribution, the total number of players are divided up evenly and 'assigned' to a hotspot. All players' initial locations, in terms of the XY plane, are normally distributed around their respective hotspot. A percentage of the players assigned to building hotspots will be up in the building itself, with these players' Z-dimension location values also being normally distributed about the building hotspot's Z-dimension value. Once a player is within a building, they are subject to building walls at all heights. Their XY locations are forced to be within the building walls if they are outside the defined area of the building. The clustering of players at the building walls can be attributed to the attraction of window views. All non-building players remain on the ground level, but do have slight variations in their Z-dimension values, which are meant to represent jumping players, environment ground elevation and steps. These ground level players are split into two additional groups of circling players, also known as 'circlers', and travelling players, or 'travellers'. The specifics of these groups are illuminated in the following section.

5.2.3 Player Movement

For movement, each player is given a destination to move towards, and they do so in a straight line. When a player reaches their intended destination, they are then assigned a new location. Each destination is generated by the same algorithm as the initial player location described above. This algorithm is reused in order to maintain a comparably similar distribution at all times. All player speeds are normally distributed about a mean of 1 unit (e.g. meter) per second. Further specifics of the three player types are as follows:

- Building players' new destinations are generated according to same building hotspot they are currently assigned to. The building player speed distribution used has a standard deviation of 0.3, so the spread in speeds is smaller than that of ground-levels players. It is acknowledged that having building players travel straight toward they destination is not truly accurate to reality, as they are passing through the physical floors within the building. In comparison with the semantics needed to model elevators and stairs in buildings however, this simpler level of granularity is acceptable as complex movement simulation is not the focus of this work.
- Traveling players are given a new destination at a hotspot other than the one they are currently assigned to. The traveling player speed distribution uses a standard deviation of 0.5, giving a spread of speeds that sees some travellers amble towards their destination, while others may achieve speeds comparable to jogging. Once a traveller reaches their destinations, they assign themselves to the new hotspot and change their type to become ground circlers.
- Ground circlers are called so because they are meant to circle around their assigned hotspot, so their new destinations are generated accordingly. Just as travellers can become circlers, a circler can change its type to a traveling player. This change occurs to a single circler when a single traveller changes its type, thus generally maintaining the number of players per class over time. Circlers use the same distribution as building players for speed.

5.2.4 Simulation Applicability

The coupled game environs and player movement defined above attempt to emulate characteristics of a generic cityscape. The hotspots on the XY plane represent points of interest to players, such as upgrades, usable items, mission/quest goals, social gathering points, or anything relevant to the presented reality. Player movement, though simplistic, is defined in such a way that generally universal MMOG behaviours are captured; players travel towards and around items of interest. Having players travel between hotspots also captures the fact that their attention sways between local and non-local items of interest.

Furthermore, the probability of hotspots having Z-dimension distribution is also relevant to the environment setting. For instance a modern metropolis, or futuristic ecumenopolis, would configure a majority of hotspots to have Z-dimension object distribution, thus creating buildings and skyscrapers. Fantasy or historical style virtual environments, with fewer technological advancements, would have fewer of these skyward reaching turrets and towers.

While the aforementioned buildings and towers are emulated for the presented evaluations, additional configurations can be derived for alternative Z-dimension usage. Use of different parameters and distributions for hotspot height, as well as initial player distribution, will have an effect on the end simulation. Thus varying the parameters and distributions can be done in order to more closely match different target environments. An example of this is Figure 21, which is achieved by varying the parameters used for the distributions in environment generation and player location assignment. As can be seen from the figure, the distribution of players in the Z-dimension does not extend down to the XY plane, but 'float' in clusters. Such a scenario could be that of a militaristic simulation, with infantry on the ground and airborne fighters engaging in aerial dogfights. Of course, the algorithm for player movement presented here would have to be slightly re-designed to accurately portray such a scenario.

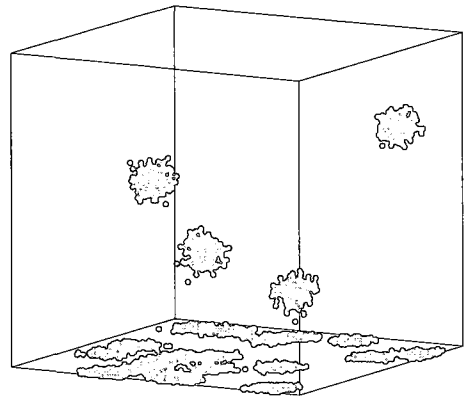


Figure 21: Player Distribution with Aerial Clustering

Chapter 6 - Performance Evaluation

The evaluation results of the described proof of concept are presented here, beginning with the initial research on location-aware DHT-ID assignment with the non-uniform Hilbert Curve and subsequent routing that was performed on the FreePastry [39] platform. This is followed by results that show the efficacy of the designed movement accommodation mechanisms. AOI updates are also evaluated, but in a two-fold manner; the results of the networking aspects are shown, as well as the efficiency of the range query algorithm itself with its optimizations.

6.1 Location Aware DHT-ID Assignment

In order to visually show the effectiveness of the non-uniform Hilbert Curve for DHT-ID generation, both standard and non-uniform Hilbert Curves were used to map the population distribution analogous to Figure 10 to the DHT-ID space. The mapping was carried out in a virtual environment of 10000 nodes that were gathered in 50 hotspots, 25 percent of which were defined as towers. Hotspots and nodes within the environment were distributed according to sections 5.2.1 and 5.2.2 respectively. The resulting distribution of DHT-IDs when mapped to the standard uniform and proposed non-uniformed Hilbert Curves are shown in Figure 22 and Figure 23 respectively.

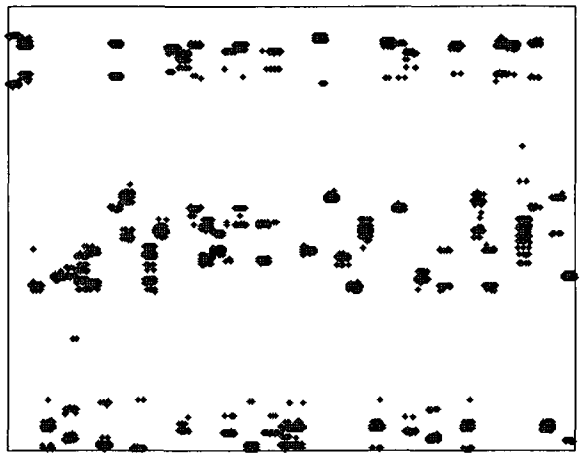


Figure 22: Uniform DHT-ID Distribution

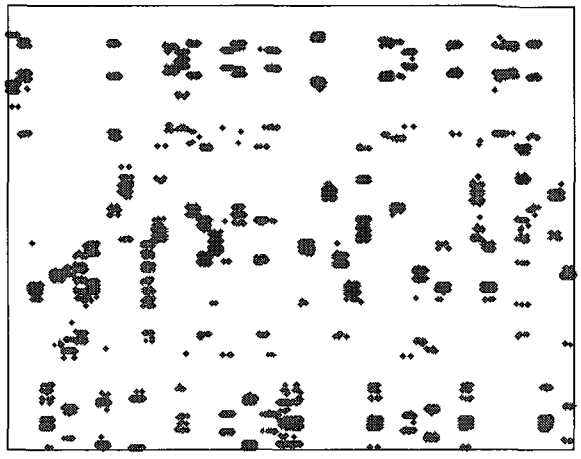


Figure 23: Non-Uniform DHT-ID Distribution

The obtained results visually demonstrate the fact that the non-uniform Hilbert Curve can successfully populate unused regions of DHT-ID space, thereby reducing the collisions in other regions. In order to further evaluate the effectiveness of the non-uniform Hilbert Curve, the number of collisions for different order Hilbert Curves is shown for both cases in Figure 24.

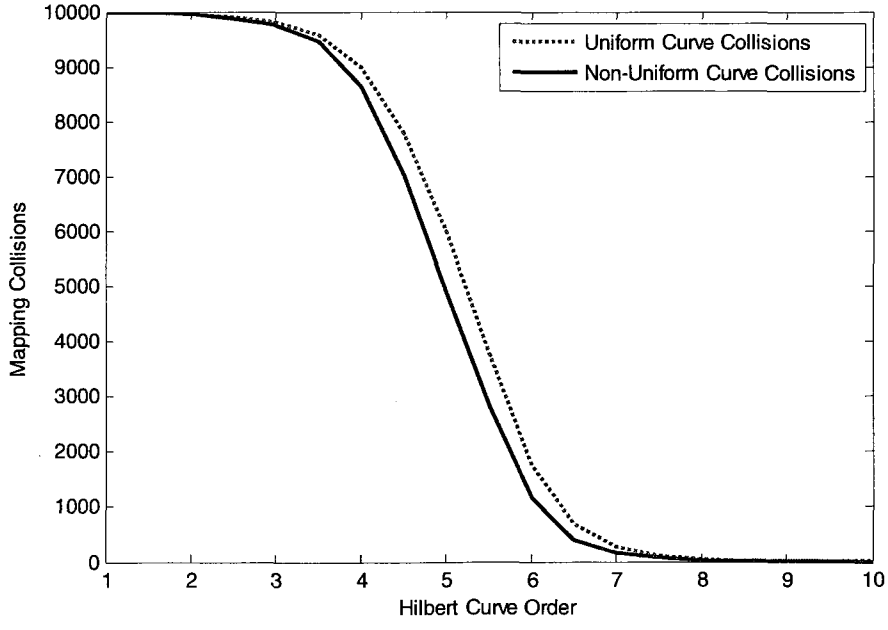


Figure 24: DHT-ID Collisions for Standard & Non-Uniform Curves vs. Curve Order

The X-axis, which indicates Hilbert Curve order, has extreme cases in mapping collisions at its lower and upper bounds. When the curve order is small, there are obviously an insufficient amount of vertices available to map to 10000 nodes. Non-uniform distribution cannot possibly aid in this situation, as demand far outstrips supply. This scenario should never be encountered if properly accounted for in the game design phase. The reverse applies at the other end with the higher order curves; the number of vertices available in any given area is more than sufficient for the present nodes. While this demonstrates that simply increasing the Hilbert Curve order can address DHT-ID requirements, this is an inefficient practice as it results in DHT-IDs remaining unused. In addition to this, it is important to remember the exponential nature of the curve; if an extremely high order curve is required to satisfy DHT-ID requirements, the associated computations become similarly complex. In all other scenarios however, the number of collisions decreases considerably by application of the non-uniform Hilbert Curve.

The efficiency gain of location-aware routing through the use of Hilbert based DHT-IDs is also determined by applying location-to-Hilbert mapping to FreePastry [39], an open-source implementation of Pastry [14]. In a similarly distributed environment with 2500 nodes, nodes were simulated to deliver the majority of their messages to nodes within their locale of the virtual environment. Message hop counts for uniform random hash-based DHT-ID assignment and Hilbert mapping for DHT-ID assignment were captured. The hop count probability distributions for both DHT-ID assignment techniques are shown in Figure 25. The comparative distributions in the figure establish that locale-sensitive DHT-ID assignment in an overlay network contributes significant efficiency gains for MMOG scenarios, even in an overlay network not specifically designed for its use.

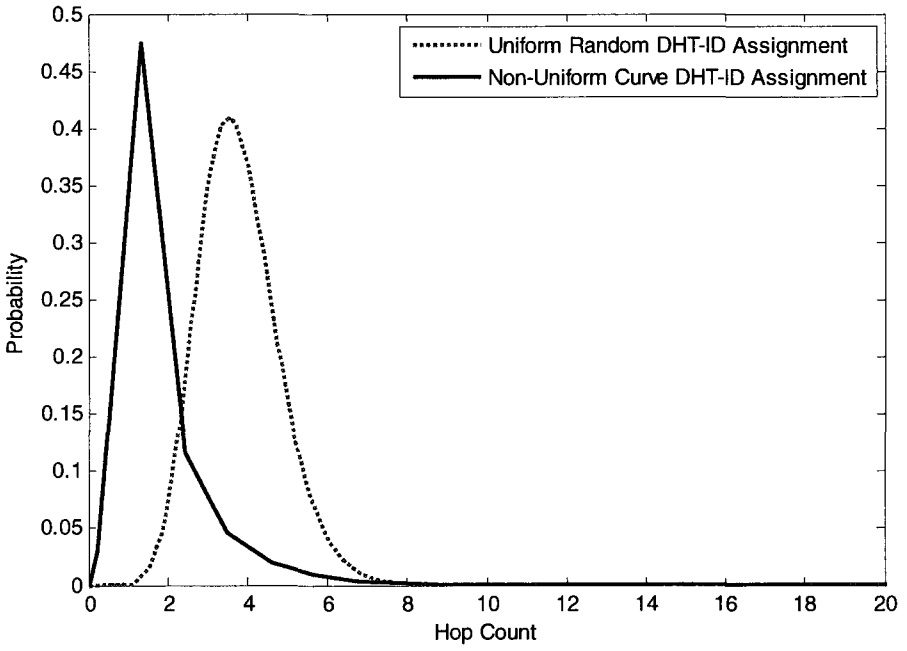


Figure 25: Hop Count Probability Distribution

6.2 Player Movement

The proof-of-concept was simulated in order to determine the performance of the architecture in the face of player movement. The overlay configuration was chosen to be b=4, L=32 and M=16; the leaf set was given greater weight due to its overall importance in the architecture. Players moved in a 100x100x50 unit (meter) environment with 50 hotspots, which were initialized according to sections 5.2.1 and 5.2.2 respectively. The DHT-ID threshold radius was chosen to be 3 units (meters), and the environment was spanned by a 10^{th} order non-uniform Hilbert Curve for DHT-ID mapping.

Two aspects of the player movement accommodation were evaluated with respect to each other, establishing a baseline network message load associated with overlay upkeep due to player movement. To this end, five minutes of game time were simulated, with the first few seconds dedicated to building the initial network connections between nodes without movement. The messages and time associated with the construction of the initial static overlay were removed from the performance calculations in order to gain a clear picture of the on-going load due to movement alone. As previously discussed however, simulation is

computationally very expensive. For this reason, full movement and overlay protocol simulation was only able to be accomplished for up to 125 active nodes.

6.2.1 Standard vs. Abbreviated Join Procedure

The first aspect of the movement accommodation analysed is the relative performance between basic overlay node duplication, which employed the standard join protocol, and the abbreviated re-join procedure used to reassign DHT-IDs to player nodes in the overlay. The observed results are very interesting, as seen in Figure 26.

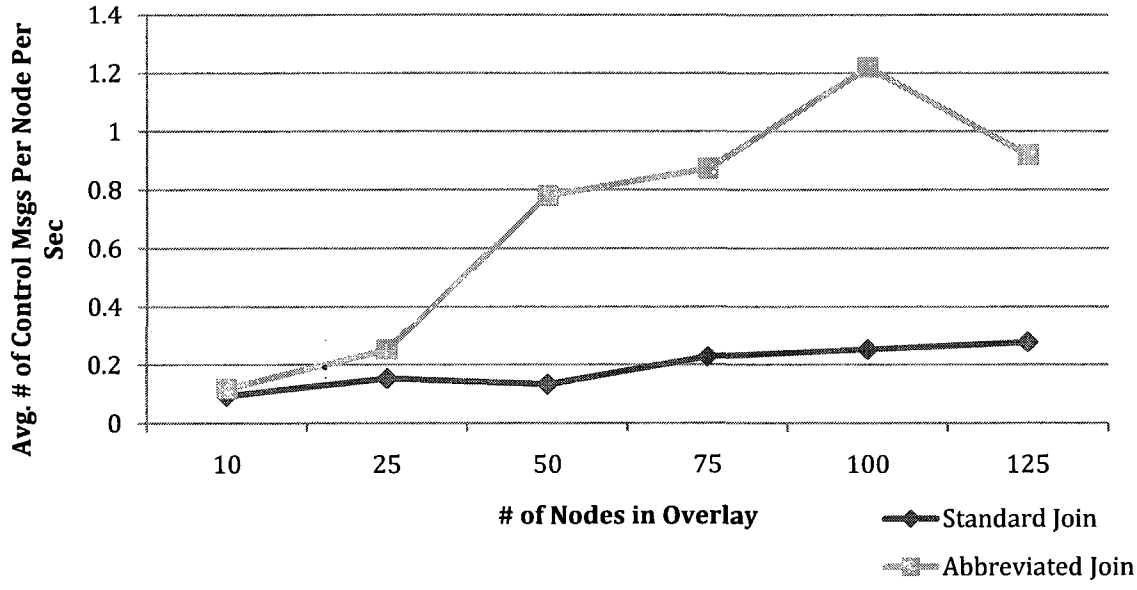


Figure 26: Comparison of Join Protocol Network Usage

As the figure shows, the abbreviated re-join procedure actually increases the average number of messages sent per node, which is contrary to the expected decrease. The performance of overlay node duplication for DHT-ID reassignment places a relatively small message load on the nodes across the overlay. This load is also rather consistent, remaining fairly steady as the number of nodes within the overlay increases, which is highly favourable for scalability.

Closer analysis of the messages within the simulations of the abbreviated re-join procedure shows a greater amount of repair procedure messages exchanged between nodes. It can thus be concluded that the standard join protocol, which sends the DHT-ID change request to a node with network proximity rather than DHT-ID proximity, is well suited to the

construction of the structured overlay. The ensuing state table exchange by the join procedure also ensures that well formed mutual awareness is achieved between nodes. By abbreviating the join procedure, nodes' state tables are not as well constructed and thus require greater repair. More importantly however, is that this suggests that the repair procedures within the traditional design of Pastry are not well suited to scenarios with DHT-ID reassignment required by MMOGs.

6.2.2 Standard vs. Selective Peer Notification

Having seen that the standard join protocol remains an efficient mechanism for DHT-ID reassignment, the accompanying suite of consistency mechanisms is now considered. Specifically, the standard and selective schemes for peer notification are compared for their network load in Figure 27 and resulting routing performance in Figure 28.

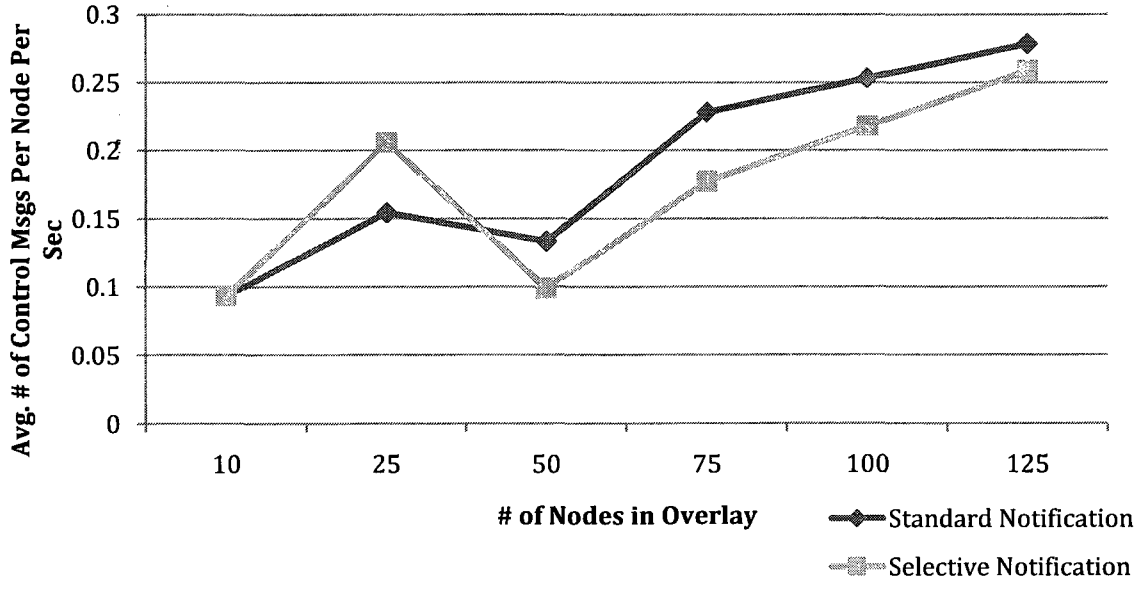


Figure 27: Comparison of Standard and Selective Notification

As can be seen from Figure 27, selective notification is successful in reducing the network load experienced by peers. The equal and reversed placements seen for smaller populations are not representative of the overall trend, as the ratio of filled to empty entries within the state tables is very low. As the number of peers increases and local state tables become filled, so too does the number of notifications start to cap. Once this occurs,

selectively sending notification does lead to fewer messages being dispatched, which leads to fewer repair procedures being initiated as well.

Though the purpose of selective notification is to decrease the number of messages sent by nodes, this is slightly at odds with achieving a well formed overlay. Selective notification delays the repair of state tables if not deemed entirely necessary. While lazy notification is used when non-updated links between peers are used, a possible overall effect might be a slightly less efficient routing of messages. This is not the case however, as Figure 28 demonstrates that selective notification does not unduly affect the routing efficiency of the overlay.

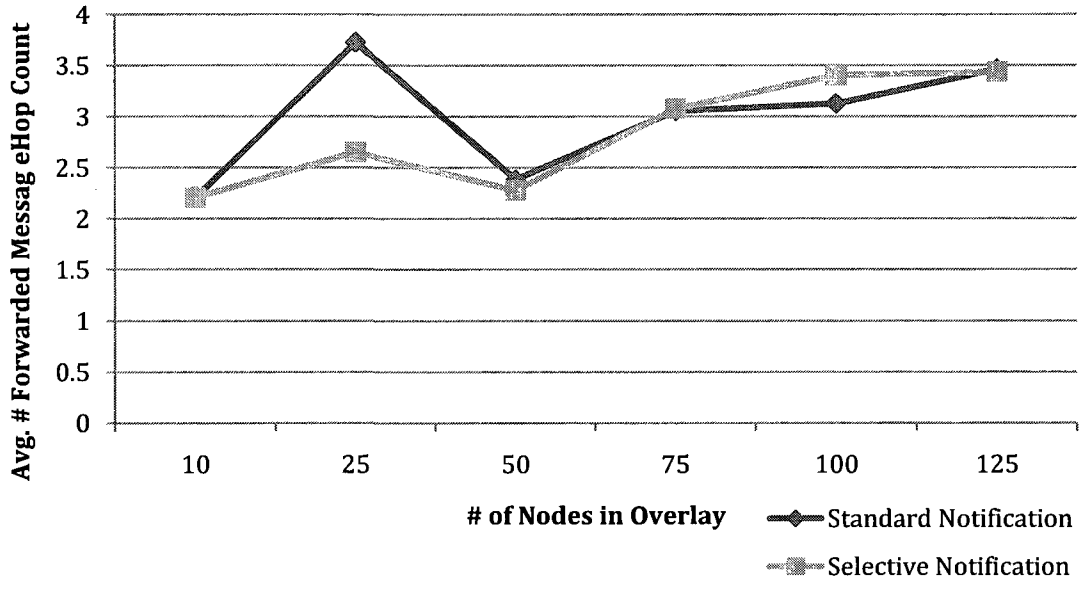


Figure 28: Effect of Notification on Hop Count

Even as the number of participating nodes increases, the average hop count experienced by forwarded messages remains extremely competitive to the performance in the standard notification scheme. Note that this average hop count value does not take into its calculations messages types which are sent directly from the sender to the receiver, as the receiver's address is known and requires only a single hop to deliver.

6.3 Area of Interest Updates

6.3.1 Hilbert Curve Range Query

The Hilbert Curve range query was executed to evaluate the performance of the base line algorithm and its two optimizations. The range query algorithm is written and compiled with the standard Java compiler, and does not have any dedicated hardware usage. The single execution thread was run on a single core of an Intel Core Duo running at 1.73 GHz with a maximum of 512 megabytes of memory available to the virtual machine.

For consecutive orders of the curve, a range query was performed for a 10 cube-unit area in a 100 cube-unit environment. The non-uniform Hilbert Curve was used with normal distribution parameters of $\mu=0.1$ and $\sigma=0.5$ for the non-uniform ratio distribution function. The two parameters used to evaluate the performance gains from the optimizations are memory usage and execution time. As the first optimization, range representation, targets inefficient memory usage, this is examined first in Table 5.

Table 5: Range Query Memory Usage

Curve Order	Memory Allocated for DHT-IDs (bytes)		
	Baseline Algorithm	with Range Representation	with Enveloped Branch Detection
1 - 4	80	80	80
5	1000	720	720
6	6616	3072	3072
7	53272	3072	3072
8	579184	73912	73912
9	4007040	134864	134864
10	26450392	1029128	1029128
11	> 536870912*	1836376	1836376
12	> 536870912*	8925568	8925568

For the first few orders of the curve, the memory usage remains constant due to the fact that there are no actual DHT-IDs within the query range. In subsequent orders greater than 4, the range representation does reduce the overall memory usage by a signification amount compared to the baseline algorithm. In fact, the baseline memory usage becomes so great for curve orders greater than 10 that it exceeds the memory available to the algorithm and is not able to complete its execution. The range representation circumvents

this issue completely, as the algorithm is able to fully execute with the optimization. As expected, enveloped branch detection does not affect the memory footprint. The full effect of the memory savings from the range representation can be seen by the percentage of memory reduction graphed in Figure 29, which is steadily reduced by approximately 90% or more for curve orders greater than 7.

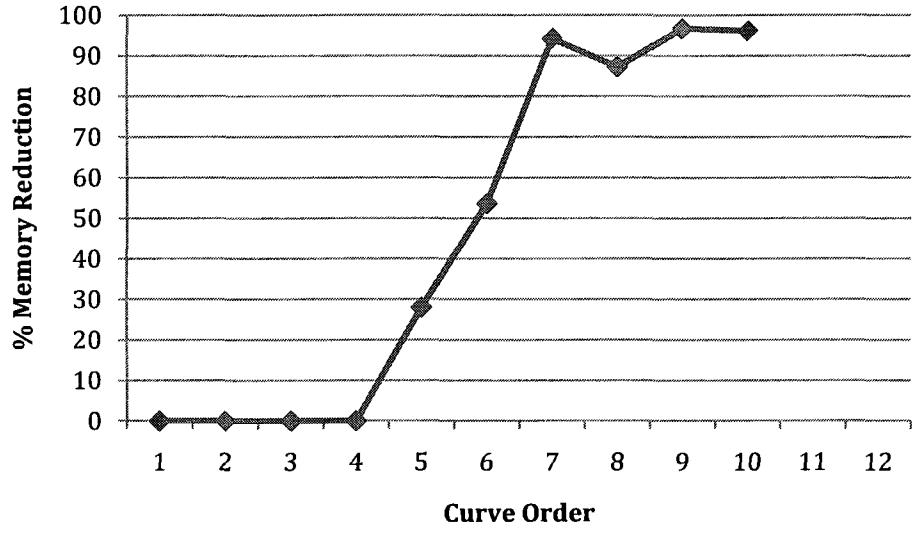


Figure 29: Memory Reduction from Range Representation

Execution times for the baseline and optimized algorithm are tabulated in Table 6; the curve orders lower than 8 do not have any true significance as the values are minimal and are more an accumulation of function calls and context switching. Curve orders of 8 and greater however show some interesting results. Range representation not only has an effect on memory usage, but shows to positively impact execution time as well, likely due to the fact that every DHT-ID is no longer individually computed. Once again, baseline execution times for curve orders over 10 are not given as the algorithm violated available memory.

Table 6: Range Query Execution Time

Curve Order	Execution Time (ms)		
	Baseline Algorithm	with Range Representation	with Enveloped Branch Detection
1 – 4	0	0	<15
5	0	31	16
6	0	15	16
7	31	31	16
8	375	313	297
9	3016	2234	625
10	26968	19453	6031
11	N/A	157078	11813
12	N/A	1367516	58500

Enveloped branch detection leads to more significant performance gains, especially in increasingly higher order curves. This is due to a greater number of higher order curve components fitting within the same volume, which the optimization is made to quickly detect and circumvent full calculation of. Figure 30 shows the speedup of the algorithm, calculated with respect to the range representation execution times in order to be able to compare the higher order curves; note that the speedup over the baseline algorithm would be even greater.

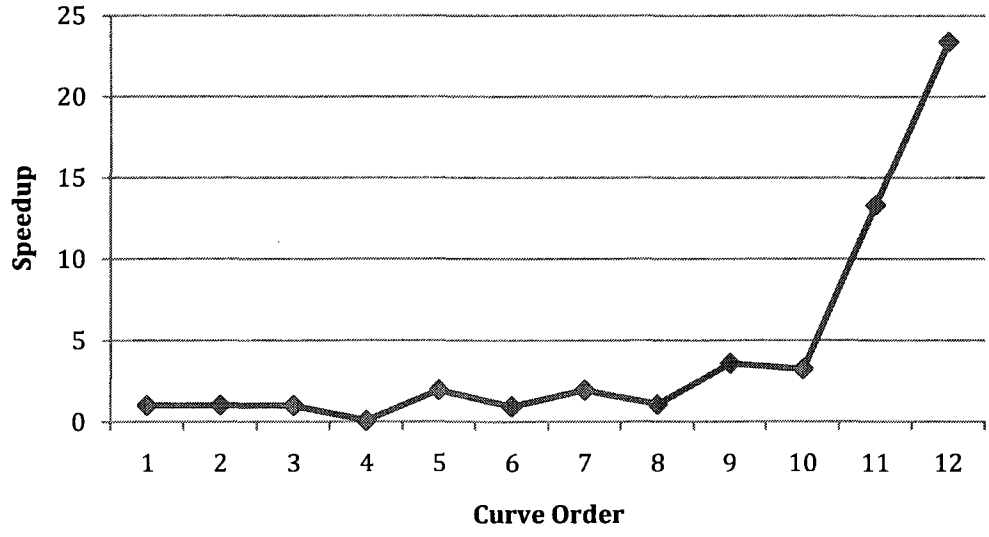


Figure 30: Execution Speedup from Enveloped Branch Detection

6.3.2 Update Request Propagation

With the range query algorithm able to calculate the potential DHT-IDs within an area/volume, the performance of contacting the live DHT-IDs within said area or volume is of interest. While the proposed design speaks of single versus double-sided request propagation, it is found that the binning value that aggregates DHT-IDs into request ranges has the greatest affect on the performance of the system. The number of messages sent in a peer discovery process is dependent on the binning value and the Hilbert Curve order. The true effect of binning was determined within the simulation parameters of section 5.2, with 125 active nodes but with the environment spanned by an 8th order Hilbert Curve as opposed to a 10th order curve.

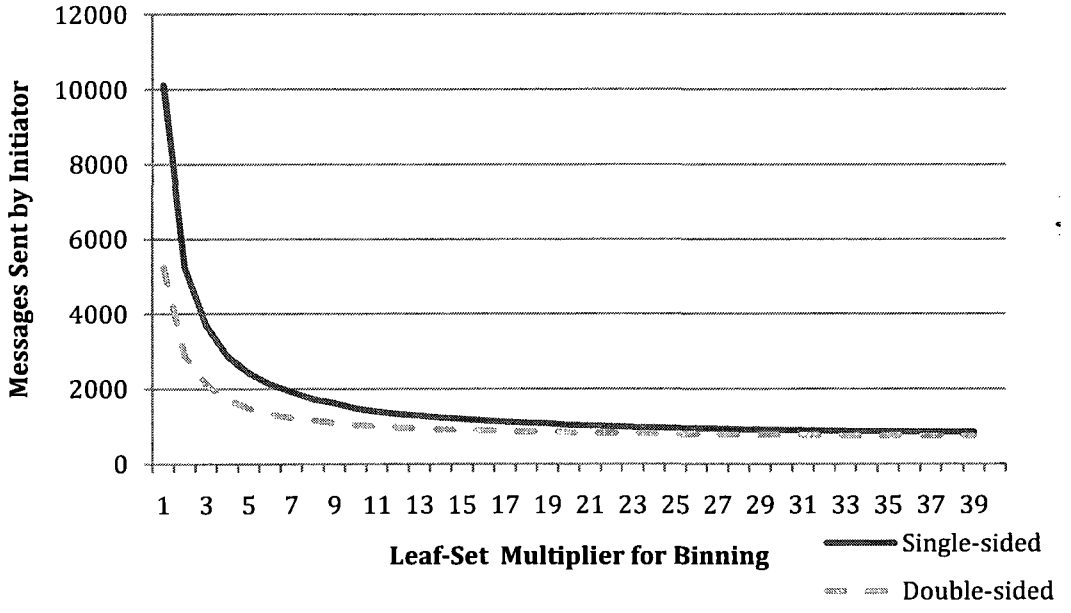


Figure 31: Messages Dispatched for Varying Leaf-Set Multipliers

Figure 31 shows the average number of messages sent by a node given a range of leaf-set size multipliers that determined the DHT-ID binned range size. Even for an environment with a relatively low 8th order curve and player AOI radius defined as 5 units, it is clear that small multiplier values, and by extension small binning sizes, require an inordinate amount of messages to contact the potential DHT-IDs. Thus large range sizes, on the order of hundreds to thousands of DHT-IDs binned together, are necessary in order to accomplish peer discovery without exhausting network bandwidth resources. This directive becomes

imperative as the curve order increases because the number of potential DHT-IDs to contact rises exponentially. It should be noted that the amount of data within in a peer discovery message is extremely small in comparison to other messages, such as those containing state table information.

To evaluate the proposed single-sided and double-sided propagation approaches on a larger scale, 2500 nodes were simulated in the environment with a 10^{th} order Hilbert Curve. In order to accomplish this however, player movement and its associated mechanisms were disabled to create a static network topology that could be simulated within the available computational and memory resources. The purpose was to determine whether nodes received duplicate request messages, and the results showed that neither approach delivered duplicate requests to a single node. The maximum number of hops required to contact a node within a peer's AOI does not seem overly affected by binning size or propagation approach, as it oscillates between 2 and 4 hops in Figure 32. It is likely that this would be affected if the player density within the area increased, as well as by the chosen leaf set size. This is not worrisome however, given the results thus far and the fact that peer discovery is not relied upon for game update dissemination. Longer peer discovery completion can easily be countered through the use of pre-emptive AOIs.

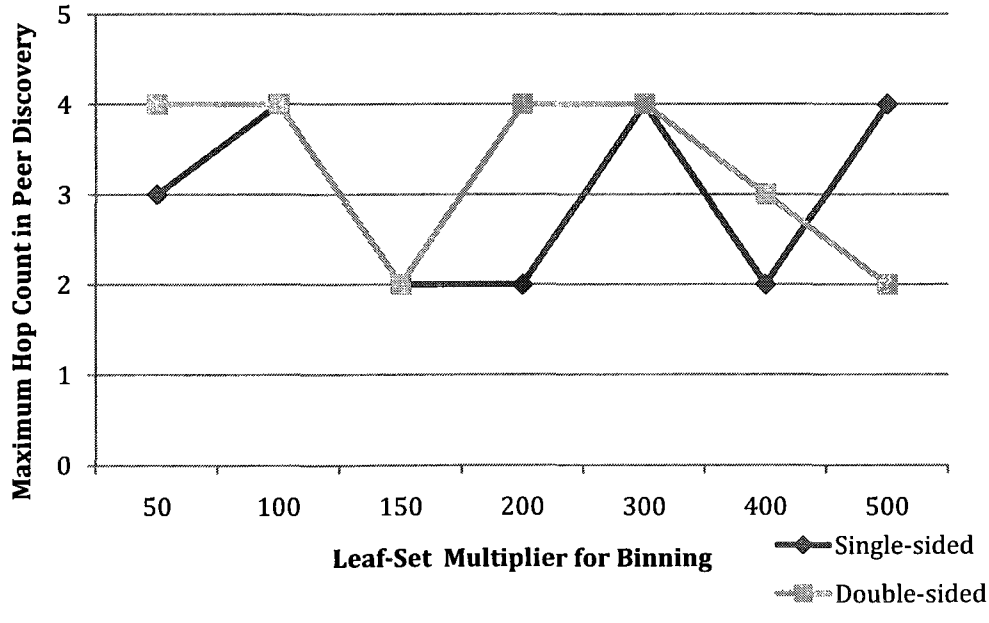


Figure 32: Peer Discovery Completion in Hops

Chapter 7 - Conclusion & Future Work

Taking into account the increasing trend of MMOG popularity among players, combined with the potential application of MMOGs in a variety of scenarios, it would be advantageous to have these environments supported by scalable and fault-tolerant infrastructure. P2P systems are emerging in ever-increasing scenarios, and the domain of MMOG networked communications is fertile ground for benefiting from the application of P2P concepts and technology.

This thesis presents a distributed P2P communication architecture customized for MMOGs. Communication between peers is optimized with respect to virtual environment locality, achieved through the use of the Hilbert Space Filling Curve. To this end, a novel non-uniform Hilbert Curve is proposed, with an associated fast location-to-Hilbert mapping algorithm used for generation of location based DHT-IDs. In order to ensure that the architecture continues to reflect player location within the environment, movement is matched by running a proposed DHT-ID reassignment procedure. A suite of consistency mechanisms accompany the movement accommodation scheme in order to maintain a proper overlay network configuration. An AOI update process that achieves 1-hop latencies over the network is also detailed, based on an optimized Hilbert Curve range query algorithm.

A proof-of-concept implementation of the architecture is given and evaluated through simulation. Performance of the baseline architecture is shown in comparison to the added optimizations in order to demonstrate the effect on system performance. The final results shows promise that this architecture can be pursued further in order to provide a basis for live deployment of MMOGs.

7.1 Avenues of Future Research

While the presented design accomplishes location-aware routing, movement accommodation and AOI update in a completely decentralized manner, assumptions

otherwise have been made with regards to initial bootstrapping. Bootstrapping is outside the scope of this particular work, but it is assumed that there is some aspect of initial rendezvous that is provided in a centralized manner by the owner/developer/manager of the MMOG in question. A server known to all game instances can, with some form of connection to the overlay, lookup a candidate peer as a bootstrap for newly joining players. While not as resource intensive as other MMOG server tasks, this method of initial rendezvous shares the same issues with any centralized architecture: scalability in the face of increased connectivity; lack or cost of backups in the face of network failure. Bootstrapping thus remains an open question for P2P systems.

Other avenues that can be explored by future research rest in improving the performance of this system. In the consistency mechanisms coupled with player movement, peer notification is presented as a method of reducing the lag time between an invalidated state table entry and its subsequent detection. As traditional Pastry's design includes specification for state table repair, these are relied upon as the response to an invalidated state table entry. They are not, however, made to be called at the high rate present in this architecture as evidenced by the effect of their use from the abbreviated join procedure. A clear potential for improvement is to redesign these repair procedures for the high rate of use seen in this architecture. A possible approach is to include suggested replacement candidates for the invalidated state table entries within the notification itself. The question with this approach becomes advanced tracking and algorithms for determining the optimal candidates to suggest, as many metrics can be tracked for this purpose, such as player location, displacement over time and network proximity.

Investigating this system's hybridization with server resources is another possible method to increase performance with regards to certain aspects of routing. Traditional Pastry employs cryptographic hashing to generate uniform random DHT-ID. This is done in order to achieve an even distribution of active DHT-IDs for the purposes load balancing the network. As this system attaches location semantics to DHT-ID values, an even distribution of active DHT-IDs rarely transpires. Servers can be employed to counteract this phenomenon and achieve load balancing within the routing overlay. Servers would connect to the overlay with DHT-IDs within the sparsely populated regions of the DHT-ID space, in

order to create a more homogeneous distribution of links between peers. As the sparely populated areas may experience low message load, the servers may then be used as virtual servers [40] in other areas of the space. The concept of virtual servers is where the server manifests itself as multiple nodes within the routing overlay, which makes more efficient use of the server resources when they are under low load. If the total load experienced by the server increases, it can reduce the number of virtual servers it manifests. The overhead incurred by virtual servers would only be the cost of joining the overlay, as they would not change their DHT-ID. Moreover, it is possible that servers' dedicated resources could also be employed for cheat detection. As a transparent participant in the overlay, servers would be able to perform the AOI game update request process in order to receive players' game updates. They can then confirm the validity of players' actions and identify cheaters.

Closer integration with the game engine may also prove to be fruitful, as information known by the game application can be of use in determining configurable values for the routing architecture. For instance, when the peer discovery process sends its multiple requests, each is destined to cover a specific range of DHT-IDs. This range is currently a static binning value, and the overall efficiency of the process is affected by this value and the number of active nodes that exist in the overlay. If the game knowledge of nearby players from game updates is processed to determine the density of nodes within the overlay, this binning value can be reconfigured automatically as needed to improve the efficiency of the peer discovery process.

References

[1] World of Warcraft: Blizzard Inc., 2004. [Online]. http://www.worldofwarcraft.com.

[2] B. S. Woodcock. (May, 2008) MMOGChart.com. [Online]. Available: www.mmogchart.com. [Accessed: Dec. 16, 2009].

[3] BBC News. (2007, August) Virtual game is a 'disease model'. [Online]. Available: http://news.bbc.co.uk/2/hi/health/6951918.stm. [Accessed: Dec. 16, 2009].

[4] S. Kumar, J. Chhugani, C. Kim, D. Kim, A Nguyen, P. Dubey, C. Bienia, and Y. Kim, "Second Life and the New Generation of Virtual Worlds," *Computer*, vol. 41, no. 9, pp. 46-53, September 2008.

[5] Second Life: Linden Research, 2003. [Online]. http://secondlife.com/.

[6] D. T. Ahmed, S. Shirmohammadi, and J. C. de Oliveira, "A Hybrid P2P Communication Architecture for Zonal MMOGs," *Journal of Multimedia Tools and Applications, Springer*, vol. 45, no. 3, pp. 313-345, 2009.

[7] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Geoganas, "A Survey of Application-Layer Multicast Protocols," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 3, pp. 58-74, 2007.

[8] S. Ratti, B. Hariri, and S. Shirmohammadi, "A Survey and Analysis of First Person Shooter Gaming Traffic on the Internet," *IEEE Internet Computing*, (accepted, to appear), 2010.

[9] S. Ratti, B. Hariri, and S. Shirmohammadi, "NL-DHT: A Non-uniform Locality Sensitive DHT Architecture for Massively Multi-user Virtual Environment Applications," in *Proceedings of 14th IEEE International Conference on Parallel and Distributed Systems*

(ICPADS '08), Melbourne, Australia, December 2008, pp. 793-798.

[10] B. Hariri, S. Ratti, S. Shirmohammadi, and M. R. Pakravan, "A Distributed Latency-Aware Architecture for Massively Multi-User Virtual Environments," in *IEEE Internation Workshop on Haptic, Audio And Visual Environments and Games (HAVE 2008)*, Ottawa, Canada, October 2008, pp. 53-58.

[11] R. Iqbal and S. Ratti, "A Distributed Camera Network Architecture Supporting Video Adaptation," in *Proceedings of 3rd ACM/IEEE International Conference on Distributed Smart Cameras*, Como, Italy, 2009.

[12] S. Ratti, C. Towle, P. Proulx, and S. Shirmohammadi, "FizzX: Multiplayer Time Manipulation in Networked Games," in *Proceedings of 8th Workshop on Network and Systems Support for Games (ACM NetGames)*, Paris, France, 2009.

[13] J. Parri and S. Ratti, "Trigonometric Function Approximation Neural Network Based Coprocessor," in *Proceedings of 2nd Microsystems and Nanoelectronics Research Conference*, Ottawa, Canada, 2009, pp. 148 - 151.

[14] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329-350.

[15] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth Content Distribution in a Cooperative Environment," in *Proceedings of 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, USA, 2003.

[16] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The Design of a Large-Scale Event Notification Infrastructure," in *Proceedings of 3rd International Workshop on Networked Group Communication*, London, UK, 2001, pp. 30-43.

[17] P. Druschel and A. Rowstron, "PAST: A Large-scale, Persistent Peer-to-peer Storage Utility," in *Proceedings of Workshop on Hot Topics in Operating Systems*, Elmau/Oberbayern, Germany, 2001.

[18] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting Network Proximity in Peer-to-Peer Overlay Networks," Technical Report MSR-TR-2002-82, 2002.

[19] M. Knoll and T. Weis, "Optimizing Locality for Self-organizing Context-Based Systems," *Springer LNCS*, vol. 2124, pp. 62-73, 2006.

[20] T. Bially, "Space-filling curves: Their generation and their application to bandwidth reduction," *IEEE Transactions on Information Theory*, vol. 15, no. 6, pp. 658-664, November 1969.

[21] A. R. Butz, "Alternative algorithm for Hilbert's space-filling curve," *IEEE Transactions on Computers*, vol. C-20, no. 4, pp. 424-426, April 1971.

[22] G. Jin and J. Mellor-Crummey, "SFCGen: A framework for efficient generation of multi-dimensional space-filling curves by recursion," *ACM Transactions on Mathematical Software*, vol. 31, no. 1, pp. 120-148, March 2005.

[23] J. Lawder, "The Application of Space-Filling Curves to the Storage and Retrieval of Multi-dimensional Data," Ph.D. dissertation, Birbeck College, University of London, London, 1999.

[24] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A Distributed Architecture for Online Multiplayer Games," in *Proceedings of the 3rd conference on Networked Systems Design & Implementation (NSDI'06)*, San Jose, California, 2006, p. 12.

[25] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," in *Proceedings of 2004 ACM SIGCOMM Conference*, Portland, USA, 2004, pp. 353-366.

[26] L. Chan, J. Yong, J. Bai, and B. Leong, "Hydra: A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer," in *Proceedings of the 6th Annual Workshop on Network and System Support for Games*, Melbourne, Australia, 2007, pp. 37-42.

[27] T. Limura, H. Hazeyama, and Y. Kadobayahi, "Zoned Federation of Game Servers: a Peer-to-Peer Approach to Scalable Multi-player Online Games," in *Proceedings of 3rd Workshop on Network and System Support for Games*, Portland, USA, 2004, pp. 116-120.

[28] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games," in *Proceedings of 23rd Conference of the IEEE Computer and Communications Society*, Hong Kong, March 2004, pp. 96-107.

[29] T. Hampel, T. Bopp, and R. Hinn, "A Peer-to-Peer Architecture for Massive Multiplayer Online Games," in *Proceedings of 5th Workshop on Network and System Support for Games*, Hawthorn, USA, 2006.

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

[31] A. Yu and S. T. Vuong, "MOPAR: a Mobile Peer-to-Peer Overlay Achitecture for Interest Management of Massively Multiplayer Online Games," in *Proceedings of 15th Workshop on Network and Operating System Support for Digital Audio and Video*, Skamania, 2005, pp. 99-104.

[32] ENet, 2009. [Online]. http://enet.bespin.org.

[33] RakNet, Costa Mesa, USA: Jenkins Software LLC, 2008. [Online]. http://www.jenkinssoftware.com/raknet/index.html.

[34] ReplicaNet: Replica Software, 2008. [Online]. http://www.replicanet.com/.

[35] Jr, P. Frisbie, HawkNL (Hawk Network Library): Hawk Software, 2009. [Online].

http://hawksoft.com/.

[36] Torque Network Library: openTNL, 2004. [Online]. http://opentnl.sourceforge.net/.

[37] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, The PeerSim Simulator. [Online]. http://peersim.sourceforge.net/.

[38] f.r.u.x.o.j, Peersim-Pastry: PeerSim-based Pastry Protocol Module Implementation. [Online]. http://code.google.com/p/peersim-pastry/.

[39] FreePastry. [Online]. http://www.freepastry.org/.

[40] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area Cooperative Storage with CFS," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Banff, Canada, 2001, pp. 202-215.

Appendix A –
Criteria for Hilbert Curve Order Selection

The first and most fundamental factor in choosing an appropriate curve order is the number of nodes that the system will include. As vertex indices from the curve become DHT-IDs, the number of vertices acts as the upper limit of the potential number of nodes the system can accommodate. Thus, the curve order chosen must produce a sufficient number of vertices.

$$V = 2^{D \times K} \qquad \text{(A.1)}$$

Equation A.1 states V, the number of vertices in the curve, is determined by a curve order K and number of dimensions D.

$$K_1 \geq (ln(N))/(D \times ln(2)) \qquad \text{(A.2)}$$

Equation A.2 states the minimum value of the curve order required to have a sufficient amount of vertices to accommodate N, the number of potential nodes to participate in the system.

In real world application scenarios, the number of dimensions spanned by the curve would likely be no more than three. Two dimensions may be used in scenarios when floors on a building each have their own communication architecture. For the purpose of MMOG environments, with an urban metropolis or where players have the capability for flight, three dimensions would be appropriate for use.

The second factor in choosing an appropriate curve order is having the vertex density of the curve be on par with the distribution of the nodes in the environment. If nodes are clustered in the environment, a higher order curve will be necessary as the traditional Hilbert Curve's vertex coordinates are evenly distributed in each of its dimensions.

$$delta = d/((2^K) - 1) \qquad \text{(A.3)}$$

Equation A.3 states *delta*, the distance between vertices in a dimension given a curve order K and the distance the dimension spans d. This analysis assumes that the Hilbert Curve is constructed such that it spans the entirety of all dimensions.

$$K_2 \geq (ln[(d/delta) + 1])/ln(2) \qquad \text{(A.4)}$$

Equation A.4 states the minimum curve order such that the smallest distance between nodes in a single dimension is greater than the distance between vertices in the curve.

$$K \geq \max(K_1, K_2) \qquad \text{(A.5)}$$

In order to ensure that requirements from both factors are satisfied, the maximum of the calculated minimum curve orders should be chosen, as stated in Equation A.5.

This curve order satisfies the above requirements, but higher values may be selected for the system. Higher order curves provides a greater number of vertices for use as DHT-IDs, but their use must be balanced against the complexity increase they incur, especially in resource limited scenarios. Care must be taken that the maximum DHT-ID value can be represented by the data types implemented used by the calculation systems without overflows occurring.

Appendix B –

Range Query Sub-Procedures Pseudo-code

The algorithm pseudo-code presented and discussed in 4.5.2.4 calls procedures which are expanded upon in this section. These sub-procedures are assumed to have access to top-level variables and have the ability to call other functions, the definitions for which are given in Table 4 and extended here in Table 7.

Table 7: Additional Definitions for the Range Query Algorithm Sub-Procedures

ssBMax$_{XYZ}$	bit array of size K to hold the upper boundary value in binary
ssBMin$_{XYZ}$	bit array of size K to hold the lower boundary value in binary
curDhtIdB	array of size K×3 to track the current DHT-ID from vertex IDs
indexBuffer	array of size K to hold the current sub-space index of each tree level
stateBuffer	array of size K to hold the current node state of each tree level
invNonUniform(Z)	applies the inverse non-uniform function on value Z
toBitArr(V)	converts the integer value V to a bit array (base 2 representation)
nextState(S)	returns the next Hilbert Curve state node given current state S
makeRange(D, E)	makes a range representation given a DHT-ID D and a range size E

Algorithm 4: Pseudo-code for Conversion to Standard Hilbert Curve

```
1:  procedure normalizeSearchSpace( )
2:      coorPerDim ← (2^K)−1
3:      sectionSize_XYZ ← (ssMax_XYZ − ssMin_XYZ) ÷ coorPerDim
4:      ssBMax_X ← toBitArr(⌈ ssMax_XYZ ÷ sectionSize_XYZ⌉)
5:      ssBMin_X ← toBitArr(⌊ ssMin_XYZ ÷ sectionSize_XYZ⌋)
6:      ssBMax_Y ← toBitArr(⌈ ssMax_XYZ ÷ sectionSize_XYZ⌉)
7:      ssBMin_Y ← toBitArr(⌊ ssMin_XYZ ÷ sectionSize_XYZ⌋)
8:      ssBMax_Z ← toBitArr( invNonUniform(⌈ ssMax_XYZ ÷ sectionSize_XYZ⌉))
9:      ssBMin_Z ← toBitArr( invNonUniform(⌊ ssMin_XYZ ÷ sectionSize_XYZ⌋))
```

Algorithm 5: Pseudo-code for Range Query Algorithm Initialization

```
1:  procedure initializeAlgorithm( )
2:      for each value of i starting at 1 up to and including K
3:          coorBMaxXYZ[i] ← 1
4:          coorBMinXYZ[i] ← 0
5:          boundaryBMaxXYZ[i] ← 1
6:          boundaryBMinXYZ[i] ← 0
7:          g ← (i− 1) × 3
8:          curDhtIdB[g+1] ← 0
9:          curDhtIdB[g+2] ← 0
10:         curDhtIdB[g+3] ← 0
11:     end for
12:     rangeCount ← 0
13:     curRangeLen ← −1
14:     curIndex ← 0
15:     curLevel ← 1
16:     curState ← ROOT_NODE_STATE
17:     boundaryBMax_XYZ[curLevel] ← ssBMax_XYZ[curLevel]
18:     boundaryBMin_XYZ[curLevel] ← ssBMin_XYZ[curLevel]
```

Algorithm 6: Pseudo-code for Upwards Tree Traversal

```
1:  procedure moveUpTree( )
2:      coorBMaxXYZ[curLevel] ← 1
3:      coorBMinXYZ[curLevel] ← 0
4:      boundaryBMaxXYZ[curLevel] ← 1
5:      boundaryBMinXYZ[curLevel] ← 0
6:      g ← (curLevel− 1) × 3
7:      curDhtIdB[g+1] ← 0
8:      curDhtIdB[g+2] ← 0
9:      curDhtIdB[g+3] ← 0
10:     curLevel ← curLevel - 1
11:     curIndex ← indexBuffer[curLevel]
12:     curState ← stateBuffer[curLevel]
```

Algorithm 7: Pseudo-code for Enveloped Branch Computation

```
1:  procedure computeEnvelopedBranch( )
2:      if (curRange ≠ Ø) then
3:          curRange.size ← curRange.size + branchSize
4:      else
5:          indexBits ← toBitArr(curIndex − 1)
6:          g ← (curIndex − 1) × 3
7:          curDhtIdB[g+1] ← indexBits[1]
8:          curDhtIdB[g+2] ← indexBits[2]
9:          curDhtIdB[g+3] ← indexBits[3]
10:         curRange ←makeRange(toInt(curDhtIdB), branchSize)
11:     end if
```

Algorithm 8: Pseudo-code for Downward Tree Traversal

```
1:  procedure moveDownTree( )
2:      indexBits ← toBitArr(curIndex − 1)
3:      g ← (curIndex − 1) × 3
4:      curDhtIdB[g+1] ← indexBits[1]
5:      curDhtIdB[g+2] ← indexBits[2]
6:      curDhtIdB[g+3] ← indexBits[3]
7:      stateBuffer[curLevel] ← curState
8:      indexBuffer[curLevel] ← curIndex
9:      curLevel ← curLevel + 1
10:     curState ← nextState(curState)
11:     curIndex ← 0
12:     boundaryBMaxXYZ[curLevel] ← ssBMax_XYZ[curLevel]
13:     boundaryBMinXYZ[curLevel] ← ssBMin_XYZ[curLevel]
```

Algorithm 9: Pseudo-code for Handling Leaf Computation

```
1:  procedure computeLeafWithinRange( )
2:      if (curRange ≠ Ø) then
3:          curRange.size ← curRange.size + 1
4:      else
5:          indexBits ← toBitArr(curIndex − 1)
6:          g ← (curIndex − 1) × 3
7:          curDhtIdB[g+1] ← indexBits[1]
8:          curDhtIdB[g+2] ← indexBits[2]
9:          curDhtIdB[g+3] ← indexBits[3]
10:         curRange ← makeRange(toInt(curDhtIdB), 1)
11:     end if
```