


# A hybrid peer-to-peer middleware plugin for an existing client/server massively multiplayer online game

by

Darren Armstrong Croucher

*Thesis presented in partial fulfilment of the requirements  
for the degree of Master of Engineering (Research) in the  
Faculty of Engineering at Stellenbosch University*

The crest of Stellenbosch University, featuring a shield with a red and white design, topped with a crown and surrounded by a red and white wreath. Below the shield is a banner with the Latin motto "Pacta sunt ossibus recti".

Supervisor:

Dr. H.A. Engelbrecht

Department of Electrical & Electronic Engineering

April 2014

# Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: ..... April 2014 .....

Copyright © 2014 Stellenbosch University  
All rights reserved.

# Abstract

Massively Multiplayer Online Games are large virtual worlds co-inhabited by players over the Internet. As up to thousands of players can be simultaneously connected to the game, the server and network architectures are required to scale efficiently. The traditional client/server model results in a heavy financial burden for operation of the server. Various alternative architectures have been proposed as a replacement for the traditional model, but the adoption of these alternatives are slow as they present their own set of challenges.

The proposed hybrid system is based on many different architectures and peer-to-peer concepts that were reviewed in the literature. It aims to provide a compromise for existing, commercially successful MMOGs to introduce peer-to-peer components into their systems with no requirement of modification to their server or client software.

With the system's design presented, the middleware software is implemented and deployed in a real, controlled environment alongside an Ultima Online game server and its clients. The movement game mechanic was distributed amongst the peers while the others remained the responsibility of the server. A number of performance experiments are performed to measure the effects of the modified system over the original client/server system on bandwidth, latency, and hardware impact. The results revealed an increase in the server bandwidth usage by 35%, slave bandwidth usage by 17% and supernode bandwidth usage by 3111%. The latencies of distributed server mechanics were reduced by up to 94%, while the non-distributed latencies were increased by up to 6000%. These results suggested that a system with absolutely no modification to the server is unlikely to provide the desired benefits.

However, with 2 minor modifications to the server, the middleware is able to reduce both server load and player latencies. The server bandwidth can be reduced by 39%, while the supernode's bandwidth is increased only by 1296%. The distributed latencies maintain their reduction while non-distributed latencies remain unchanged from the C/S system.

# Uittreksel

Massiewe Multispeler Aanlyn Speletjies (MMAS) is groot virtuele wêreldes op die Internet wat bewoon word deur spelers. Aangesien duisende spelers gelyktydig kan inskakel op die speletjie word daar verwag van die bediener en netwerk argitektuur om effektief te skaleer om die groot hoeveelhede spelers te kan hanteer. Die tradisionele kliënt/bediener model lei tot 'n groot finansiële las vir die operateur van die bediener. Verskeie alternatiewe argitekture is al voorgestel om die tradisionele model te vervang, maar die aanvaarding en in gebruik neem van hierdie alternatiewe (soos eweknie-netwerke) is 'n stadige proses met sy eie stel uitdagings.

Die voorgestelde hibriede stelsel is gebaseer op baie verskillende argitektuur- en eweknie konsepte wat in die literatuur oorweeg is. Die doel is om 'n kompromie vir bestaande komersiële suksesvolle MMASs te verskaf om eweknie komponente te implementeer sonder om die die bediener- of kliënt sagteware aan te pas.

Met hierdie stelsel se ontwerp word die middelware sagteware geïmplementeer en gebruik in 'n regte, dog gekontroleerde omgewing, tesame met 'n Ultima Online bediener en sy kliënte. Die beweging speletjie meganisme word versprei onder die eweknie netwerk en die ander meganismes bly die verantwoordelikheid van die bediener. 'n Aantal eksperimente is ingespan om die effek van die hibriede stelsel te meet op die oorspronklike kliënt/bediener stelsel, in terme van bandwydte, vertraging en impak op hardeware. Die resultate toon 'n toename van 35% in bediener-, 17% in slaaf-, en 3111% in supernodus bandwydte gebruik. Die vertraging van verspreide bediener meganismes neem af met tot 94%, terwyl onverspreide vertraging toeneem met tot 6000%. Hierdie resultate wys dat 'n stelsel wat geen aanpassing maak aan die bediener sagteware onwaarskynlik die gewenste voordele sal lewer.

Deur egter 2 klein aanpassings toe te laat tot die bediener, is dit moontlik vir die hibriede stelsel om data las van die bediener en die speler se vertraging te verminder. Die bediener bandwydte kan met 39% verminder word, terwyl die supernodus bandwydte slegs met 1296% toeneem. Die verpreide vertraging handhaaf hul vermindering, terwyl die onverspreide vertraging onveranderd bly van die C/S stelsel.

# Acknowledgements

- My supervisor, Dr. Herman A. Engelbrecht, for his guidance, expertise and motivation.
- My family and friends for their support and understanding throughout the process.
- The MIH Media Lab for providing opportunities and financial support for my research.
- My lab colleagues for sharing their knowledge, and contributing towards the enjoyment of the project.

# Publications

The following publications arose from this research:

- Croucher, D.A. and Engelbrecht, H.A.: A peer-to-peer middleware plugin for an existing MMOG client-server architecture. In: *2012 IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE 2012) Proceedings*, pp. 140-141. IEEE, October 2012. ISBN 978-1-4673-1567-8.

# Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Uittreksel</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>Publications</b>	<b>v</b>
<b>Contents</b>	<b>vi</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Acronyms</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Components of an MMOG . . . . .	2
1.2 Architectures . . . . .	4
1.2.1 Client/Server . . . . .	4
1.2.2 Peer-to-Peer . . . . .	5
1.2.3 Hybrid . . . . .	6
1.3 Problem Statement . . . . .	7
1.4 Proposed Solution . . . . .	7
1.5 Objectives . . . . .	7
1.6 Contributions . . . . .	8
1.7 Overview . . . . .	8

<i>CONTENTS</i>	<b>vii</b>
<b>2 Background &amp; Related Work</b>	<b>10</b>
2.1 Introduction . . . . .	10
2.2 Massively Multiplayer Online Games . . . . .	11
2.2.1 History . . . . .	11
2.2.2 Architecture . . . . .	11
2.2.3 Latency . . . . .	13
2.2.4 Traffic . . . . .	13
2.2.5 Current Solutions . . . . .	14
2.3 Peer-to-Peer . . . . .	15
2.3.1 Overview . . . . .	15
2.3.2 Usage in MMOGs . . . . .	16
2.3.3 Traffic . . . . .	17
2.4 MMOG Game State Models . . . . .	18
2.4.1 Update-Based Model . . . . .	18
2.4.2 Event-Based Model . . . . .	19
2.5 Requirements for Peer-to-Peer MMOG systems . . . . .	20
2.6 Interest Management . . . . .	22
2.6.1 Overview . . . . .	22
2.6.2 Area of Interest . . . . .	23
2.6.3 Supernode . . . . .	23
2.6.4 IM Model: Spatial . . . . .	25
2.6.5 IM Model: Region-based . . . . .	25
2.6.6 IM Model: Hybrid . . . . .	26
2.7 Existing System . . . . .	27
2.7.1 Requirements . . . . .	27
2.7.2 Selecting a System . . . . .	28
2.7.3 Automation . . . . .	28
2.7.4 Architecture and Operation . . . . .	28
2.7.5 Positional System . . . . .	29
2.7.6 Interest Management . . . . .	30
2.7.7 Limitation of Modification . . . . .	30
2.7.8 Imposed Constraints . . . . .	30
2.8 Summary . . . . .	31
<b>3 Design &amp; Implementation</b>	<b>33</b>
3.1 Introduction . . . . .	33
3.2 Hybrid System Overview . . . . .	33



<i>CONTENTS</i>	<b>viii</b>
3.3 Interest Management . . . . .	36
3.3.1 Introduction . . . . .	36
3.3.2 Aims . . . . .	36
3.3.3 Design . . . . .	36
3.4 Network Architecture . . . . .	39
3.4.1 Aims . . . . .	39
3.4.2 Design . . . . .	39
3.5 Implementation . . . . .	43
3.5.1 Overview . . . . .	43
3.5.2 Core . . . . .	43
3.5.3 Networking . . . . .	45
3.5.4 Interest Management . . . . .	54
3.5.5 Game Logic . . . . .	62
3.6 Summary . . . . .	65
<b>4 Experimental Investigation</b>	<b>67</b>
4.1 Introduction . . . . .	67
4.1.1 Performance Metrics . . . . .	67
4.1.2 Game Automation . . . . .	69
4.1.3 Testing Duration . . . . .	69
4.2 Bandwidth Evaluation . . . . .	70
4.2.1 Overview . . . . .	70
4.2.2 Test Platform . . . . .	70
4.2.3 Test 1.1: Baseline Bandwidth . . . . .	71
4.2.4 Test 1.2: Single Region Bandwidth . . . . .	73
4.2.5 Test 1.3: Inter-region Bandwidth . . . . .	79
4.2.6 Conclusion . . . . .	83
4.3 Latency Evaluation . . . . .	84
4.3.1 Overview . . . . .	84
4.3.2 Protocol Modification . . . . .	85
4.3.3 Test Platform . . . . .	85
4.3.4 Test 2.1: Latency Baseline . . . . .	86
4.3.5 Test 2.2: Latency with Supernode situated near server . . . . .	87
4.3.6 Test 2.3: Latency with Supernode situated far from server . . . . .	91
4.3.7 Conclusion . . . . .	94
4.4 Hardware Impact Evaluation . . . . .	95
4.4.1 Overview . . . . .	95

<i>CONTENTS</i>	<b>ix</b>
4.4.2 Test Platform . . . . .	95
4.4.3 Test 3.1: Hardware Impact of server software . . . . .	96
4.4.4 Test 3.2: Hardware Impact of middleware software . . . . .	97
4.4.5 Conclusion . . . . .	99
4.5 Summary . . . . .	101
<b>5 Simulation</b>	<b>102</b>
5.1 Introduction . . . . .	102
5.2 Requirements . . . . .	102
5.3 Platform . . . . .	103
5.4 Design . . . . .	103
5.4.1 Node . . . . .	103
5.4.2 Traffic . . . . .	104
5.4.3 Configurations . . . . .	105
5.5 Simulation: Bandwidth . . . . .	107
5.5.1 Motivation . . . . .	107
5.5.2 Configuration . . . . .	107
5.5.3 Results & Discussion . . . . .	107
5.6 Simulation: Latency . . . . .	111
5.6.1 Motivation . . . . .	111
5.6.2 Configuration . . . . .	111
5.6.3 Results & Discussion . . . . .	111
5.7 Conclusion . . . . .	112
<b>6 Conclusion &amp; Recommendations</b>	<b>115</b>
6.1 Introduction . . . . .	115
6.2 System Performance . . . . .	115
6.3 Feasibility Outcomes . . . . .	116
6.4 Related Work . . . . .	118
6.5 Future Work . . . . .	118
<b>Bibliography</b>	<b>120</b>
<b>A Hexagonal Co-ordinate System</b>	<b>127</b>
<b>B Test 1.1: C/S Baseline Bandwidth Results</b>	<b>130</b>
<b>C Test 1.2: Hybrid Single Region Bandwidth Results</b>	<b>131</b>

<i>CONTENTS</i>	x
<b>D Measurement Computer Specifications</b>	<b>132</b>
<b>E Hardware Usage Results</b>	<b>133</b>
<b>F Simulation: Bandwidth Results</b>	<b>136</b>

# List of Figures

2.1	Standard MMOG Network Architectures . . . . .	12
2.2	Update-Based Model showing the process of a state update. . . . .	19
2.3	Area of Interest surrounding a player . . . . .	23
2.4	Spatial Interest Management Model . . . . .	25
2.5	Example Voronoi cells where red lines are cell borders. A player is connected to two neighbours of cells intersecting his AOI. . . . .	26
2.6	Region Shapes: Red represents the player AOI while grey is unnecessary additional region information. . . . .	27
3.1	Example Hybrid Topology . . . . .	35
3.2	Example MOPAR connectivity for a player in a region . . . . .	37
3.3	Modified Hybrid Network Architecture . . . . .	40
3.4	System Modular Overview . . . . .	44
3.5	Basic networking module objects . . . . .	48
3.6	Networking node pairs where the centre middleware is the supernode .	48
3.7	Queue Processing Flow Diagram . . . . .	49
3.8	Supernode Handover Timeline . . . . .	52
3.9	Process on the middleware's receipt of a new connection. . . . .	54
3.10	Hexagonal Mapping . . . . .	58
3.11	Interest Units Hierarchy . . . . .	59
3.12	Neighbour Updating Packet Configuration . . . . .	61
4.1	C/S baseline results . . . . .	72
4.2	Bandwidth comparison between clients and slaves . . . . .	74
4.3	Bandwidth comparison between servers in the C/S and hybrid systems	76
4.4	Comparison in the number of movement events in the C/S system and periodic updates in the hybrid system received by the server respectively.	76
4.5	Supernode Bandwidth Distribution . . . . .	77

*LIST OF FIGURES***xii**

4.6	Potential bandwidth totals of hybrid nodes with client/server nodes for reference. . . . .	79
4.7	Bandwidth impact on supernode for hosting a slave at different durations	82
4.8	Comparison of Round Trip Times for clients sending non-distributed events and receiving their associated updates between the C/S system and hybrid system with a supernode located in the US West region. . . . .	89
4.9	Comparison of distributed latencies for peers communicating within the same region between the C/S system and hybrid system with a supernode located in the US West region. . . . .	89
4.10	Comparison of distributed mechanic latencies between peers where green connections indicate hybrid latencies, and blue indicate C/S latencies.	90
4.11	Comparison of Round Trip Times for clients sending non-distributed events and receiving their associated updates between the C/S system and hybrid system configurations. . . . .	92
4.12	Comparison of non-distributed mechanic latencies of updates between regions in the hybrid and C/S systems . . . . .	93
4.13	Comparison of distributed latencies for peers communicating within the same region between the C/S system and hybrid system configurations	94
4.14	Hardware Usage of server software between systems. . . . .	98
4.15	Hardware Usage of the middleware software. . . . .	100
5.1	Simulation Node Model . . . . .	104
5.2	Simulated Node AI State Machine . . . . .	106
5.3	Simulation Node Model . . . . .	108
5.4	Bandwidth comparison between simulated clients and slaves . . . . .	109
5.5	Bandwidth comparison between servers in the C/S and hybrid systems	109
5.6	Simulated Supernode Bandwidth Distribution . . . . .	110
5.7	Simulated Latency Results . . . . .	113
A.1	Hexagon . . . . .	128
A.2	Hexagonal Grid Co-ordinate Lookup . . . . .	129

# List of Tables

3.1	Directory Communications . . . . .	58
3.2	Logic Interface Functions . . . . .	64
4.1	Automated Player Behaviour . . . . .	69
4.2	Number of movement events/updates received by the server in the C/S and hybrid systems respectively. . . . .	75
4.3	Bandwidth values of supernode in the single region hybrid system and change from clients in the C/S system (B) . . . . .	77
4.4	Potential comparisons between nodes with server modifications (B) . . . . .	78
4.5	Bandwidth cost per slave for supernode (Bps) . . . . .	78
4.6	Cost of a login to the migrating node (B) . . . . .	81
4.7	Supernode inter-region bandwidth showing migration effects according to duration as a supernode, number of slaves over the period, and the amount of re-login traffic handled. . . . .	81
4.8	Supernode inter-region bandwidth showing overhead effects according to duration as a supernode, number of slaves over the period, the amount of neighbour update traffic experienced inbound and outbound, and the amount of AOI instruction sent. . . . .	83
4.9	Supernode inter-region bandwidth makeup of total traffic throughout duration . . . . .	83
4.10	Round Trip Time to Server per Region (ms) . . . . .	86
4.11	Average RTT of speech (non-distributed) with the supernode in USWest Region (ms) . . . . .	88
4.12	Average RTT of movement (distributed) with the supernode in USWest region (ms) . . . . .	88
4.13	Average RTT of speech (non-distributed) with the supernode in Ireland region (ms) . . . . .	91

*LIST OF TABLES***xiv**

4.14	Average RTT of movement (distributed) with the supernode in Ireland region (ms) . . . . .	93
B.1	Bandwidth values of clients in the C/S system (B) . . . . .	130
B.2	Bandwidth totals of server in the C/S system (B) . . . . .	130
C.1	Bandwidth values of slaves in the single region hybrid system and change from clients in the C/S system (B) . . . . .	131
C.2	Bandwidth values of server in the single region hybrid system and change from server in the C/S system (B) . . . . .	131
D.1	Measurement Computer Specifications . . . . .	132
E.1	Server without middleware . . . . .	133
E.2	Server with middleware . . . . .	134
E.3	Middleware software . . . . .	135
F.1	Bandwidth of clients in simulated C/S system (B) . . . . .	136
F.2	Bandwidth of peers in simulated single region hybrid system compared to to C/S system(B) . . . . .	136
F.3	Bandwidth of server in hybrid and C/S systems (B) . . . . .	137

# List of Acronyms

AOE	Area of Effect
AOI	Area of Interest
AWS	Amazon Web Services
C/S	Client/Server
C/MS	Client/Multi-Server
DHT	Distributed Hash Table
IM	Interest Management
LAN	Local Area Network
MMOG	Massively Multiplayer Online Game
MOPAR	Mobile Peer-to-Peer Overlay Architecture
MUD	Multi-User Dungeon
NPC	Non-Player Character
P2P	Peer-to-Peer
RTT	Round Trip Time
Rx	Received
Std	Standard Deviation
Tx	Transmitted
UO	Ultima Online
Upd	Updates



# Chapter 1

## Introduction

Massively Multiplayer Online Games (MMOGs) are computer games that allow large numbers of players to participate simultaneously over the Internet. Typically, an MMOG is set in a virtual world that is co-inhabited by all players of the game. The virtual world, also called a virtual environment, is most often persistent, where the actions of players in the game have a lasting effect on the world beyond a single session of gameplay. Players of the game are able to interact with the world and other players on various scales, from small groups of players performing tasks together, to large armies waging war on one another. While there are many characteristics between games that are common, such as movement and speech, the setting of the game often defines the broader goals of the players, such as: raiding a medieval dungeon for treasure or boarding an enemy spaceship to hijack its cargo.

A player is able to join the game by executing the game client software on their own computer, creating an account with the game host, and logging onto the host's server. Many games require that the player pays a once-off fee for the game account, a monthly subscription fee to continue playing, or both. Alternatively, a game account is free but restricted to certain in-game activities, and requires pay-as-you-go fees to access additional game content. A player is then granted access to the game by their account, and may participate in the virtual environment as long as they don't violate the game's terms of use. This allows the game host to control access to its system and generate long-term revenue as it maintains a large player base over time. For this reason, it is important that players have a good experience in the game so they remain part of the player base.

The business model of MMOGs turns players into revenue, therefore being able to scale the game to cater for more players generates higher revenue. It is important for the underlying network architecture and game protocol to be well designed and

capable of supporting large numbers of players.

## 1.1 Components of an MMOG

Regardless of the underlying architecture or implementation, all MMOGs have a common set of components. These components together make up the full functionality of the system and are considered the design principles of an MMOG. These components, some of which are described in a 2013 survey by Yahvini and Kemme, are summarised below [1].

**Player Interface** As the MMOG is played by people, it is necessary that the players have a way of interacting with the game. A software game client is required to act as an interface between the player and the game system. The client typically presents the game by rendering 2D or 3D graphics and using sound. Through the client, a player is able to input his desired actions, known as *events*, and perceive the changes, known as *updates*, that result from his own and other player's actions.

**Communication** The MMOG includes many players sharing the virtual environment, playing the game together and interacting. This requires that communication channels exist between players, hosts and administrators of the game. As the games are played over the Internet, this typically involves socket communication between nodes.

**Objects** An MMOG takes place in a virtual environment made up of a number of objects. Game objects can be categorised into four different categories: (1) *Immutable objects* represent the set of objects which do not change throughout the course of the game, such as buildings and terrain. They are often entirely static and included as part of the client software. (2) *Mutable objects*, in contrast, are dynamic and modifiable. For example, a player might consume some food or equip a weapon. (3) *Player mobiles, player characters* or *avatars* are all words to describe the player's representation in the virtual world. A player controls their mobile through their input into the game client. (4) *Non-player characters* (NPCs) are characters which are controlled by artificial intelligence, acting without player input. All objects typically have additional parameters associated with them to describe their current state, such as the amount of health points.

When a player joins the environment, they receive a local copy of the virtual environment made up of objects. For every object, there exists a single *authoritative*

copy, and various *replicas*. When any player interacts with an object, the event is first transmitted to the owner of the authoritative copy who decides whether the event is valid. Once processed, an update is transmitted from the owner to all replica holders who apply it to their local copy. The game client uses its copy of the objects to populate the graphical interface.

**Interactions** Typically players have three categories of interaction in an MMOG: (1) *Player updates* are interactions in the world which only affect the player, such as movement causing a change in position. (2) *Player-object* interactions refer to all those between the player and mutable objects, such as picking up some food. (3) *Player-player* interactions occur between players, such as one player attacking another and reducing their current health.

The manner in which these interactions may occur is defined by the rule set of the game engine. The game engine contains game logic which processes an interaction to produce an update.

**Consistency** It is necessary that all players perceive the world in the same way. At any point in time, the virtual environment has a specific *game state* based on the current state of authoritative objects. Each player has a local copy of the game state which is displayed by their game client. If there are discrepancies between the authoritative state and the players' local game states, it might result in strange behaviour, such as a player attempting to use an item that is no longer there or out of their reach. Therefore, consistency of game state is important to ensure that players view and interact with the world accurately.

**Persistence** The virtual environment continues to exist beyond when a player leaves the game. If any player leaves the game and returns a few days later, their player mobile should still be available and in the same state that it was when they left. Similarly, any changes that the player made to the virtual environment should still be in effect when they return. This requires that the game state is recorded somewhere and stored persistently.

**Interest Management** Players typically have limited sensing capabilities and are only able to perceive what is near to their player mobile in the virtual world. A player's *interest* is limited to only what is relevant to them at that point in time. By limiting a player's local copy of the virtual environment to what they are interested in, it reduces the transmission of information pertaining to irrelevant

objects. Therefore, an MMOG must be able to distinguish what is relevant to a player's interests and manage the objects and state accordingly.

**Security** Access to the game should be restricted to authorised players. The MMOG should be secure against various negative and malicious influences such as cheating, virtual crimes, hacking attempts, malicious attacks on the network, or simply players who violate the terms of use.

## 1.2 Architectures

### 1.2.1 Client/Server

Traditionally MMOGs have been developed to use a Client/Server (C/S) network architecture in which a server is the centralized authority hosting the virtual environment. Players are required to login to the server using their accounts and are provided access to the world. The server, being the authority, is responsible for all aspects of the game: access control, storage of persistent game data, processing interactions, acting for all non-player characters, informing players of updates to objects, and detection of cheating or rule violations. In this architecture every client is connected to the server, thus all game traffic is sent to the server that responds in kind.

Not all games use only a single server, but rather implement a Client/Multi-Server (C/MS) architecture. Multiple interconnected servers share the load of hosting the game between them, allowing the game to be scaled further by introducing additional servers which increase the hosting capacity without having to upgrade any existing hardware.

Centralised architectures offer a number of advantages for the game host. Centralised control allows the developer to manage all access to the server, inherently increasing the security of the system. As the server hosts all authoritative data, it is able to verify all player actions in a consistent fashion and detect any irregularities. If any updates or modifications to the game logic need to be made, it need only be applied on the server to affect the whole system. All persistent data is stored by the server, which makes the task of backing up game data trivial. Finally, the game can be scaled to handle more players simply by upgrading the server hardware and Internet bandwidth.

However, centralised architectures are not without disadvantages. The expense for hosting the server(s) is considerable as they have to deal with the full burden associated with each player. Hardware for servers needs to be acquired, maintained and even upgraded if necessary. Networking-related costs for bandwidth and traffic usage over the Internet also contribute towards hosting expenses. The operating expenses can consume considerable amounts of the revenue stream.

The generation of revenue relies on maintaining a strong player base, which in turn requires the players to receive a quality entertainment experience. Players who are not geographically situated near to the server might experience degraded gameplay due to latency issues. For example, a delay in a combat situation can impact the game experience negatively by breaking immersion and causing frustration. Similarly, if the server becomes overloaded, all players will suffer from latency problems. Furthermore, servers in centralised architectures are singular points of failure, and if one were to fail it can catastrophically affect the players of the game by causing downtime and loss of game data.

### 1.2.2 Peer-to-Peer

Due to the shortcomings in centralized server architectures, alternative architectures have been proposed in recent years. A number of these methods aim at altering the network structure in a way that may benefit both the players and hosts of games. One of the primary alternatives is considered to be a Peer-to-Peer (P2P) network architecture. The aim of a P2P architecture for an MMOG is to fully distribute the bandwidth and processor requirements of a game amongst the players rather than being solely hosted by the server. Therefore, no central authority exists in the system, and the game is completely hosted by the players. The motivation for this change in architecture stems from the increase in end-user's computing power over the past decade coupled with the increase in Internet bandwidth availability.

Every peer in the network acts as both a client and a server with the authoritative objects distributed amongst them. Each peer hosting an authoritative object is capable of processing the interaction events and producing updates which can be disseminated to all replica holders. Distributed mechanisms exist to maintain persistence and consistency of the objects. The network is structured so that communication is done between peers rather than through a central location, and various methods of distributed interest management exist to aid scalability. The requirements of P2P architectures for MMOGs, as described by Schiele et al. in [2], are more extensive than the basic components of an MMOG.

Distributing the load amongst the peers eliminates the need for a server. This results in the developer no longer having the start-up and ongoing expenses that were described in the C/S architecture. This offers the potential for new developers to enter into the MMOG market without such a large capital investment. Lower expenses might also translate into lower player fees, encouraging more players to join the game.

As there is no central server, players are able to communicate more directly with peers. This contributes to a reduction in latency, eliminating part of the delay in interactions and producing an improved gameplay experience.

Finally, P2P systems have no single point of failure which makes them more robust against system downtime and data loss. Many peers would have to fail simultaneously for the P2P MMOG to be impacted.

P2P MMOGs are not without disadvantages. As the requirements for hosting an MMOG on a P2P architecture are extensive, each requirement presents complex problems to the designer and no system has yet been able to meet all of them.

Beyond the complexity in designing a P2P MMOG, it is necessary for the host of the game to relinquish its absolute control and distribute it amongst the players of the game. It is unlikely that a company who has invested heavily into developing an MMOG will willingly transfer control of the game to the players.

Therefore, P2P architectures are slow to be adopted as viable architectures for commercial MMOGs [1].

### 1.2.3 Hybrid

As both architectures offer their own set of advantages and disadvantages, many hybrid architectures have been proposed to incorporate aspects of both the C/S and P2P architectures into one. By allowing the server to maintain its control over the game, but distributing certain aspects amongst the players, a middle ground can be established between the two architectures. An example of such a distribution might involve the virtual world and authoritative objects being hosted by the peers, allowing them to process interactions, while the server maintains access control to the game, monitors security issues, and manages persistence.

## 1.3 Problem Statement

Well-established MMOGs require servers that are capable of hosting thousands of concurrent players. Each player places a further bandwidth and hardware requirement on the server that ultimately incurs a cost to the game host. In total, the bandwidth and processing for hosting all of the players generates significant ongoing expenses for the host. Furthermore, the host must continuously provide the players with a quality gaming experience to maintain revenue generation.

With the idea that using a P2P architecture will reduce the server bandwidth and improve the gameplay experience, the developer could consider redesigning their original C/S architecture to support P2P components. However, this is not always a viable option as doing so poses many risks, and has serious financial implications on the developer who has already invested heavily into maturing the MMOG into a stable and commercially viable state.

## 1.4 Proposed Solution

The problem statement leads to the proposition of this system: a middleware plugin for established C/S MMOGs. The transparent middleware plugin aims to modify the existing architecture by introducing peer-to-peer elements without any requirement of modification to the original client or server software. It is able to autonomously configure the peer-to-peer network and distribute the processing of any selected game mechanics, such as movement or speech, amongst the players. This should result in a reduced server load since the load is distributed amongst numerous peers. In addition, as portions of the game processing will be performed by the players, a reduction in latency should occur for players who are situated geographically close to each other.

If this can be achieved, it should reduce the expenses of operating the server, as well as improving the gameplay experience for players without impacting them beyond their capabilities.

## 1.5 Objectives

The objectives of this study are as follows:

1. Design a novel middleware plugin that will transform an existing, unmodified system into a hybrid P2P architecture and distribute selected game mechanics

amongst the peers.

2. Implement the middleware plugin on an existing system as a proof-of-concept.
3. Evaluate the proposed hybrid architecture and measure its effectiveness compared to the traditional C/S system.
4. Simulate the proposed architecture to confirm the trends observed in the practical tests.

## 1.6 Contributions

The contributions of this study are as follows:

1. The design of a peer-to-peer middleware plugin for a real game.
2. Real, practical results showing the feasibility of this type of system.
3. A base platform for further research into P2P MMOG concepts to be conducted upon.

## 1.7 Overview

Chapter 2 introduces the background concepts pertaining to various aspects of the middleware plugin and its hybrid system. It starts with a brief history of MMOGs, and examines the traditional architectures used by them in some detail. Some of the drawbacks of the traditional systems are highlighted, and what current solutions are in place to alleviate them. This opens the door to the introduction of peer-to-peer architectures into MMOG systems. After reviewing the basic principles of P2P networks and their applications in MMOGs, some of the design concepts are presented in finer detail. An examination of game state models, P2P MMOG requirements and interest management each contribute towards the design choices of the system. With the key concepts covered, some details of Ultima Online, the chosen existing MMOG system, are explained, with particular attention to how they will affect the design of the system.

This leads into chapter 3 which details the design and implementation of the middleware plugin. Discussing from various levels, the chapter systematically goes into finer detail beginning with a top-level description of the designed hybrid architecture, the network architecture to support it, and the interest management scheme that was implemented for the system. With the broader design covered, a



more detailed examination of the design of the middleware software is presented according to its different sub-modules. Each design component has its intended aims presented, and how they were accomplished.

With the system implemented, Chapter 4 begins by introducing the performance metrics by which the middleware will be measured: bandwidth, latency and hardware impact. Common elements of the tests are presented before each performance metric is measured. Three bandwidth tests define the baseline for comparison; the middleware single region bandwidth; and the middleware inter-region bandwidth. Three latency tests define the baseline for comparison; a hybrid system best case configuration; and a hybrid system worst case configuration. The hardware impact of the hybrid system is examined for its effects on the server, as well as how it will scale for players. Each test is discussed with regards to the test platform, motivation, configuration and results.

As a means to verify what was observed in the prior practical tests, Chapter 5 presents simulation models of the C/S and hybrid systems. The chapter discusses the requirements, platform and design of the simulation. Results of the simulated models are presented focusing on the effects of the hybrid system on bandwidth and latency.

Chapter 6 concludes the thesis by presenting the final outcomes of the study and relating it to other hybrid systems. Finally, it presents recommendations for future work.

## Chapter 2

# Background & Related Work

### 2.1 Introduction

This section provides background to many key concepts associated with the middleware plugin and the hybrid system. It begins with a history of Massively Multiplayer Online Games and the architectures they are built upon, the challenges of those architectures and what some of the current solutions to the challenges are. Following this, peer-to-peer networking architectures are introduced, along with their usage in MMOGs and their current state of bandwidth usage.

With the base systems defined, MMOGs and their P2P architectures are examined in finer detail. The game state models of MMOGs are explained, providing context to how MMOGs maintain their virtual worlds. The requirements of peer-to-peer MMOGs are listed and summarised. Interest management, a common mechanism for P2P MMOGs, is investigated with a focus on *area of interest* and *supernode* concepts and the different interest management models.

Finally, background is given on Ultima Online: the existing system chosen for developing the middleware. The requirements of the existing system are listed, the process of selecting a specific game is explained, and then specific aspects of Ultima Online are discussed, such as the positional system, the built-in interest management, how the game can be automated, and finally the limitations introduced by this experiment and the constraints which result from them.

## 2.2 Massively Multiplayer Online Games

### 2.2.1 History

Before the arrival of large-scale multiplayer games, there were Multi-User Dungeons (MUDs). Developed first in 1978 by students of Essex University, it allowed multiple players to connect over a network and explore a dungeon environment through a textual interface. The content and inspiration of the game derived from the traditional tabletop roleplaying game, Dungeons and Dragons [3]. From that point, multiplayer games continued to develop towards what some consider the first MMOG developed in 1991, Neverwinter Nights. Supporting up to 200 concurrent players, this game boasted simple graphics while maintaining the textual instructions that were originally featured in MUDs [4]. It was the release of Meridian 59 in 1996 that coined the term *massively multiplayer* by securing the support of thousands of players.

Finally, Ultima Online (UO) was released in 1997 and is considered as the first commercially successful MMOG. Using the *Ultima* title from an already successful single player game series, UO quickly gained 100 000 subscribers and still maintains an active player base today [5]. Many more MMOGs have since been released, World of Warcraft being the most popular amongst them having achieved a peak of 12 million subscribers in 2010 [6]. A number of the recent MMOGs have over 600 000 concurrent players connected to their servers simultaneously [7].

### 2.2.2 Architecture

Currently, commercial MMOGs all fall into one of two categories of network architectures, either a Client/Server (C/S) architecture or a Client/Multi-Server (C/MS) architecture. A game host, who produces the game, usually owns the server(s) where the MMOG itself is hosted and all data pertaining to the virtual environment is stored. A client is a software application which allows a player to access the virtual world as a player mobile, a virtual representation of the player, through a graphical interface.

A C/S based system has a single server which performs all the tasks required of the game host. This includes: authorising access to the game through some form of login system; storing all of the virtual environment's data (such as objects or non-player character locations) to ensure that players are able to experience a persistent world; computing outcomes of player's actions based on the set of game rules defined for the MMOG (such as player mobile movement, spell casting, etc.); detecting irregular activity and cheating; hosting of chat servers to allow player

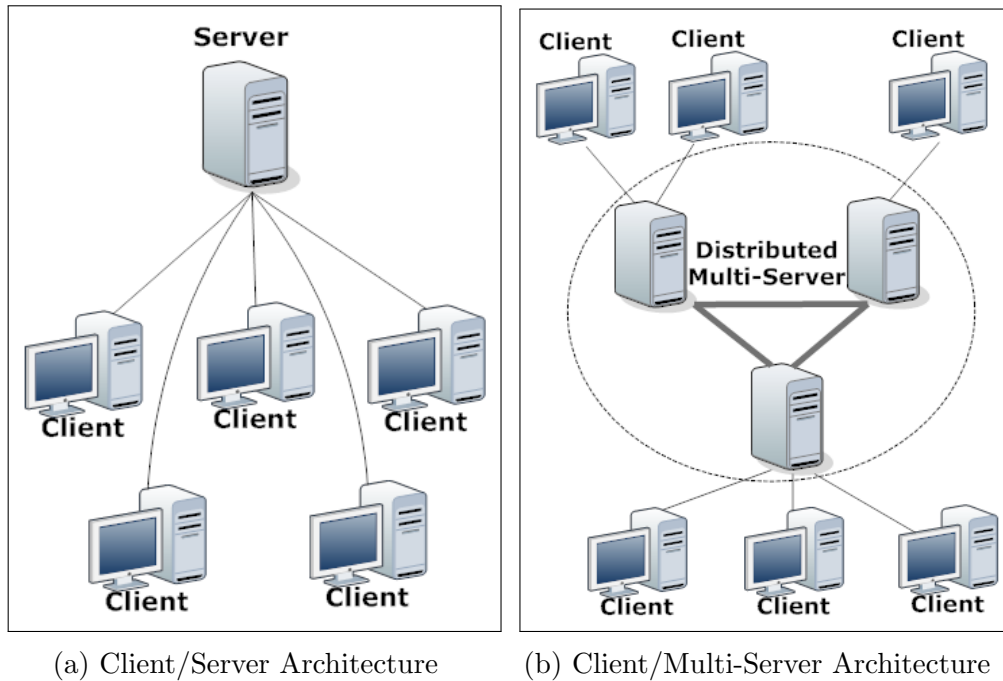


Figure 2.1: Standard MMOG Network Architectures

communication; and various other tasks. Furthermore, the server generally hosts patching updates for outdated clients connecting to the server [8]. A typical setup of a C/S system is shown in Fig. 2.1a.

A C/MS based system performs similar tasks to the single server, but in a distributed manner. There are fast, high bandwidth connections established between each server and the tasks can be divided up in a number of ways. A typical setup of a C/MS system is shown in Fig. 2.1b.

In all of these mentioned setups, each player has their own copy of the client software that establishes a connection with the game host's server and authenticates itself. Once authenticated, this becomes the only channel of communication that the client has with the game. Therefore, the client software must send all player-initiated events to the server and receive the associated updates about the changes in a player's mobile and surrounding world. The updates received are generally limited to what is within the player's *area of interest*. The area of interest can be described as a bounded area within the MMOG's virtual environment that the player is interested in. Using the area of interest principle limits the traffic that would otherwise be used to update each player about *everything* in the world.

When examining the costs for developing a system to support the massive num-

ber of players, the costs incurred for both software and hardware can run well into the millions of dollars [9][10]. For this reason it is understandable that game production companies are reluctant to modify their well-established setups and why start-up companies are reluctant to enter into the MMOG field at all.

### 2.2.3 Latency

As MMOGs are played in real-time, latency has a significant effect on the players experience of the game. A 2002 study into the effects of latency on real-time multi-player games reveals the importance of low latency in competitive situations [11]. If players are competing against one another, differences in network latencies can affect the performance of players, providing unfair advantages in terms of the gameplay. A good example is that of player-versus-player combat in an MMOG: if two players attack each other at the same real time, but have a difference in latencies, the player with the lower latency will appear to the server to have struck first. Therefore, it is important that latencies are minimized as far as possible in instances where players are competing.

There also exists cases in MMOGs where latency is not as critical. In a study on the MMOG *Everquest2*, it was shown that for simple movement and combat against non-player characters, higher latencies were found to be tolerable [12]. The reason behind this was found to be the queue-based combat approach that *Everquest2* uses. However, in MMOGs such as *Guild Wars 2*, the positioning of players can significantly impact the outcome of combat against non-player characters as dodging mechanics are included. In this case, latencies once again become important to decide whether the player completes the dodge in the allowed time-window.

Therefore, it can be considered that low latencies are necessary for providing players with a positive gameplay experience through fairness, equality, and by eliminating the frustration that is derived from poor latency.

### 2.2.4 Traffic

There have been a number of studies into the bandwidth and traffic usage of clients and servers in established MMOGs. One of these was performed in 2005 on *Shen-Zhou Online*, a commercial Taiwanese MMOG hosting thousands of players. The study implemented traffic monitoring software on the game's server and ultimately concluded that, with the assumption that each client requires a downstream bandwidth of 10kbps to play the game, for the server to host 370 000 concurrent con-

nections it would require a 3.7Gbps connection [13]. A 2008 study into *Second Life* client bandwidth usage predicted that a minimum downstream of 10kbps would be required when a player's mobile is not moving and its area of interest is sparsely populated by players and objects. However, when the player's mobile is moving through a densely populated area of interest, the bandwidth usage can spike as high as 877kbps [14]. A simulation of *World of Warcraft* concluded that a client uses an average downstream of 51kbps over all areas of the game [15].

Given these bandwidth usage statistics of MMOGs, while no exact figures are publically available, it is clear that the cost for hosting such a system continuously will be very high to cater for both the bandwidth and traffic requirements [9]. In addition to this, the server must have enough processing capabilities that it is able to process events received from all players connected to the game. The costs of such high-end hardware and the maintenance thereof raises the production cost of the MMOG even further. It has been estimated that a minimum of 40 percent of revenue generated by MMOGs is consumed by the costs to run and maintain the server. In some cases the estimation can reach as much as 80 percent [16][17].

### 2.2.5 Current Solutions

While the adoption of peer-to-peer alternatives are very slow [1], C/S architectures are achieving scalability through other means. By making use of cloud computing, game hosts are able to launch more instances of servers to manage fluctuating player numbers. By using this technology, a host is able to temporarily scale their processing capabilities, without the need to secure additional physical hardware. Architectures are available to seamlessly manage multiple servers being added and removed from the system.

One such commercial solution is *BigWorld*, which offers dynamic, real-time, load balancing. Players are migrated between servers transparently, without interrupting gameplay, allowing for additional servers to be added or removed as the load changes. The system is also fault-tolerant, recovering from any server failures with no lost data [18].

*Kiwano*, developed by Daiconu and Keller, offers a game-agnostic system for distributing movement and positional game mechanics [19]. The system divides the virtual world into dynamic zones, and has each zone managed by a different server that hosts a spatial index of all objects within the zone. The servers are interconnected and exchange information pertaining to their borders with servers of neighbouring zones. Players connect to a proxy which connects, on behalf of

the player, to the correct zone server and handles communications with it. When a player changes zones, the proxy automatically switches to another server, and the change remains transparent to the client. When the player moves, the zone server is informed and it in turn notifies all other nearby players of the changes. Additional servers can be added and removed from the system to handle more of the dynamically allocated zones. The proxy in the Kiwano system shares a lot of functionality with the proposed *middleware* plugin.

While solutions like these are very elegant for solving the scalability problems inherent to C/S architectures, they are still a financial burden to the game hosts. The volume of data requiring processing and game traffic that the players produce must be serviced and is still the complete responsibility of the game host.

## 2.3 Peer-to-Peer

### 2.3.1 Overview

A generic decentralized P2P network architecture is very different from a C/S network architecture in that there is no central server that every node connects and communicates through. Instead, all nodes in the network are potentially directly connected. Because there is no need for a server, developing applications to use a P2P network architecture could dramatically lower startup and maintenance costs. Many publications discuss how such a P2P network should be arranged and connected to provide a stable network upon which an application can be built [20]. The way in which the P2P network nodes are arranged is what is known as a *network overlay*. Overlays can be considered distributed lookup protocols for P2P networks, allowing nodes to locate where data can be stored in and accessed from the network. Some of the network overlays are Chord [21], CAN [22], Tapestry [23], and Pastry [24]. The self-organising overlays provide the functionality of a Distributed Hash Table (DHT) where keys and address values of different nodes contained within the network can be looked up to provide an address of where in the overlay the nodes are located. On top of these overlays, additional layers have been constructed to provide functionality to the network, such as multicast messaging through the use of Scribe [25] and persistent data storage through PAST [26].

Many P2P overlays make use of *direct communication*, where any nodes who need to communicate connect directly to each other [1]. Direct communication overlays typically offer the lowest latencies as data is transmitted from the origin directly to its destination with no additional hops. In addition, these overlays are

often the simplest to setup and maintain. However, they are often limited by the upload capacity of a node, as the originating node is responsible for sending the data to all interested nodes, which in some cases might be many. Therefore, interest management is necessary to efficiently manage direct communication.

An alternative to direct communication overlays are *multicast* overlays. Nodes are organised in a tree structure where every node is responsible for transmitting data to its children. This has a lower bandwidth requirement as nodes typically have few children, and thus only have a few nodes to transmit data to. However, this structure suffers from higher latencies as data requires numerous hops before it reaches its destination. In addition, many nodes see an increase in bandwidth even if they only act as forwarders and receive data that has no relevance to them. Finally, multicast overlays are more complex to configure and maintain.

### 2.3.2 Usage in MMOGs

In a 2004 paper by Knutsson et al., the approach of using a P2P network overlay to implement an MMOG was proposed [27]. In the experiment, a P2P MUD was implemented using the Pastry network overlay and Scribe enhancements. With this setup, up to 4000 players were connected simultaneously to the virtual environment. The results of this study opened the field for research into P2P MMOGs. Since then, many further architectures capable of supporting P2P MMOGs have been proposed. A paper by Hampel et al. builds on Knutsson's architecture by introducing PAST into the system to manage game objects and resources through a storage system [28]. A fully distributed architecture focusing on scalability of the system is presented by Hu and Laio describing how to arrange connectivity between nodes, introducing the concept of distributed interest management [29]. Another architecture was introduced in 2007 by Chan et al. called Hydra. Their system addresses the need for reliability where network churn, the continuous entry and exit of nodes in the system, occurs in a P2P environment, and managing nodes that leave holes in the P2P overlay [30]. Each of these architectures are built focusing on different requirements of P2P networks for MMOGs.

While the previous proposals were all based on pure P2P architectures where no central authority is present, a number of others have taken a hybrid approach where some form of centralized authority is maintained to perform chosen tasks while the P2P network makes up the bulk of the architecture. Chen and Muntz suggest that a solution to some of the difficulties introduced by peer-to-peer systems can be solved by leaving the server in the system to handle issues like bootstrapping, persistence



and security. The remaining P2P segment of the network can then manage server computation and traffic to increase scalability of the system [31]. *VoroGame*, an architecture by Buyukkaya et al., integrates the propositions of [29] into their system to arrange connectivity between correct nodes [32]. Carter et al. developed a full hybrid system, *Net Homura 2.0*, where they engineer their own server to act as a ‘bouncer’ and ‘usher’ while allowing their P2P clients to manage the game state [33].

In a 2009 paper by Varvello et al., they develop a practical P2P system for *Second Life*, a commercial MMOG, and evaluate its feasibility. They distribute the management of the virtual world amongst their peers over a structured P2P network built atop of Kad, a P2P overlay that was developed for file-sharing. Using captured player traces, they emulate the gameplay of *Second Life* on their system and show that using the standard P2P structure, they were able to provide a consistent, persistent and scalable virtual world. While being able to maintain some features, players migrating between areas, or joining the game, caused a large spike in latency. In cases where an area was densely populated, it prevented a moving player from receiving a consistent view of their current area. However, these findings do reiterate it is possible for peer-to-peer MMOGs to work effectively and host the entire virtual world on a distributed system [34].

It would appear a *middleware* application, that can integrate into an already established C/S MMOG to transform it into a hybrid P2P system, has not yet been implemented.

The motivation to utilizing a P2P network is to reduce the need for a server by having each player in the game contribute its own available processing power and bandwidth to host itself. Thus some or all of the server tasks previously described in section 2.2.2 are distributed between the peers. Implementing P2P MMOGs present the developer with both benefits and challenges over a traditional C/S architecture. If executed correctly, a P2P MMOG can be more robust, more scalable, have a lower cost, reduce latency, handle the peak transient load more effectively and eliminate the need for a server altogether [35].

### 2.3.3 Traffic

A study released in 2010 discusses the near-term feasibility of P2P MMOGs [15]. In this study the traffic and bandwidth usage of a popular MMOG, *World of Warcraft*, is simulated and analysed as both a C/S and P2P architecture. The final outcomes were stated that for the near future, a full P2P architecture would be unsupportable given the current capabilities of the average player’s available bandwidth. This

points towards the use of a hybrid centralized system being a more viable possibility.

## 2.4 MMOG Game State Models

Describing the MMOG game state models helps in understanding how an MMOG maintains the virtual world. Gilmore's paper on state persistency addresses the two traditional models: the event-based model and the update-based model [35]. Reiterating the following terminology aids in understanding the models he describes which are summarised below.

**Events** are generated by the client software whenever a player interacts with objects in the virtual world, e.g. Eating an apple.

**Game Logic** is part of the game engine and forms the rule-set of the game. It describes the response to different activities in the game, e.g. Eating an apple reduces hunger.

**Updates** are the result of applying game logic to events. They describe how objects in the the virtual world should change, e.g. Player is no longer hungry.

**Game State** refers to the current state of all objects in the virtual world, including every detail about them.

### 2.4.1 Update-Based Model

The update-based model is the traditional C/S model of MMOG operation. A server hosts the authoritative game state of the entire virtual world. All the game logic is stored on the server, and clients are required to send events to it for processing. Each client also has its own local game state that it believes to match the servers authoritative state.

The process of a game state change is shown in Fig. 2.2. Initially the game is in state  $S_0$ , and a client produces an event. This causes his own state to change to a predicted next state  $S'_0$ , and for his event to be sent to the server. The server applies the game logic to the event upon receiving it to produce an update. It applies the update to its own global game state to update the authoritative state to  $S_1$ . The server then broadcasts the update to all clients, which apply the update to their own local states, updating themselves to  $S_1$ .

The update-based model offers strong security as clients are unable to alter the game state in any way other than through events. They rely on updates from the server to have their game state changed. If there are ever any conflicts in state, the server's state is always deemed to be authoritative and overwrites all others.

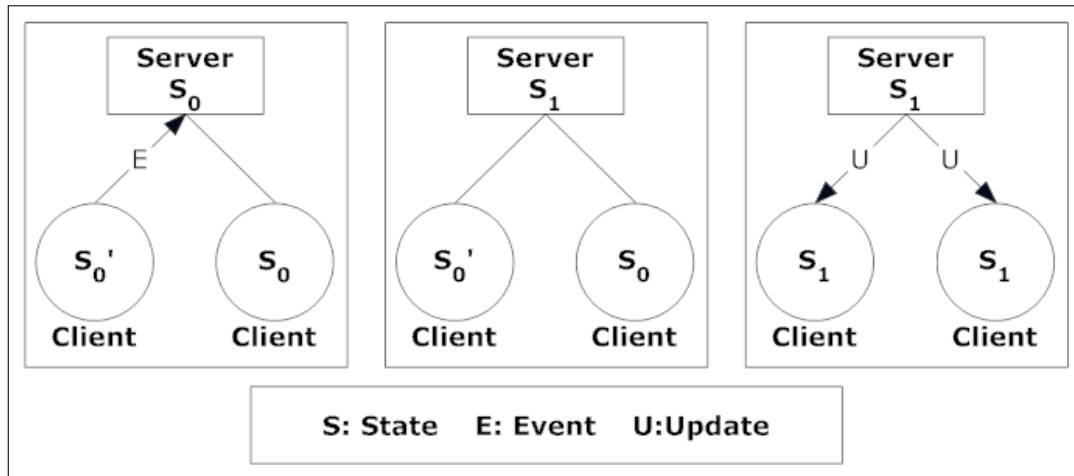


Figure 2.2: Update-Based Model showing the process of a state update.

Therefore, all security benefits of a C/S system apply to this model. Ignoring costs, this model has the benefit of being simple to scale, by just upgrading the server the capability of the system grows. Just about all current MMOGs use this model.

Even in C/MS systems, update-based models are used. The authoritative game state is just distributed across the multiple servers. One example of this is *sharding* where multiple servers each host a copy of the virtual environment. Players are able to connect to any one server and interact with that instance of the world, but their interactions do not carry across to other worlds hosted on other servers. Sharding is a simple way of scaling a system, and allows for a smaller virtual world to accommodate many more people by mirroring it. Even though the full MMOG is distributed across multiple servers, each individually manages its copy of the game state according to the update-based model.

### 2.4.2 Event-Based Model

The event-based model follows a traditional fully distributed P2P model. Every peer in the system has a full copy of the game logic and is able to process any event to produce an update. Each player also houses his own global game state. Using these tools, players transmit their events to all other peers, which are then able to apply their set of the game logic to update their own game state.

In this model, event ordering becomes important. A peer might receive events from multiple different peers that influence each other in a different order from someone else. Processing these events might result in differing game states. Therefore the order in which events are applied are important. It is also not a very

scalable approach as all peers are connected with all other peers, causing traffic to grow quadratically. Finally, this model inherently lacks security, as clients could maliciously influence traffic, state or game logic.

## 2.5 Requirements for Peer-to-Peer MMOG systems

A P2P MMOG must fulfil a number of requirements to properly facilitate the game-play and all its associated aspects. Many of these requirements overlap with the basic components of MMOGs, but focus on their application in a distributed architecture. A 2007 paper by Schiele et al. lays out each of these requirements which are summarised below [2].

**Distribution** requires that the game logic, data and computation are all decentralized and distributed between the peers of the network. Peers are required to store game data pertaining to game state, objects, and mobiles. Furthermore, peers must be capable of processing non-player characters artificial intelligence activities, and capable of applying game logic to player actions. The system requires distributed data management, as well as distributed computation mechanics.

**Consistency** is important in MMOGs as multiple players share the same virtual world. All players active in the world should have the same view of the world and perceive events in an identical fashion. Although strict consistency exists in a C/S system, P2P systems should, as a minimum, aim to meet player expectations. This introduces complicated problems like event ordering in a distributed system to ensure that players do not take actions that contradict each other.

**Self-Organisation** manages the ever-changing availability of devices in the network. Players continuously joining and leaving the game can influence the performance of the network. If a node was assigned a task, and then abruptly leaves, the task remains unfinished and must be redistributed quickly. Alternatively, new players joining offer more computation and storage capability, so they must be incorporated into the system to improve its performance and to balance the load. All of these actions must happen autonomously throughout operation.

**Persistency** of the world's state must be maintained over time. Players effects on the world should be maintained between play sessions, and their own mobiles should be ready for their return. This does not only apply to their own actions, but if other players influence the world in their absence, it should be perceivable upon return. Thus, the state of the world must be stored persistently over time,

and not lost when players depart. This can also be extended further by recording some history of the game state for the purpose of tracking virtual crimes.

**Availability** of the game is especially important for players who pay for the service. This requires that the system have high reliability and high fault-tolerance. The system should also be capable of dealing with issues introduced by network address translation (NAT), firewalls and running multiple clients.

**Interactivity** is a fundamental aspect of MMOGs, having players interact with the game, and receive a direct response to their actions, such as issuing orders, taking part in combat, etc. If this is not available, it can have a significant impact on the gameplay and players perception of the game. This leads to the requirement of maintaining a low latency between players to receive responses to their actions as quickly as possible to reduce experienced delays.

**Scalability** is necessary to achieve to massive numbers of players that MMOGs are titled for. Large numbers of concurrent users all occupying the same virtual world must be manageable by a P2P system. Not only must the system be able to deal with fluctuations and peaks in numbers, but must also be continuously expandable to be able to grow the game further. The possibility of a P2P network to scale naturally counts as a big advantage over the traditional C/S system which requires financially taxing server upgrades. However, this requires special attention as scaling incorrectly could easily overload a node.

**Security** against virtual crimes and cheating is necessary in a game where many players are sharing the same environment. Players seeking unfair advantages might attempt to hack their clients to gain them. Cheating prevention is necessary to keep the game balanced and fair for all players to keep them active and interested. Following this, a means to track virtual crimes such as offensive language or fraud also contributes towards maintaining a large player base. Finally, for an MMOG to have a successful business model, it requires some form of secure accounting system and payment management, resistant to attacks.

**Efficiency** of the peer-to-peer mechanics will leave hardware resources free for processing the usually highly-detailed 3D game. If the game is not up to modern standards, players will be disappointed and move elsewhere. Therefore, the underlying system must operate as efficiently as possible to still be able to contribute to hosting the game without influencing the actual gameplay.

**Maintainability** of MMOGs allows the game to stay updated, often introducing new features as well as patching bugs and exploits that have been discovered. Anti-cheating mechanisms will always need to be updated as players search for new ways to cheat the system. While a C/S system can disconnect all players for main-

tenance to occur, a P2P system needs to perform this on-the-fly while the clients are operating as the hosts while playing the game. This also needs to adhere to the consistency requirement.

These are the full requirements of developing and hosting a fully distributed P2P MMOG. The proposed hybrid system, as a proof-of-concept, specifically focuses on the requirements of distribution, self-organisation, interactivity, scalability and efficiency.

## 2.6 Interest Management

### 2.6.1 Overview

In a traditional client/server MMOG setup, players are often located at entirely different places throughout the virtual world, and are only aware of what is relevant to them at that point in time. The amount of information that a player is made aware of at any time is controlled by the server. This is done in an effort to reduce the traffic that is generated and distributed to each client. If a player were to receive updates about the entire world, it would serve little to no purpose as a player is only actually able to see and interact with a smaller area around themselves at any given point in time. In the context of a game, the player's ability to only see and shoot an arrow a limited distance translates into the player's area of interest.

As a pure peer-to-peer system aims to eliminate the need for a server, the same rule for a client/server system applies for a peer-to-peer system: a player should only ever receive updates about the area in which they are situated and able to interact with. This serves to reduce the total traffic experienced by peers in the network, rather than facing the problem of overloading peers by transmitting every event occurring in the world to every other peer connected to the game.

Interest management (IM) schemes aim to facilitate this server feature in a peer-to-peer network. There have been many articles published in the last decade which have presented different methods of defining a player's area of interest and ways to distribute events and updates amongst the peers. Comparisons are often drawn between the different schemes and provide interesting insight into which are suited to specific situations. Most IM schemes fall into one of three models as fully detailed by Lu Fan in 3.2 of [36]: the spatial model, the region-based publish/subscribe model, and the hybrid communication model.

In the rest of this sections, we first define *area of interest*, address the concept

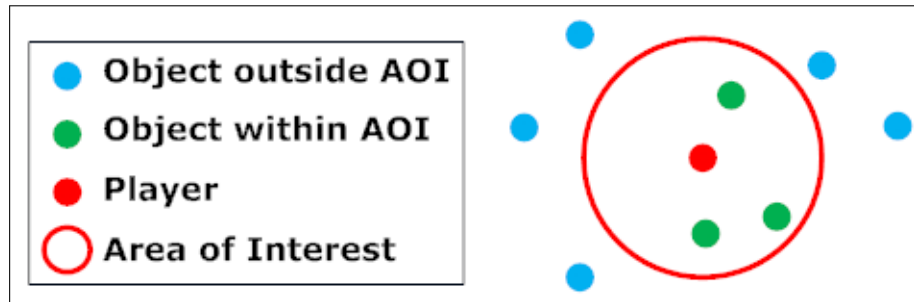


Figure 2.3: Area of Interest surrounding a player

of a *supernode*, and finally summarise the models as described by Fan.

### 2.6.2 Area of Interest

Across all implementations of Interest Management, the concept of an Area of Interest (AOI) exists [36][29][37][38]. An AOI is described as an area surrounding a player that he has influence over, or influences him. Often described by a circle or rectangle surrounding a player's current position, everything within the area is considered to be of the player's interest as it is within range of his perception. The complexity of the AOI can vary from every object within the area, to limiting it to what is currently visible by the player, to including prediction into what will next be visible. An example of an AOI surrounding a player is shown in Fig 2.3. The concept is similar to a C/S MMOG system where a server only sends updates to a player pertaining to what is nearby them.

The AOI serves as a limit to what updates a player should receive at any given time. If something is outside of the player's AOI, there is usually no value in updating them about changes to game state in foreign areas. The exact definition of an AOI can be entirely dependant on the bounds of the game it is being applied to. In a paper by Bharambe et al., they develop the concept of an Interest Set which serves the same purpose as an AOI, with the difference that it concentrates on what the player's attention is on, rather than what is in its surrounding area, and uses that to limit the traffic to peers [39].

### 2.6.3 Supernode

Another IM concept which appears in many schemes is the *supernode*, or *superpeer*. Originating from the evolution of P2P networks, the term is used to describe a peer which has been elevated to a position of higher power, usually responsible for more

tasks than the standard network peers. Used in aid of network efficiency, supernodes are often the nodes with the most computing power which are selected to take on server-like responsibilities to assist standard peers [40]. For example, a supernode may compile a list of all objects available on its nearby standard peers, exchange lists with distant supernodes, and use this expanded knowledge to locate a particular object more efficiently. This reduces overhead and redundant traffic generated by standard peers.

Supernodes are usually responsible for a smaller subset of standard peers. This requires that the full set of peers be clustered or grouped into smaller groups. A single peer in the group is then selected to be the supernode of the group. This improves the organisational structure of the network by allowing every peer to be served by one supernode, and eliminating any redundancy of contacting multiple supernodes. Having a more reliable knowledge of the current members of the network, supernodes are able to act as entry and exit points of the P2P network. Some architectures suggest using multiple different types of supernodes to help facilitate and maintain the P2P network successfully [41].

The selection of supernodes in a network presents an interesting problem. There are many criteria from which a supernode could be selected such as: computation power, network speed, position in the network, position in the virtual environment, reliability, active duration, or proximity to other supernodes [42].

Along with the benefits of a supernode, having an elevated peer can introduce a number of risks for the system. Any supernode becomes a point of failure in a network, and therefore additional mechanisms must be instated to increase the fault-tolerance of the system. The ability to exchange supernodes and recover from losses is essential. Supernodes also become the targets for malicious attacks on a network, such as denial-of-service attacks. If enough nodes are removed from the network to overcome any recovery mechanisms, the entire network could collapse [43].

Because the goal of Interest Management in P2P MMOGs is to reduce the traffic generation, a supernode is able to assist in managing the flow of updates between nodes based on their areas of interest. Furthermore, it is able to communicate with other supernodes to exchange summarised information about nodes under supervision. Grouping of nodes also forms a fundamental mechanic of some IM schemes. Therefore, in many cases, supernodes are a useful mechanism for interest management schemes.



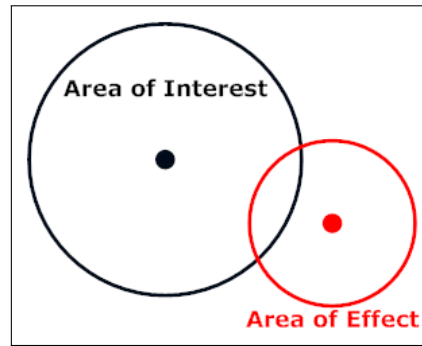


Figure 2.4: Spatial Interest Management Model

### 2.6.4 IM Model: Spatial

The spatial model completely revolves around the location of objects (mobiles or items) in the world. An object typically has an area of interest surrounding itself at all times, and when that area of interest intersects any other object's area of effect (the area surrounding an object where the effects of that object are experienced), then they become aware of each other and exchange data. However, this model requires that all objects regularly exchange positional data. This model is shown in Fig. 2.4.

Many of the more computationally complex spatial algorithms are those that calculate an area of interest based upon a *Voronoi Diagram* [44]. Each player has a dynamic Voronoi cell which borders on other players' cells. The player is then able to reduce the connections it maintains to only neighbours occupying cells within the player's own area of interest. The connected neighbours then inform the player when there are changes within its area of interest. An example of this setup is shown in Fig. 2.5. This requires regular computations to be performed by each peer to calculate its own Voronoi diagram surrounding the player based on the player's proximity to others. Voronoi based interest management appears in [29][45] and is also used in a comparison with alternative schemes in [46]. As the spatial information of objects are only exchanged between connected nodes, Voronoi Diagrams aim to reduce object location traffic of the traditional spatial model at the cost of increased computational complexity.

### 2.6.5 IM Model: Region-based

The region-based publish/subscribe model divides the virtual world up into static regions and data is only exchanged between objects within the same, or neighbour-

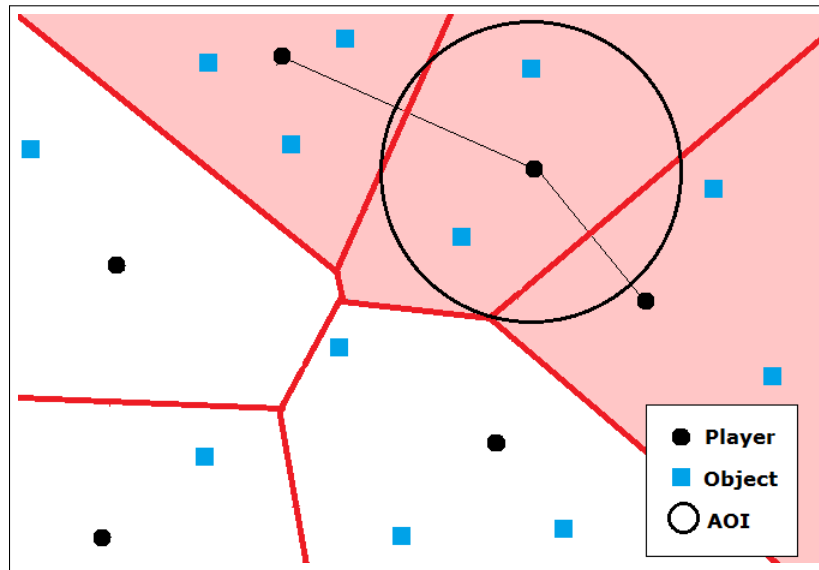


Figure 2.5: Example Voronoi cells where red lines are cell borders. A player is connected to two neighbours of cells intersecting his AOI.

ing, regions. A peer then subscribes to the regions with which its own area of interest intersects and receives all updates pertaining to objects in those regions.

The size and shape of the static regions can have a significant effect on the amount of data transferred between nodes. If regions are larger than what the player is interested in, the player will receive a lot of information that has no relevance to its current perceived state. It has become widely accepted that hexagonal regions in place of square regions can already result in a significant reduction in traffic [47][37][38]. This is largely due to the resemblance of the hexagons to a circle. Fig. 2.6 shows the difference in subscribing to neighbouring regions of different shapes. Even more advanced shapes based upon triangles are presented in [38], but the more advanced shapes result in higher complexity computations.

### 2.6.6 IM Model: Hybrid

The hybrid communication model divides the virtual world into static regions, and promotes a peer in each region into a supernode, which effectively becomes the administrator of that region. The supernode is then responsible for managing all spatial-based interest management bounded within that region.

A good example of a hybrid model is *MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games* [37]. In this implementation, static hexagonal regions are defined, and supernodes are

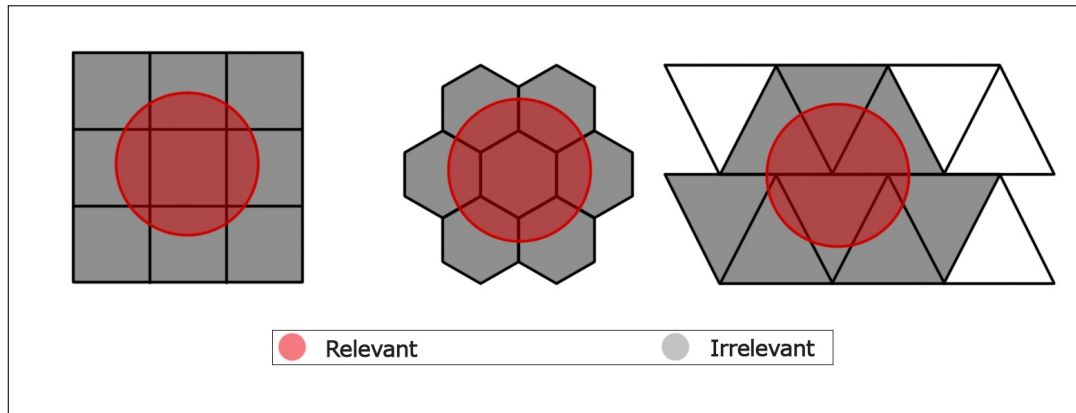


Figure 2.6: Region Shapes: Red represents the player AOI while grey is unnecessary additional region information.

promoted to administrate these regions. The addressing of regions and supernodes are handled by Pastry, a distributed hash table built into the P2P network [24]. When a player enters a region, it sends a join request to the supernode, who responds and informs all other nodes within the region about the new player. The supernode then manages the interest of the group members and communicates with adjacent regions. Hybrid models attempt to exploit the benefits of both other models while avoiding their downfalls.

## 2.7 Existing System

### 2.7.1 Requirements

As the goal of the proposed middleware plugin is to transform an existing system without modification of the client or server software, a pre-existing system is necessary to base it upon. The following requirements were deemed important for the selection process:

- The system must use the Client/Server network architecture.
- The system must offer a freely available open-source server. Open source will help with designing the middleware plugin and as the system will distribute game logic amongst the peers, it will require the use of the server code.
- The system must offer a freely available client, preferably with an open-source option.
- The protocol must be well documented including detailed information of the

system's packets.

- The system must be or have been a commercially successful MMOG.

## 2.7.2 Selecting a System

Two C/S MMOGs were initially identified as potential systems: World of Warcraft and Ultima Online. Both were globally the most popular MMOGs at different times, and both had emulated server software available, along with an easily accessible client.

Further research into both possibilities revealed that while World of Warcraft is more recent, the emulated server software was poorly developed, unstable and lacking in documentation. Ultima Online, however, could offer a well developed, stable, open-source server with an active community. As an additional benefit, an open-source client for UO was also available. Therefore, Ultima Online was chosen as a base system. The standard UO client is available for free download from the developers at [48]. *RunUO*, the open-source, reverse engineered server is available from [49]. *Iris2*, the open source client, is available from [50].

## 2.7.3 Automation

Many macro script applications were developed throughout the course of UO's lifespan, allowing players to automate their activities in-game. *EasyUO*, the macro application available from [51], is a well established macro scripting platform which allows for complex automation of any player activity if necessary. In the context of the hybrid system, this is useful for facilitating the experimental evaluation of the system as game clients are required to operate as they would in a real system and generate interactions accordingly.

## 2.7.4 Architecture and Operation

Ultima Online was originally developed with a client/multi-server architecture using the sharding principle. When a new client connects, a centralized login server authenticates it before redirecting it to the shard of its choice. The client then connects to the shard and enters into that instance of UO's virtual world of Britannia. It follows the update-based consistency model, where an acting client sends their event to their world's hosting server. The hosting server applies game logic to produce an update which is then returned to the client for its game state to be updated. The server hosts the global authoritative world state and is expected to process every

client event into an update. RunUO modifies this approach slightly by disregarding the sharding principle, and hosts a single instance of the world in the same package as the login server.

When the login process is complete, a large set of updates is sent from the server to the new client pertaining to its current position in the world. All these updates are applied to the default starting game state to update it to the current authoritative view of the world. The client holds no information from its previous session, therefore the server sends sufficient updates at login to bootstrap any client from nothing to the current state.

In Ultima Online, characters are known as mobiles and all parameters of their state are stored server-side. When a player has a mobile of its own, it is only informed of selected information by the server, such as its current position and the status of the mobile in terms of how strong, fast and intelligent it is. The server maintains its security by only allowing authenticated clients to connect to it. If the authentication check passes, the player is able to select a mobile to play, and that mobile is associated to its connection. The server strictly only accepts events of a player's mobile if it originates from the connection that was associated to it upon login. This mechanic prevents malicious players from sending rogue packets pertaining to other players' mobiles.

### 2.7.5 Positional System

The UO world functions on a discrete 3-dimensional co-ordinate system, where each point has an associated set of  $(x,y,z)$  integer co-ordinates. Although the world is 3-dimensional, the vertical component is often ignored for many purposes. The entire world is divided into a collection of 7 large maps, which are sub-divided into sectors and regions.

Sectors are the result of maps being divided up into equal-sized rectangles along the horizontal plane, where none overlap. They facilitate organisation of data storage on the server, where objects and mobiles are stored in a sector according to their horizontal position. A sector is only owned by a single map.

Regions are virtual areas of variable size that can overlap and define gameplay rules for the area. Examples of which would be defining an area as a farmland, a guarded area, or a dungeon. A sector can have multiple regions, and a region can span multiple sectors.

### 2.7.6 Interest Management

Built into the UO server is a method of interest management that defines which clients should receive updates to an event. When an event is generated by a client, the server processes it and locates the point in the virtual world where the effect of the update occurs. From this point, a radial search is performed outward until a radius is reached where the originating point can no longer be perceived by players. Between the start and end points of the radial search, an update is transmitted to the associated client connection wherever a mobile is found.

The radius around the originating point can be defined as an area of effect, and every player mobile found within the area of effect receives the update. Certain events might produce more than one area of effect, which results in same procedure being repeated until all affected player mobiles are identified.

### 2.7.7 Limitation of Modification

Once compiled, most of the game logic, packet definitions, networking capabilities and processing rules are all sealed within the server. Additionally, the server offers a scripting interface which allows for extensions to the base game within the limitations of the script capabilities. The main use of the scripting is to introduce additional player commands. Players are able to input textual commands into their chat window which transmit to the server resulting in the script being executed. Therefore, for the purposes of this project, the limitation on any modifications to the server are through the scripting interface only.

The UO client is an entirely closed piece of software with no modifications available. For the purposes of this project, the client needs to be used entirely in its original state. In its original state, a user is able to specify the server IP address that the client attempts to connect to.

### 2.7.8 Imposed Constraints

The use of UO imposes two constraints on the design of the middleware plugin and hybrid system.

If a supernode has to send aggregated updates on behalf of the nodes in its group, it will be necessary to add a client command to the server scripting interface, since the UO server natively does not support such aggregated updates. The design impacts of this constraint are detailed in section 3.5.5.

As mentioned in the Architecture and Operation, the server security requires that events from a player mobile originate from the client connection that the mobile was associated to upon login. Therefore, for any peer to communicate with the server on behalf of another peer, the connection has to be rerouted through that peer, and thus must re-login to the server from that peer. The design impacts of this constraint are detailed in section 3.4.

## 2.8 Summary

In this chapter the background concepts of the middleware plugin and hybrid system were presented. A brief history of MMOGs and their traditional centralised client/server architectures were examined. Latency and bandwidth, the two important factors affecting an MMOG architecture were discussed, and some of the current solutions for improving these factors, such as BigWorld and Kiwano, were presented.

Following this, the focus was shifted towards peer-to-peer architectures, their origins, and their usage in MMOGs. When examining the original P2P overlays, the concept of a distributed hash table was introduced, and specific attention was given to the communication protocols, namely direct communication and multicast. Investigating the use of P2P in MMOGs revealed many existing architectures, some fully distributed while other using a hybrid approach. No solution was found to address the problem of introducing P2P aspects into an established C/S system. A study into P2P MMOG traffic suggested that a pure P2P architecture would not be a feasible near-term option due to bandwidth constraints.

After covering these base concepts, the attention was shifted to some of the design concepts of MMOGs. The game state management models were examined, with the update-based model being the most prevalent in current MMOGs. All the requirements of P2P MMOGs were addressed, and distribution, self-organisation, interactivity, scalability and efficiency were identified as the primary requirements for the hybrid system. Interest management concepts were discussed as a method of providing scalability to the system. The important IM concepts of a supernode and area of interest were defined, and then various IM models were examined.

Finally, a discussion of Ultima Online as the existing system to develop upon was presented. The client/server architecture of UO was examined, its update-based model defined, and some basic operational characteristics discussed. Specific attention was given to the co-ordinate system and inherent interest management

as they factor into the design decisions of the middleware. Finally, the limitation of modification and the constraints that result from the limitation were presented in detail, which heavily influence the design of the middleware plugin and hybrid system.

In the following chapter, many of the design choices are based upon concepts that were introduced throughout this chapter.



# Chapter 3

## Design & Implementation

### 3.1 Introduction

The middleware plugin is tasked with transforming the MMOG's network into a hybrid of client/server and peer-to-peer topologies. It is necessary for the system to be designed on multiple levels to successfully implement this.

This chapter presents details on different levels of the system design. It begins with an overview of the hybrid system. This serves as a broader view of the system from the highest level, giving context to the rest of the chapter. It lists some of the major design decisions of the hybrid system that led to how the system topology is structured to facilitate the aims of the middleware.

Next, a high-level discussion of the purpose and design of the custom interest management scheme is presented. This is followed by an examination of the network architecture in terms of its requirements and functionality.

Finally, more detail is given to the lower level design of implementing the software itself. Each of the software modules are presented individually: the software core, the networking, the game logic, and the interest management. The design and functionality of each are described.

### 3.2 Hybrid System Overview

The middleware plugin is developed as a proof-of-concept, focusing on introducing specific aspects of P2P MMOGs into an existing C/S architecture. For the purposes of the experiment, a number of P2P requirements were consciously ignored for being outside the scope of the desired outcomes. The design choices were based upon the

two primary goals of the system: (1) to reduce server load and (2) to reduce player latency.

Because the majority of traffic in Ultima Online originates from movement events and updates, it was chosen as the game mechanic for distribution so that the performance of the proposed system can be evaluated. If the greatest producer of traffic can be reduced, the server should benefit immediately. All other game mechanics are the responsibility of the server, as they would be in a traditional C/S system. This includes mechanics such as speech, spell casting, item interaction, non-player-character processing, etc.

For mechanics to be distributed, the server must receive periodic updates about the changes in the game state, so that other non-distributed game mechanics which it processes are consistent. Therefore, supernodes are included to aggregate the updates of all the peers they are responsible for and send batch updates to the server containing the aggregated information. By creating batch updates, the overall system bandwidth usage is reduced from the case where each node sends its own updates. This results in the system using a heterogeneous peer-to-peer network structure, where there are two levels of peers: a slave and a supernode, and every peer has the ability to assume either of the two roles. For the purpose of this study, the issues associated with supernode selection are ignored and simply the first node in a group is selected to be the supernode of that group.

As the server security restricts player events originating from connections not associated to that player, and the supernode is responsible for updating the server: slaves are required to reroute their connections through the supernode toward the server. An example of the hybrid topology is shown in Fig. 3.1a. This ignores any connections resulting from shared areas of interest.

In a traditional C/S MMOG, the server hosts the authoritative copy of all game state. The server is responsible for verification of any events received from players in the world before producing an update and applying any changes to the game state. As part of the distribution strategy, the game objects representing player mobiles are replicated on each peer, and the authoritative copy is held by the owner of the mobile. A slave always sends its movement updates to the supernode who applies the update to its own game state. The supernode is then responsible for sending the update to the server where it can be applied to the server's global game state. Verification of mobile changes could take place on the supernode's replica, but because security and cheating mitigation is outside the scope of the experiment, it was not included for this implementation.

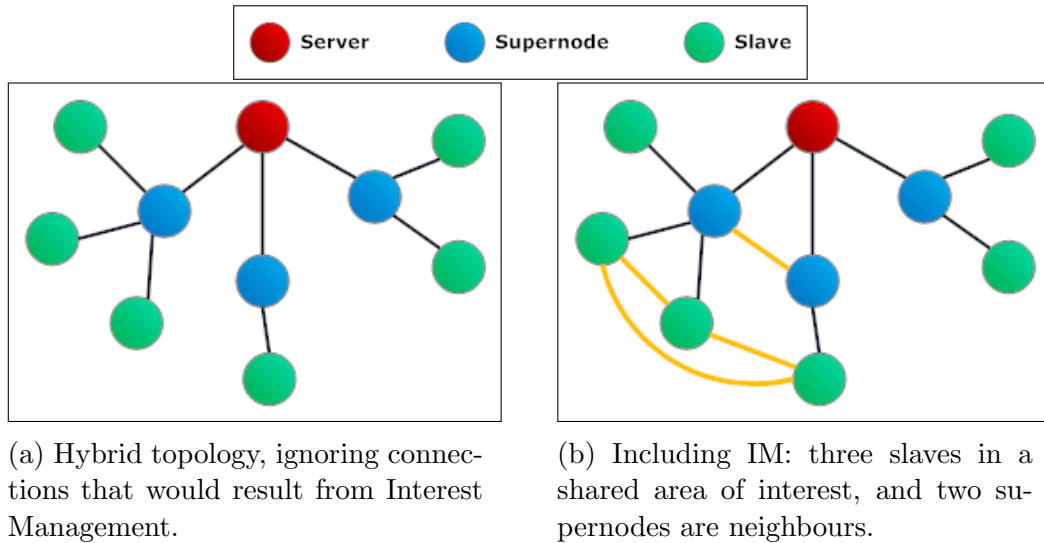


Figure 3.1: Example Hybrid Topology

In an effort to improve scalability, an interest management scheme is included in the system. MOPAR, described completely in [37], is a good candidate for basing the hybrid system's IM scheme on as (1) supernodes are already included for server updates, and (2) a region-based IM scheme is similar to the sectorized architecture of the UO server. The virtual world is divided into hexagonal regions where peer groups are formed for each occupied region, each containing a single supernode and any number of slaves.

The network uses a direct communication overlay, where every node has a unique identifier derived from its IP address, and connections are established directly between peers wishing to exchange data. The overlay is aided by a directory server which is hosted in the same place as the game server and serves as an entry point into the overlay by providing the address of a supernode to connect to. Joining the overlay occurs after the location of the player has been established from the game server, and the correct supernode can be identified according to its region. The supernode commands its slaves on how to manage their connections to other peers according to their individual areas of interest. Fig. 3.1b shows an example topology where IM has been applied: the two connected supernodes are in neighbouring regions from one another, and the three connected slaves are within each others areas of interest.

## 3.3 Interest Management

### 3.3.1 Introduction

The primary function of IM is to provide scalability to the system. In the hypothetical case where no IM is present in a P2P environment, no peer would be aware of which others their events or updates apply to, and which they should be sent to. This would result in peers either missing events and updates resulting in gaps in gameplay, or peers receiving an excess of unnecessary data causing them to be overloaded and resulting in increased latency and possible timeouts.

IM allows peers to maintain connections only with the relevant players and only send and receive traffic necessary for maintaining consistent local game state. This prevents peers from being overloaded with irrelevant data. While an overload might still occur if an area of interest becomes overpopulated, most dynamic IM schemes are able to adjust accordingly up until a point where the peer would anyway be overloaded in the standard C/S setup.

### 3.3.2 Aims

The primary aim of the interest management scheme is to reduce the data bandwidth experienced by peers in the network by limiting the events and updates communicated to a player to only what is considered relevant to that player's current location and state. It is critical to the scalability of the system.

For traffic relating to distributed game mechanics, the middleware should take into account a player's area of interest and filter out packets which do not hold relevant information for that area. These tasks should all be performed in a way that does not negatively influence the latency experienced by players within the network.

### 3.3.3 Design

The hybrid system's interest management scheme is based upon the hybrid model which was discussed previously in section 2.6.6. The scheme is largely based on MOPAR [37], an interest management scheme used for pure peer-to-peer systems by combining a DHT with unstructured P2P communications. It makes use of region-based hexagonal divisions of the virtual world to organise peers into smaller groups. Each of the groups have a master node as well as a collection of slave nodes.

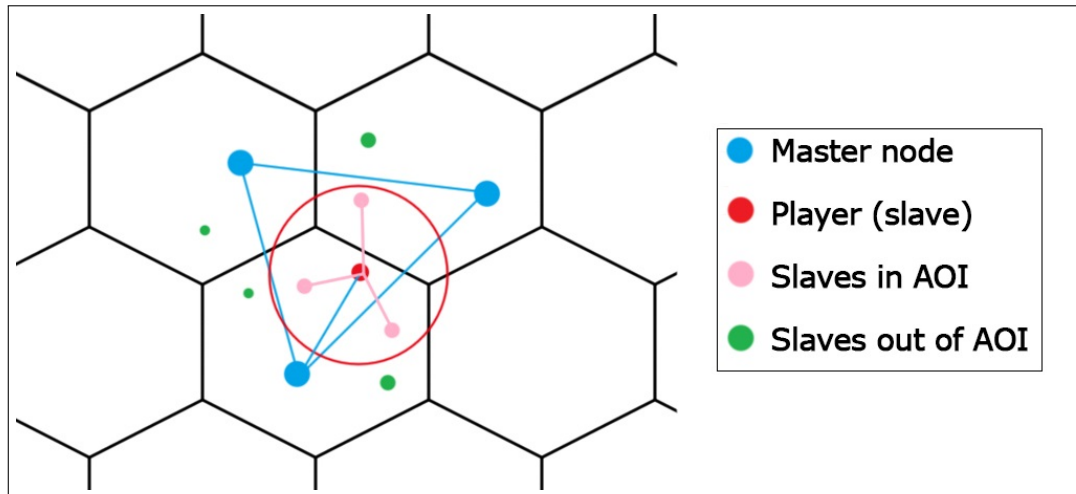


Figure 3.2: Example MOPAR connectivity for a player in a region

### Supernode

The master node, which is synonymous with a supernode in this case, acts as an administrator of the group. The master node of a region maintains connections with the master nodes of all the neighbouring regions surrounding its own and each exchanges group lists containing the location of players within their region. Thus, master nodes have a much wider knowledge of the surrounding area than the slave nodes. Through this knowledge, the master nodes are able to manage the areas of interest for their slaves, putting slaves in connection with each other when they are within range, and terminating their connections when they are not within range. Therefore a slave is always connected to their master, and only to other slaves within their own area of interest. An example of this connectivity is shown in Fig. 3.2.

The interest management in the hybrid system is similar to what has been described by MOPAR, with a few alterations to accommodate the hybrid system instead of the pure peer-to-peer system it was designed for. In addition to the master node duties, the supernodes also remain in connection with the server and are responsible for providing the server with periodic state updates of the distributed mechanics which have been processed by the peers. In turn, the supernode also filters the redundant client updates which are distributed by the server in response to the supernode's state updates.

## Directory Server

A third node type exists in traditional MOPAR, known as a home node, and is used for routing peer connections towards an appropriate master node based on the players location. This is achieved by using the DHT functionality of MOPAR's overlay. In the hybrid system, the home nodes and DHT are replaced by a directory server. A peer queries the directory server to find which supernode it should connect to, and then connects directly to it and becomes a member of that group. Since the network already contains a centralised server, it is a negligible addition to include a centralised directory server.

## Migrations

Migrations occur when a player's mobile leaves a region to enter a new region. There are 4 types of migrations which can occur depending on the current roles of peers in the associated regions:

1. *slave* → *slave*
2. *slave* → *supernode*
3. *supernode* → *supernode*
4. *supernode* → *slave*

Each migration type presents a number of additional effects on the migrating peer and other peers in the region. In (1), when a slave migrates to a new region with a different supernode, it simply has to assign itself to that new region's supernode. Conversely, in (2), if a slave migrates to a new empty region, it must undergo a transformation into a supernode role.

In (3), when a supernode leaves a region to become the supernode in the new region, it requires that a replacement supernode be selected in the old region, and all remaining slaves reassign themselves to that supernode. Similarly, in (4) when a supernode leaves a region to become a slave in another, the old region must undergo the same procedure. In addition, the migrating peer must undergo its own transformation and assign itself to a supernode in the new region.

The software implementing the interest management scheme is discussed further in section 3.5.4.

## 3.4 Network Architecture

### 3.4.1 Aims

As the middleware introduces peer-to-peer networking capabilities into the game system, the design of the network architecture is important from both a physical and virtual perspective.

- The goal of the network architecture is to cater for all the connectivity requirements of the system and to efficiently facilitate the routing of data packets.
- The architecture setup should allow communication to go uninterrupted between the server and the client when required, but should always be under observation in the event that certain packets are to be extracted and recorded or processed.
- For the arrival of new players into the network, an entry point must be provided. This will allow any new player to connect to and become a part of the peer-to-peer segment of the network and participate in the distributed processing.
- As there can be a varying number of players in the game at any time, the architecture must be scalable. It should be able to handle the case when there are only a few players as well as when there are a few hundred. Thus it must be compatible with an interest management scheme to limit the amount of traffic. The interest management design was discussed in section 3.3.
- As one of the primary system goals is to improve the game experience, latencies should be reduced between players located within shared areas of interest.

### 3.4.2 Design

A conceptual diagram of the proposed network architecture is shown in Fig. 3.3. In the figure, each block represents a physical computer running software modules which are indicated by shaded ovals. The game server, running on the server machine, hosts the game's virtual world. A client machine has both an instance of the game client and middleware software running. A directory server, also hosted as a part of the network, stores important information regarding members of the network and acts as an entry point into the peer-to-peer network.

The middleware, which introduces the peer-to-peer segment of the network, is responsible for establishing all inter-connectivity between players. It maintains connections to other middlewares, as well as providing a dedicated route for the client to communicate with the server while still being able to monitor all data that passes

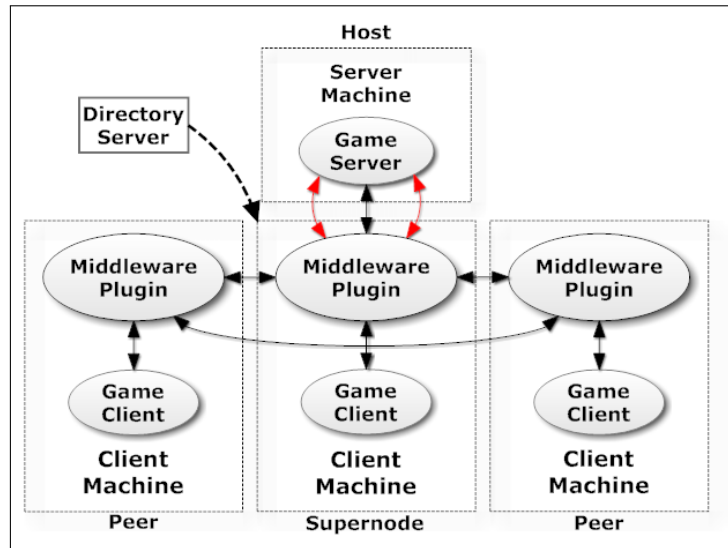


Figure 3.3: Modified Hybrid Network Architecture

through. This configuration ensures that each client has a path to the server and can communicate without interruption. In addition to this, peers can communicate directly between each other, allowing for a more efficient routing of game information when appropriate.

### Interest Management

As scalability is an important factor, the architecture is designed to accommodate the interest management scheme. This is introduced by the ability to form smaller administrated groups of players. When a group is created, its founding member is designated as a supernode (previously described in section 2.6.3). Supernodes provide the ability to administrate peer groups, to facilitate connectivity between members, to manage communications between different groups, and to perform packet filtering. By intelligently creating groups of players, it can reduce overall traffic by ensuring that players only receive game information that is relevant to them.

### Connection Routing

Two major constraints of the UO server influence the design of network structure. *Firstly*, as the server only accepts actions of mobiles who are attached to a specific socket connection, a supernode would ordinarily be unable to communicate with the server on behalf of its group members. This precludes the possibility of aggregated



or batch events/updates being sent to the server by the supernode on behalf of the group. *Secondly*, when the server receives some form of aggregated event or update describing what has already occurred amongst the P2P network in the past, the server still processes them as if it had just received them in standard C/S operation, causing it to produce redundant updates which are transmitted to the relevant clients. If redundant updates arrive at clients and are applied to their game state, they could contradict the current state and disrupt the flow of gameplay. To overcome this in a distributed manner, significant overhead would be required to keep all peers informed of which updates needed to be ignored, and which were a result of non-distributed game mechanics processed by the server.

Therefore, to cater for these two constraints, all members of the group route their connections through the supernode, giving the supernode access to all communicated data. This allows the supernode to more effectively track player activity, communicate with the server on behalf of the group members, and act as a point of filtering for events and updates that are not necessary to be disseminated amongst the group. Furthermore, as the supernodes are the only peers responsible for updating the server, cheating mitigation techniques can be applied primarily at those points in the network to ensure fairness if necessary.

To facilitate connection routing, when a middleware reroutes its connection via the supernode to the server, it is required that the middleware autonomously performs a *relog*. A relog is the action of logging out and back into the game in rapid succession. This allows the connection between the supernode and server to be correctly authenticated.

## Directory Server

As previously described in the IM requirements, the directory server acts as a static point of entry into the P2P network. It maintains a list of supernodes and directs peers to their relevant addresses. Hosting such a server is trivial as there is already a game server present in the architecture, and the directory server can be hosted alongside it.

## Joining

The process of *joining the network* starts once a player has started both the client and the middleware software instances on his machine and he begins the login pro-

cess from the game client. An internal connection from the client software to the middleware running on the same machine is established. The middleware connects to the server and proceeds to login to the game up until it is aware of the player's position. Using the position, the middleware autonomously queries the directory server and is provided with a supernode's address. The middleware terminates its connection to the server and connects to the supernode who proceeds to facilitate the player's integration into the P2P network. The supernode completes the connection to the server on behalf of the middleware and provides the middleware with a list of other peers within its AOI. The middleware then proceeds to establish connections to the relevant peers. This completes the login process. In the event that no supernode for the region is present on a player's login, a new group is formed and the new player assumes the role of the supernode.

### Leaving

The process of *leaving the network* occurs when a player logs out from the game and the client software terminates its connection to the associated middleware. On detection, the middleware proceeds to gracefully terminate its connection to the supernode and other peers within the group.

This action becomes significantly more complex if the exiting player is a supernode. On detection of a client termination, the supernode middleware must proceed to identify and notify a new peer to assume the role of the supernode. It informs all other peers in the group that the supernode duties are being transferred and the peers are required to reroute their connections through the new supernode. Once the new supernode has taken control of all the connections, the terminating middleware completes its shutdown.

### Virtual Nodes

On the software level, the network can be divided up into three virtual node types. The *Server Node* represents the dedicated game server which hosts the virtual environment. A *Middleware Node* represents a middleware plugin in the network. Finally, a *Client Node* represents a player's client application which would usually connect directly to the game server in the absence of the middleware. These three nodes are the software representation of the network points shown by the ovals in Fig. 3.3 and are discussed further in section 3.5.3.

## 3.5 Implementation

The previous section discussed the design requirements of the hybrid architecture, as well as a high-level overview of the desired interest management and network topology. In this section, the implementation details are explained.

### 3.5.1 Overview

The system is divided up into six modules, as shown in the overview of the middleware software in Fig. 3.4. These modules make up the full functionality of the system. The *Core* is the central point of the system and maintains the interconnectivity between the other modules. It is also responsible for capturing statistical data. The *Networking* module provides all the networking capabilities of the system, including the P2P aspects of the network, such as grouping, storage of node information and packet routing. The *Local Game Logic* module is the collection of game processing functions specifically implemented on the middleware to process events into updates and store game state. This module also manages, sorts and filters many of the packets received through the networking module. It interacts directly with the *Logic Interface* module which is used as an abstraction interface between the local game logic and the libraries imported from the game server. These libraries are located in the *Imported Game Logic* module which houses the server's processing capabilities. This allows the local logic to call functionality from the server logic for tasks such as processing events. Finally, the *Interest Management* module introduces functionality for regional divisions; supernode slave management; and AOI and neighbour communication. A lot of the individual modules' functionality is dependant on the functionality offered by other modules.

Every peer has the full functionality of the middleware available to them as they might be required to fulfil any role in the system. The total set of functionality can be divided into a base set, used by both a slave and a supernode, and an extended set that is used only by a supernode. Each of the modules are made up largely of base functionality, and extended to facilitate supernode duties.

### 3.5.2 Core

#### Aims

The core module operates as the central point of the software system. Its first duty is to handle the startup of the system, including the initialisation of all the

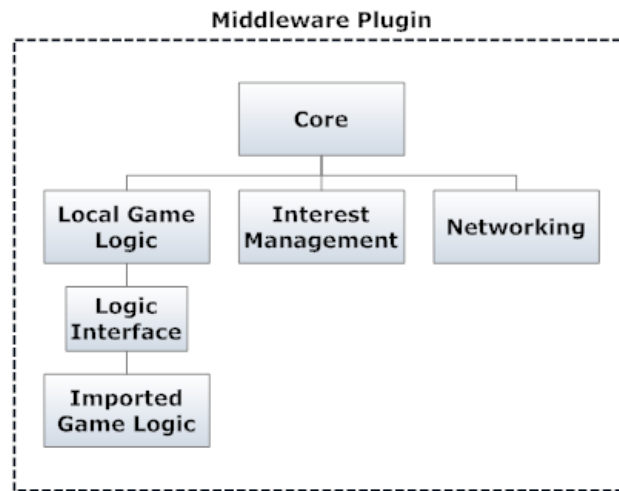


Figure 3.4: System Modular Overview

modules. This involves establishing internal communications between the modules, and providing a means for sharing of data. The core must also handle the graceful shutdown of the system by prompting each module as well as properly disposing of all used resources.

The core acts as the central processing unit of the software system. It drives processing throughout the system's modules in an organised fashion that is robust against multithreaded implementations. This is important as many aspects of the system will be operating on multiple threads and through asynchronous networking actions.

Finally, as this is an experimental system, the core must provide a method for data collection that can be saved to secondary storage for analysis at a later stage.

## Design

As the core is the channel through which all communication takes place, all global parameters are stored in the core module. These are accessible to all other modules in the system.

Managing multiple threads in a robust manner can be achieved through performing all processing at a central location. Therefore, the use of `Queue` data structures as processing queues is used extensively. Any data packet, connection or node that requires any nature of processing is placed in a respective queue catering for its type, and each queue is processed consecutively with each iteration of the processing cycle. The core is responsible for initialising and maintaining the thread which processes each queue as it is required. There are three major queues: (1) New Con-

nections, (2) Packet Processing, and (3) Outgoing Packets. The queues are fed by the Networking module, the methods of which is discussed in the following section.

A statistics module provides a place for timestamped events to be recorded. A record call can be inserted anywhere in the system with arguments, where it will be triggered and written to a buffer. When the application is shut down, the buffer is written to secondary storage. Throughout the execution of the system, the statistics module collects different measurements pertaining to the analytical metrics chosen for evaluation of the system.

The startup of the system involves initialising all of the previously mentioned core aspects as well as all the other modules with the relevant properties and establishing channels of communication between them. The core performs similar activities for shutting down the middleware gracefully by closing the statistics activities, prompting each module to dispose of itself, and finally disposing of the remaining resources. This, in turn, requires that each module is capable of cleaning up its own in-use resources.

### 3.5.3 Networking

#### Network Actions

For the networking module to establish, manage and maintain the network architecture of the hybrid system, it should be able to perform the actions below.

- **Join** Connect to the P2P segment of the network. Automatically orientate itself within the architecture and assume a designated role.
- **Leave** Detect and disconnect from the P2P network. Provide enough information on exit for other peers to re-arrange the network as necessary.
- **Listen/Accept** Listen for new connections from a new client, or other peers in the network. Accept new connections and handle them accordingly.
- **Send/Receive** Routing a packet from one peer to another within its AOI. Routing packets between supernodes. Having a dedicated communication channel between the client and server. Accepting packets from any open connection.
- **Form Group/Disband Group/Migrate** Starting a new group of peers, terminating a group of peers, and being able to migrate between groups.
- **Transform** Changing roles between supernode and regular peer duties.
- **Process Packet** Process received packets according to registered packet handlers.

## Aims

The networking module introduces and facilitates all connectivity, communication and the transfer of data between network nodes. This fundamentally requires that the networking module is able to send and receive packets of data. Therefore, the module must be able to address specific nodes and be able to route data to them, either directly or through other nodes. This requires that nodes' information is recorded within the module. This is of particular importance when routing data between the client and server, for the portions of traffic pertaining to non-distributed game mechanics.

As only certain game mechanics are being distributed amongst the peers in the hybrid system, the networking module is required to distinguish between packets and execute them according to their contents. This requires for specific packets to be registered with custom defined handlers. In addition to this, a default action for unregistered packets or packets without specific processing needs must also be included. This will enable the middleware networking module to effectively filter packets as intended.

For the interest management scheme to be implemented, both peer grouping and supernode functionality is required. Each peer must maintain a record of peers they are connected to resulting from IM, and be able to communicate with them efficiently. Supernodes require additional capabilities to filter and redistribute packets as required; to maintain a record of peers in their group and a record of neighbouring supernodes; to perform administrative tasks for peers joining and leaving their group; and finally to be able to handover their duties to a new supernode in the event of the player departing the game, or a migration.

## Design

The networking module was designed with a number of objects and sub-modules with the ability to offer the system the required network actions. The following elements are discussed: Nodes, Pairs, Socket Configuration, Packet Handling, Peer Grouping, Migrations and Listening.

**Nodes** For the software to address specific nodes in the network, as well as record information pertaining to that node, each virtual node in the network is represented by a `Node` object in the software. The three virtual node types (server, middleware and client) were discussed previously in section 3.4. It is these `Node` objects that form the basis of the middleware networking module. Every socket connection that is

established is assigned to an independent **Node**, and all interaction with the socket is done through its associated **Node**. As most of the functionality between the different node types overlap, a **Node** is used as the base object and functions as a middleware node. Derived from this are the two other node types, treated as special cases of the middleware node, namely the **ServerNode** and the **ClientNode**. The **Node** hierarchy is shown in Fig. 3.5a.

A **Node** is always instantiated with either a connected TCP/IP **Socket** or an IP Address to connect to. Coupled with the connection, a **Node** holds a buffer for data that has been received as well as one for data to be sent. Finally, a **Node** is able capable of storing the login data that has been used by the middleware or client to log into the game server. This is done in the event that an autonomous relog to the server is required, such is the case during the handover of a supernode or a migration between groups. Every **Node** has an ID number which is derived from its IP address.

**Pairs** As the aim of the middleware is to distribute only *a portion* of the game mechanics amongst the P2P section, the remaining traffic will always flow along the traditional path between the client and the server. Therefore, a method to route non-distributed traffic both efficiently and correctly is implemented by pairing **Nodes** together. This method uses a set of **ConnectionPair** objects which ensure that any data received from one **Node** can be routed closer to its final destination efficiently. The three types of pairs are Client-Middleware, Client-Server, and Middleware-Server, and their software hierarchy is shown in Fig. 3.5b. The **ConnectionPair** object primarily houses two **Nodes** as well as logic to transform pair types in the event of network role transformations.

The type of pairings used depends on the role and position the nodes in the network. An example of the pairings are shown in Fig. 3.6. The supernode, having to maintain connections between the server and other nodes, uses Client-Server and Middleware-Server pairings. Regular peers need only maintain a paired connection between their own client and the supernode using a Client-Middleware pair.

**Socket Configuration** All network connectivity runs on asynchronous TCP/IP sockets. Using the asynchronous mode provides two major benefits for the system over the synchronous alternative. The first of which is the reduction in the number of threads required for maintaining the connections. In the worst case of synchronous sockets, each connection requires a thread of its own and thus a large peer group would result in a large number of threads on each client machine in the group.

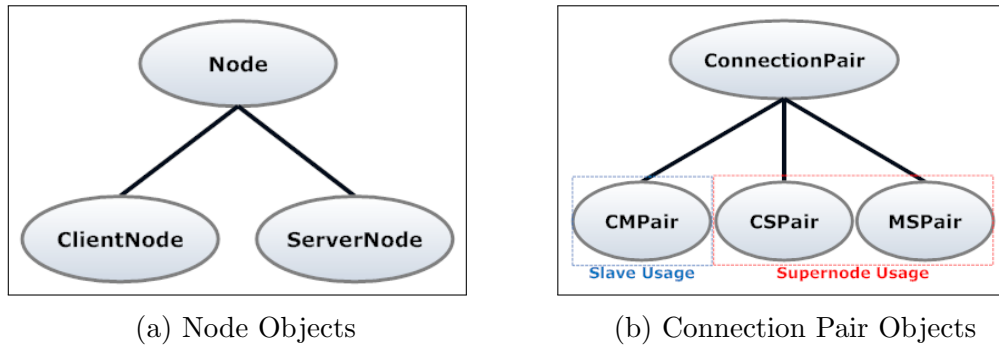


Figure 3.5: Basic networking module objects

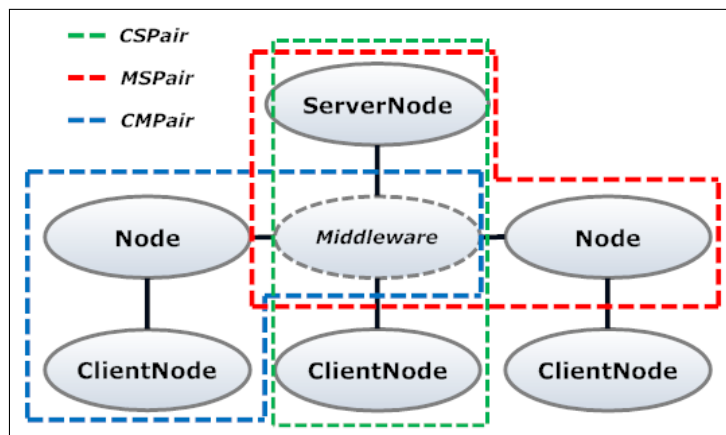


Figure 3.6: Networking node pairs where the centre middleware is the super-node

Asynchronous sockets remove the need for polling connections in any form and this leads to the second major benefit: any connection activity triggers a callback function which can be used as a feeder for the rest of the system. This is used extensively to populate the processing queues in the system.

**Packet Handling** Packets are managed at multiple stages throughout the middleware. Initially, all the packet structures are registered with the `Packets` class. Registering a packet involves the packet's unique identifier number and the length of the packet, being either a static or variable length. If a packet is one which requires specialised processing, then a `PacketHandler` must be created which will be called and executed if a packet of that type has been received and is being processed. This provides the game logic and IM with a method to register a handler and operate on received packets. It is a simple solution to allow any packet structure to be included



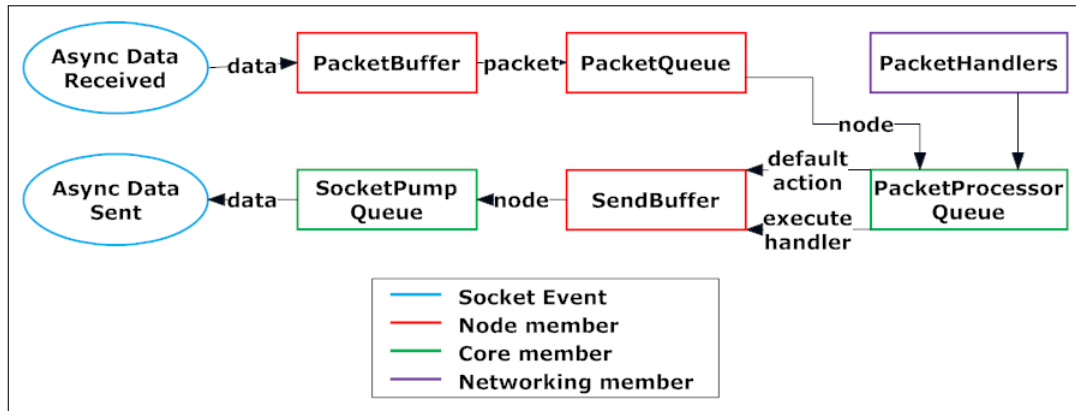


Figure 3.7: Queue Processing Flow Diagram

within the system, and to be processed in a custom fashion. The nature of Ultima Online's packet defines that packet identifiers are only unique to either the client or the server, but not when considering the full set of packets. Therefore, this required for each packet to be registered only to one of three particular set of packets, belonging to either the client set, server set, or custom middleware set. This, in turn, requires that each packet's source node's type be checked before retrieving the registered packet information.

The full packet handling process is summarised in Fig. 3.7 and described below.

When data is first received by an asynchronous socket belonging to one of the *Node* objects, the received data is placed within that *Node*'s receive byte array buffer. The buffer is immediately passed onto the *Node*'s *PacketBuffer* which is an object designed to package the received data into packets. If enough data has been received so that one or more complete packets are detected by the *PacketBuffer*, they are each packaged, placed in an output queue, and the *Node* from which they originated is placed into the *Core*'s packet processing queue.

When the *Core* processes the packet processing queue, the *PacketProcessor* class dequeues and processes each *Node* individually. Once dequeued, a *Node*'s *PacketBuffer* is retrieved, and each packet in its buffer is processed. When a packet is extracted from the *PacketBuffer*, the originating node type is checked, and the handler of the packet is requested from the registered packet set belonging to the node's type. If a handler exists for the packet, it is called and executed to process the packet. This involves either interfacing with the Game Logic module, calling the IM module, or working internally with the networking module. If no handler exists, this is indicative of a packet belonging to a game mechanic that is not distributed. Thus, the default action is triggered and the packet is routed along

to the paired `Node` in the originating node's `ConnectionPair`. This would typically involve routing a client's packet towards the server, or vice versa.

If the result of any packet processing involves sending data to another `Node`, or if any middleware event triggers data transfer, a generated packet is passed to the destination `Node` object through its send function. A `Node`'s send function adds the packet into its own outgoing buffer, and places the `Node` into the `Core`'s outgoing packets queue. When the `Core` processes the outgoing packet queue, the `SocketPump` class is responsible for dequeuing each `Node` individually and sending all data contained in its outgoing buffer via the `Node`'s socket.

**Peer Grouping** For use with interest management, the middleware is implemented with functionality to organise peers into smaller groups that are administered each by a designated supernode. Any peer is able to form a group by becoming a supernode on instruction from the directory server. Similarly, any supernode migrating out of an otherwise empty region can disband the group by informing the directory server.

Each peer maintains a `NeighbourList` structure for storing the `Node` objects of peers within their own area of interest. This allows peers within a shared AOI to communicate directly with each other. A peer is able to add to or remove from the `NeighbourList` accordingly. This can be performed by the peer itself, or under instruction from another peer, such as a supernode that has calculated an AOI.

In addition to this base functionality, the supernode requires the following extensions to facilitate and maintain the peer grouping:

- It accepts new connections, establishes connection pairs to the server, and maintains a list of peers within its group.
- As the IM scheme requires it to have a knowledge of other groups and their activity, the supernode establishes connections and maintains a list of neighbouring supernodes. It is able to communicate with them directly.
- Using the `PacketHandler` system, it is able to filter redundant or unnecessary packets that shouldn't be distributed amongst the peer group.
- In the event of a player quitting or a migration, the supernode is able to identify and transfer its supernode duties to another group member through the handover sequence, which is discussed below.

When a supernode migrates, it selects another peer within the group to assume the role of the supernode. A timeline of the handover process is shown in Fig. 3.8. The chosen peer is notified of its selection and the old supernode waits for a re-

sponse. When the chosen peer receives notification, it sets its own supernode flag and transmits an acknowledgement packet to the old supernode. The old supernode then sends an update to the directory server, informing of its departure and which peer will be assuming the new role. Following this, all the peers in the group are informed of the handover by the old supernode and each peer closes its connection to the old supernode. As previously mentioned, during the first login process, the login information is farmed by the middleware and stored within the `Node` objects. This allows the group members to autonomously begin a new login procedure with the new supernode, directly if a connection already exists, or after establishing a new connection. The new supernode, on receiving a login request from a group peer, establishes a connection to the server and the connecting peer continues the login emulation procedure. The communication then resumes as normal and the old supernode shuts itself down or migrates to a different region.

**Migrations** Interest management requires that peers are able to migrate between groups. As part of migrations, peers often have to undergo role transformations, as well as reassigning themselves to different supernodes. Each of the migration types previously mentioned in section 3.3 are managed differently, each resulting in different changes in the connectivity.

In most cases, a migration requires that the migrating peer severs its original connection to the server (via a supernode if applicable), and re-establishes a new connection with the server either through a different supernode, or directly to the server. Whenever a new server connection is established, a relog must occur and the full login process has to be followed so that the new connection is properly authenticated.

In the cases of an *any*  $\rightarrow$  *slave* migration, where a peer has entered a new region to be a slave, it must relog via the new region's supernode to the server. This results in the slave, server and new supernode each processing and communicating the relog traffic.

Furthermore, in the case of a *supernode*  $\rightarrow$  *any* migration, where a supernode departs the old region, it can leave behind a number of slaves. If this occurs, a supernode handover must occur, where all the remaining slaves are required to relog via the new supernode to the server. Therefore, the server and new supernode must both process and communicate multiple instances of relog traffic originating from each slave.

In the case of the *slave*  $\rightarrow$  *supernode* migration, the slave departing the old region terminates its connection to the old supernode, undergoes a transformation

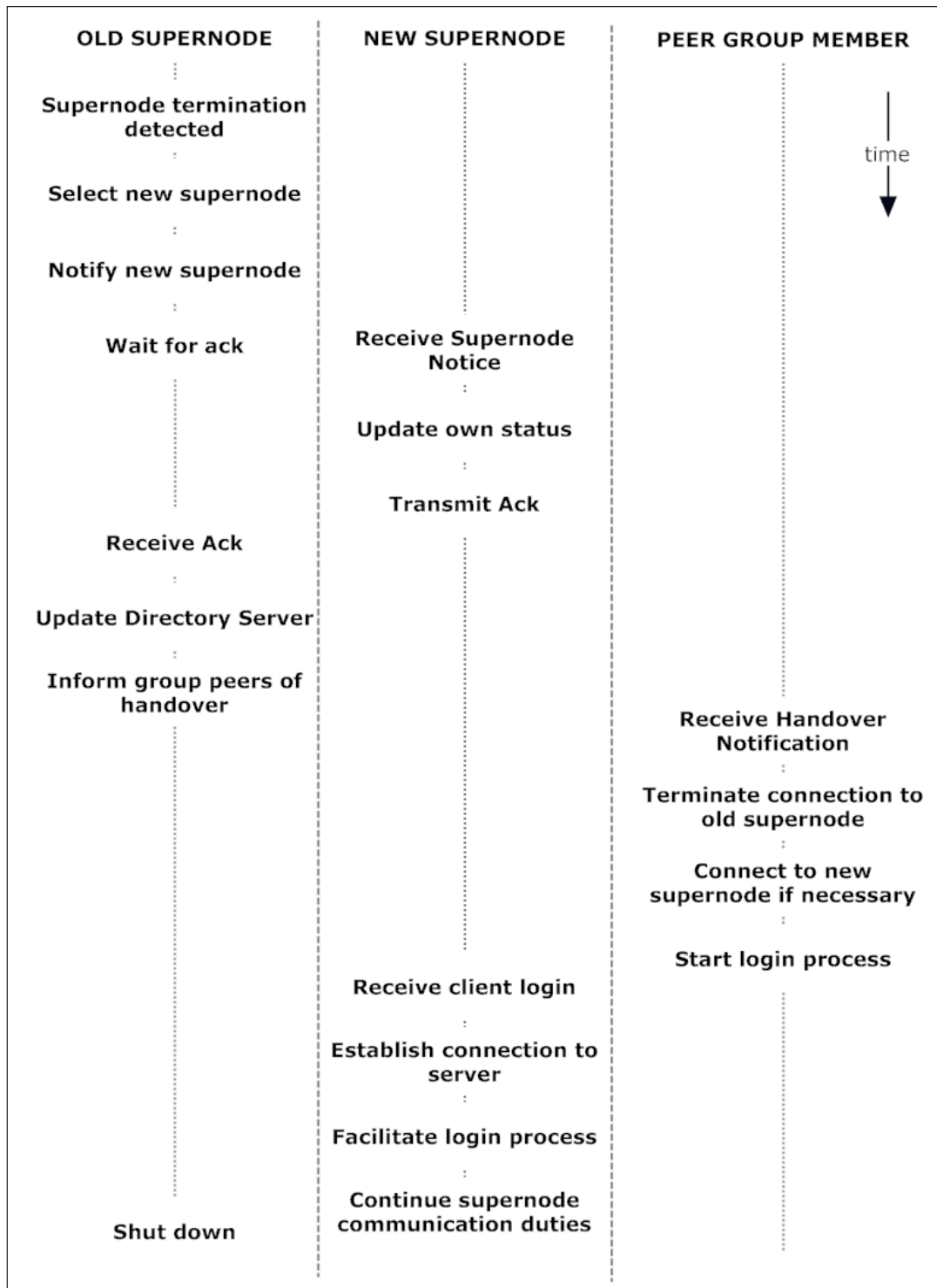


Figure 3.8: Supernode Handover Timeline

and performs a relog directly to the server.

The only migration which does not incur a relog to the migrating peer specifically is the *supernode* → *supernode* migration, where the supernode simply maintains its connection with the server through the migration.

**Listening** As the middleware accepts connections from a variety of different nodes, a **Listener** object is instantiated as part of the networking module's startup. The **Listener** maintains an asynchronous socket that listens for connections on a specified port. When a new connection is established with the socket, the **Listener** accepts the connection and places the socket into the Core's new connection queue. When the Core triggers the new connection queue for processing, each socket is dequeued and begins the process of establishing a **Node** object to cater for it. The method of processing a new connection is summarised in Fig. 3.9 and described below:

Initially, the first byte is checked to see the type of connection that has been established. Typically the first connection will arrive from a game client. The middleware proceeds to connect to the server and log into the game. The moment the middleware is aware of its current ingame location, it uses the IM module to query the directory server for the relevant supernode to connect to. Once this occurs, one of two outcomes are possible:

1. If it receives a supernode address, the middleware then connects to the supernode and generates a **ConnectionPair** containing a **ClientNode** and a middleware **Node**.
2. If the group is empty and no supernode address is provided, the middleware itself assumes the role of a supernode and establishes a connection directly to the server, in turn generating a **ConnectionPair** containing a **ClientNode** and a **ServerNode**.

If the connection does not originate from a game client, but from another peer, then three cases exist:

1. If the listener is *not* a supernode, the connection can only originate from a new peer in its AOI, and thus the peer is added to the listener's **NeighbourList**.
2. If the listener is a supernode, the connection can originate from a neighbouring supernode. The new **Node** is then passed to the IM module for processing.
3. If the listener is a supernode, the connection can originate from a new peer joining the group. In this case, a **ConnectionPair** containing a middleware **Node** and a **ServerNode** is established. Once aware of the new peer's in-game

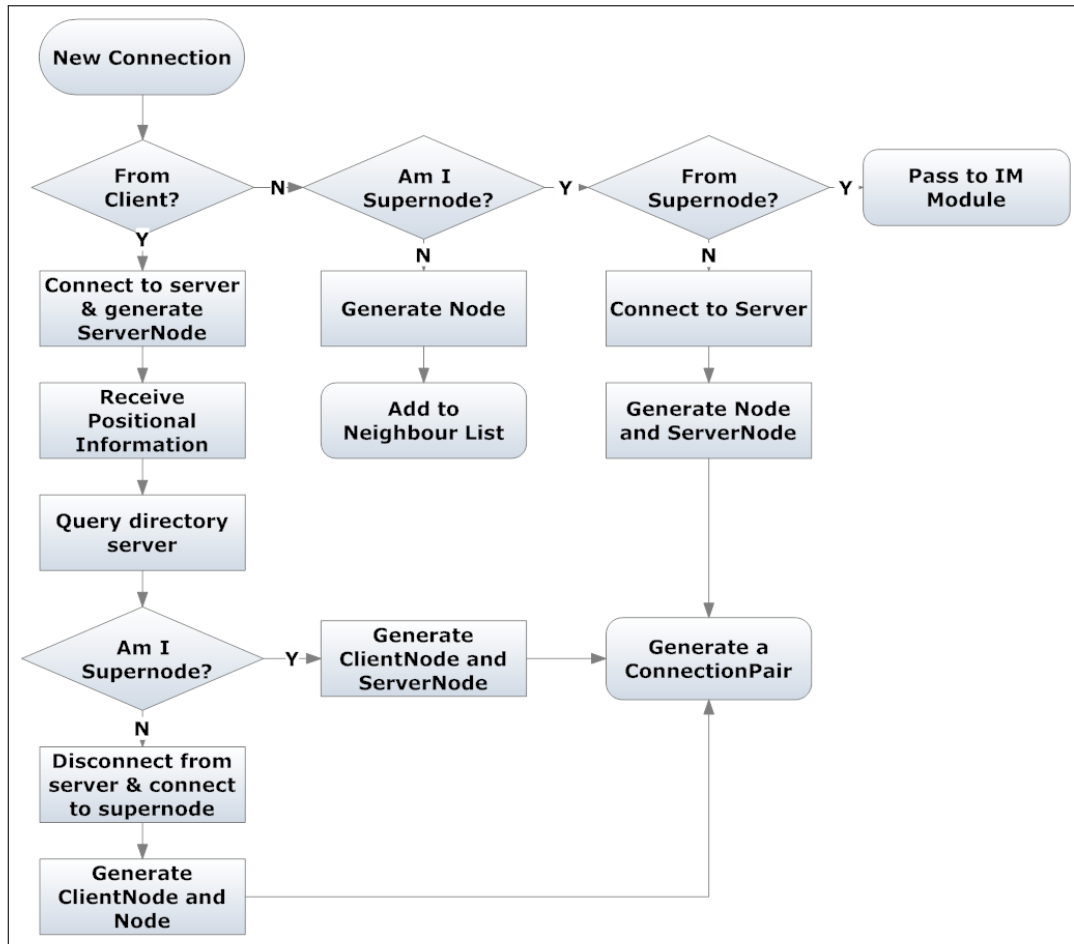


Figure 3.9: Process on the middleware's receipt of a new connection.

position, the Node is passed to the IM module for processing as a slave.

### 3.5.4 Interest Management

#### Interest Management Actions

A peer participating in the interest management system needs a set of actions at its disposal for navigating and operating within the virtual world. Therefore, the design of the interest management module should provide all peers with the functionality to perform the actions detailed below.

- **Initialisation** The interest management system must be initialised by dividing the virtual world into static regions that are consistent between all peers. Following this, it must initialise its own location within the resulting grid of regions and facilitate its entry into the peer-to-peer network with the

directory.

- **Query/Update Directory** Querying the directory for locating the supernode of the current region, and locating supernodes of surrounding regions if necessary. Updating the directory of supernode changes in a region.
- **Join/Leave/Migrate Region** Joining a region by connecting to a supernode and awaiting instruction. Leaving a region by gracefully disconnecting from the supernode. Migrating from one region to another by chaining together leave and join actions.
- **Add/Remove from AOI** Maintaining a peer's own area of interest by adding and connecting to other peers, or removing and disconnecting them.
- **Broadcast Updates to AOI** Broadcasting necessary information pertaining to the distributed game mechanics amongst the peer-to-peer system.
- **Transform** Becoming a supernode in a region where a supernode has left, or when joining a region that is unpopulated.

In addition to these base actions, supernodes need additional actions at their disposal to manage the interest management scheme.

- **Compute** Computing the current state of each slave's area of interest.
- **Update Slave's AOI** Controlling the slaves' areas of interest by sending state updates instructing additions or removals of other peers.
- **Handover** Extends the leaving and migration actions by handing over the current supernode duties to a new supernode in the region being exited.

## Aims

The interest management software module aims to implement the MOPAR-like scheme that is described in section 3.3. This is to introduce functionality to efficiently limit and control connectivity only between specific peers, in contrast to a setup where every peer is connected to every other peer.

Each peer group should be assigned to a region, therefore the entirety of the virtual world should be subdivided into smaller hexagonal regions. Each region must have a single supernode which will manage all other peers located within the same region. The regions must be uniquely identifiable and consistent between all peers in the system.

The directory has two primary roles: it acts as an entry gateway into the peer-to-peer system and it stores the supernode information of each region. When a new

peer connects to the system, it queries the directory with its location to find the supernode with which it should connect. If the region is empty, the directory must instruct the new peer to assume the role of supernode, and to connect and pair with all other supernodes located in neighbouring regions. The directory must always be updated with changing supernode regional information so as to provide accurate instruction to querying peers.

With the introduction of interest management, every peer must establish and maintain its own AOI so that a peer's distributed updates may be transmitted only to interested peers. This includes making use of existing functionality to initiate and terminate connections. A slave should accept instructions from a supernode on how to update its AOI, and apply the changes accordingly. A supernode must simply apply any updates to its own AOI without further instruction.

As the supernode acts as the controller for each of its slave's areas of interest, it must have a wider knowledge of the surrounding regions. It should always store the locations of each peer located within its own region, as well as neighbouring regions. The slave locations should be obtained directly from them, and the neighbouring regions information must be obtained from neighbouring supernodes located through the directory server. By storing this information, the supernode may then compute the areas of interest for itself as well as each of its slaves. Once computed, the new state of the slave areas of interest should be stored locally, and the area of interest state updates sent to each respective peer.

## Design

**IM functionality** A lot of the underlying functionality required for Interest Management was developed in the networking module for facilitating peer grouping. Peers are able to maintain their AOIs through the `NeighbourList` and are able to communicate with the peers contained in the list; peers are able to join, leave and migrate between groups; and role transformations are handled by supernode handovers. Therefore, it was only required that the IM module expand on these features, register specific packets and implement their associated handlers to control the IM tasks.

**Regional Division** As each peer group is assigned to a virtual region, the existing virtual world must be divided. Conceptually, the two horizontal dimensions of the 3D virtual environment are overlaid with a virtual 2D grid of hexagons to divide the world into regions. Although it is understood as an overlaid grid, the



only information that is actually required for further interest management action is the mapping of a 3D point in the virtual world onto a 2D coordinate reflecting which hexagon a player is located.

The hexagonal grid can be computed either statically at startup, or dynamically throughout the use of the system. In the case where the grid is stored statically, every point in the virtual world is iterated through and is computed individually with the result stored in a large 2D array, referencing the point to an appropriate hexagon identifier. This introduces a large computational activity at startup, and requires storage in memory throughout the execution of the software. In the case of Ultima Online, this translates to around 56MB<sup>1</sup> per map, with there being up to 7 maps in use. The benefit of static storage allows very fast lookups of any coordinate into the hexagonal system. However, given that its purpose is to have a minimal impact on system performance while operating in the background, the increased memory requirement of the software is not ideal. Thus, a dynamic lookup strategy has been selected over a static system. By distributing the initial computation between each coordinate lookup, it offers a massive reduction in memory requirements, with a form of on-demand computational cost. This is further justified as no peer will ever require the full information of every point in the world.

Therefore, a `HexGrid` object has been created to perform this task, acting as the 2D grid. It is initialised by specifying the length of a single hexagon side. From the side length, the other hexagonal components are calculated by creating a `Hexagon` object which stores all hexagonal properties within. Using the hexagonal properties, a virtual grid is emulated by the `HexGrid` and can map a 3D point onto a hexagon in the 2D grid, an example of which is shown in Fig. 3.10. To compliment this, a `HexPoint` is used to store a hexagonal coordinate, along with a unique identifier for the particular hexagon. The algorithm of the hexagonal coordinate lookup is detailed in Appendix A.

**Directory** The directory server acts as the entry point into the peer-to-peer network. Therefore, its foremost function is to accept new connections. Once the connection is established, it receives a query or update from a peer, processes it, responds appropriately and then terminates the connection. Both queries and updates requires that the directory store a dictionary of supernodes alongside their associated hexagonal coordinates. The possible query and update communications are listed in table 3.1.

---

<sup>1</sup>Calculated with the vanilla Ultima Online 7168x4096 point map at 2 bytes per hexagonal identifier

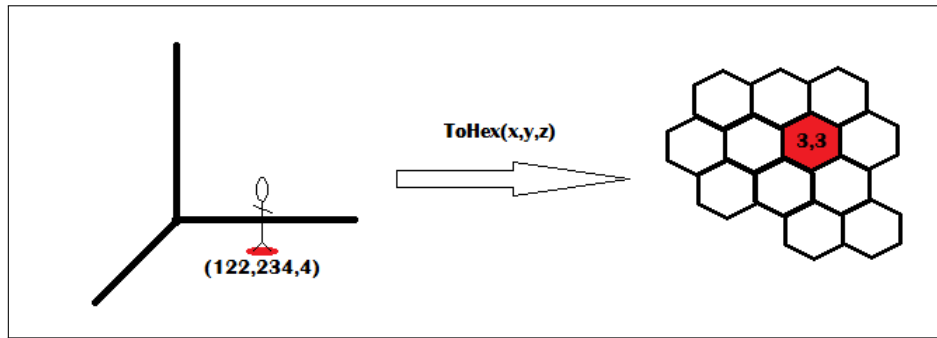


Figure 3.10: Hexagonal Mapping

Command	Arguments	Description
Region Query	Hex ID	Queries for the supernode in a region. Responds with an address, or a null to reflect an empty region.
Handover Notification	Hex ID; New IP info	Updates the directory to point to the new supernode in a region.
Region Termination	Hex ID	Updates the directory to reflect a region as empty.
Neighbour Query	Hex ID	Queries for a list of neighbouring supernodes of surrounding regions.

Table 3.1: Directory Communications

The remaining IM module components presented in this section are for use exclusively by supernodes.

**Interest Units** Every peer that enters into the interest management system must be registered with a supernode and have its details recorded. Necessary information needs to be stored to facilitate two tasks: (1) computation of areas of interest, and (2) providing addressing data for connections to be established between slaves. In addition to these requirements, a supernode must maintain the state of all its slaves' AOIs so that it may correctly apply updates and prevent excessive redundancy. From these requirements, the concept of an `InterestUnit` was introduced into the system for use by supernodes. This object represented the most basic entity of a peer located in the system, simply holding the locational information of the peer,

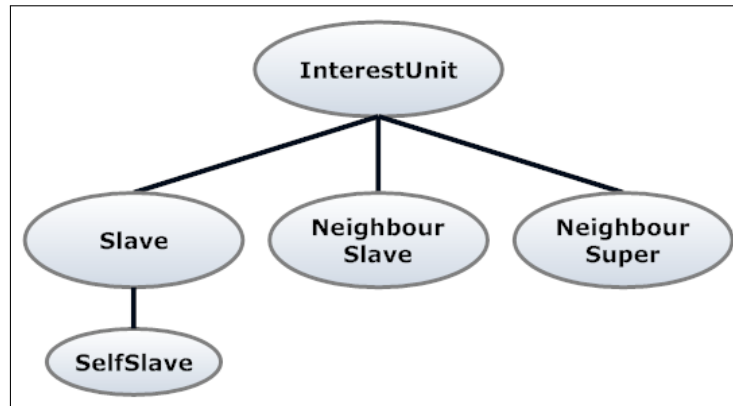


Figure 3.11: Interest Units Hierarchy

its IP address information as well as its associated serial numbers. From this base unit, four extensions were introduced to cater for the four types of units that could exist in the system from the supernode perspective. The inheritance structure is shown in Fig. 3.11.

A **Slave** extends the **InterestUnit** by integrating with the Networking module and storing its **Node** object, which is described in section 3.5.3. This allows a supernode to communicate directly with any slave it controls while remaining within the bounds of the interest management scheme. More importantly, the **Slave** object introduces storage of the AOI state for a supernode's own reference. With this, a supernode may amend changes to the current state of any slave's area of interest, while simultaneously generating and transmitting a state update packet to the actual slave.

For the supernode to control its own AOI within the system, a **SelfSlave** allows the supernode to treat itself as a slave from the computation perspective, while interacting directly with its own AOI rather than updating itself through a packet. This interest unit overrides most of the **Slave** functions.

The supernode needs to be in direct contact with neighbouring supernodes, as well as provide them with regular updates of the positional data of slaves within its own hexagonal region. This is handled by a **NeighbourSuper** object which, like the **Slave**, references directly to a **Node** object, allowing the supernode to communicate directly with neighbouring supernodes from within the interest management module.

Finally, a **NeighbourSlave** mostly resembles an **InterestUnit** by storing the location and IP addressing information of all slaves located in neighbouring regions. The only addition to the base object is that each neighbouring slave references the **NeighbourSuper** it belongs to.

The inclusion of these units provide entities for the Management Logic to operate on.

**Management Logic** The management logic utilises the aforementioned building blocks to fully implement the interest management scheme. Separate storage lists are used for each type of `InterestUnit` so that each type may be processed separately, with the exception of a `SelfSlave` which is treated from the processing perspective as if it were a `Slave`.

As previously stated, each `Slave` object is able to store the state of the slave's AOI. The management logic introduces a *delta list* into the `Slave` object. Each slaves' *delta list* is determined by calculating if the slave is within the AOI of any other interest units known by the supernode. When an interest unit is found to be within a slave's AOI, the `DeltaAdd(InterestUnit)` function is called. This checks if the interest unit has already been added to the AOI and if not, it is added to the *delta list*. Similarly, the `DeltaRemove(InterestUnit)` operates as a removal function. Finally, once all adds and removes have been performed, the *delta list* is merged with the current AOI state list, and the *delta list* is transmitted to the associated slave as an AOI update packet.

The update packets instruct the slaves to either establish new connections to additional peers, or to terminate connections with peers that are no longer within range. When a slave receives an area of interest update from its supernode, it applies it to its own `NeighbourList`. In addition, when slave *A* adds slave *B* to its `NeighbourList`, *B* also adds *A*. This introduces redundancy into the system to ensure that areas of interest are consistent between relevant slaves.

With a slave's area of interest established and maintained by its supernode, or the supernode having established its own AOI, it is able to update peers that are nearby in the virtual world. The game logic module is able to transmit any distributed mechanic updates directly to all peers contained within its `NeighbourList`.

The supernode uses the directory to connect itself with neighbouring supernodes, from which it periodically receives state information about all peers in neighbouring regions. Combining the knowledge of all surrounding peers allows a supernode to accurately compute the area of interest for each slave within its own region. It is therefore able to instruct its own slaves to connect with neighbouring slaves directly.

Finally, a slave always stores its supernode in its `NeighbourList`. This allows a supernode's game logic module to always receive timely updates of its slaves distributed actions and maintain an up-to-date state.

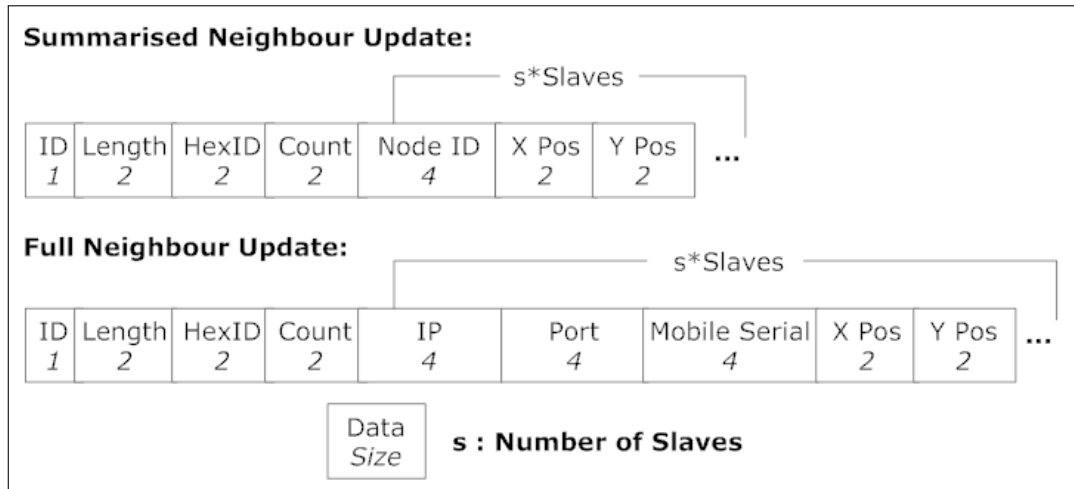


Figure 3.12: Neighbour Updating Packet Configuration

**Neighbour Updates** As a region's supernode requires information about player mobiles in neighbouring regions to manage AOIs that cross borders, it is necessary for supernodes of neighbouring regions to exchange information pertaining to their slaves. In addition to region *join* and *leave* notification packets, two types of periodic neighbour updates are used in this system: a summarised update and a full update. The construction of the periodic neighbour update packets are shown in Fig. 3.12.

Whenever a new slave joins a region, 23 bytes are transmitted by the supernode of that region to each of its neighbouring supernodes. This packet resembles the full periodic update packet seen in Fig. 3.12 where  $s = 1$ . It contains enough information for a neighbouring supernode to establish a new personal record of the joining slave so that the neighbouring supernode (1) is able to compute AOIs, and (2) is able to instruct its slaves to connect directly to neighbouring peers by using the stored network addresses. Similarly, a peer leaving a region incurs 7 bytes to the supernode for each of its neighbouring supernodes to inform them of the event.

Every 5 seconds, the supernode sends a summarised update of all slaves in its region, seen in Fig 3.12, to each supernode in immediate neighbouring regions. The summarised update totals  $n(7 + 8s)$  bytes transmitted by the supernode to update its neighbours, where  $n$  is the number of current neighbours, and  $s$  is the number of slaves currently owned by the supernode, including itself. This 5-second summary update contains limited information for a supernode to update the positional information of player mobiles it is aware of in its own records.

Every 30 seconds, a full update, seen in Fig 3.12, is sent to each supernode in immediate neighbouring regions. The full update totals  $n(7 + 16s)$  bytes transmit-

ted by the supernode and contains a full detailed description of the slaves in the region. The full neighbour update serves as an authoritative copy to overwrite any information that might have been lost or mistaken.

These neighbour update packets allow each supernode to have a wider view of the virtual area, and provide enough information to compute slave AOIs accurately.

### 3.5.5 Game Logic

#### Aims

The focus of the game logic is to allow portions of game mechanics to be distributed amongst the peer section of the hybrid network. This translates to a peer being able to receive an event from a client, process it, and produce a game state update that can be returned to the client and applied to its own state. Not only should the state update be applicable to the event-originating client, but to other clients that are part of the network, so that each player experiences the world in a consistent fashion.

As the middleware is required to process game events, it should have access to the mechanics and functionality that the server houses. By granting access to these functions, the middleware will be able to react in the same way that the server would if it were to receive the same game event. However, the server logic should not be directly inserted into the middleware, but rather abstracted through an interface to prevent any module from having direct access to and interacting with the full server functionality. In addition to this, abstraction will allow a separation between game-dependant and game-independent logic. This should allow for the specific game server logic and interface to be plugged out and interchanged when working with other games. The game logic module should therefore initialise the necessary server game logic on startup, and be able to access it when necessary to process events into updates.

Furthermore, the supernode is required to periodically update the server of its slaves' activities. Therefore, the game logic module should be able to produce state updates that can be sent to the server. These state updates should contain the aggregated data of all middleware-produced updates that have been recorded in the period of time since the last update. In addition, the game logic module should provide functionality for a supernode to identify which game packets require filtering out of the system.

## Design

The Game Logic Module is split into three different parts: the server game logic, the local game logic and the logic interface.

In the implementation of this system, the server game logic is imported directly from the server source and hosted inside the application. This allows parts of it to be modified to suit only the specific uses within the middleware rather than to initialise a full instance of the game server on a peer. Therefore, the server code was modified to introduce a simplified initialisation procedure so that only the modules relating to desired mechanics, in this case *movement*, are initialised on startup. With the server logic initialised, the middleware is able to locally store replicas of the game state pertaining to the location and status of all player mobiles involved in the system. In addition, the server logic provides the functionality to process events the same as the server would, directly on the local game state. The authoritative state of a specific player's mobile is considered to be the version hosted on the middleware belonging to that player. Each player mobile is referenced by a serial number that matches its reference on the server. Since each player mobile must be owned by a peer, it is also assigned with its owners `Node` ID number. This allows the host of the authoritative state to be recorded, and offers a fast lookup of mobiles according to nodes, and vice versa.

The local game logic and logic interface are closely interlinked and make extensive use of each other. The local game logic sets up all packets with the networking interface that require processing, creating and registering packet handlers for each that interact with the required functions from the logic interface. If the received packet is found to be from one of the distributed game mechanics and is an *event* packet, the handler will call the logic interface which will interact with the game server logic and produce an update that can be distributed to the `NeighbourList`. Similarly, if the received packet is an *update* packet, the handler will call the logic interface which checks whether the update originates from the owner of the authoritative state, and proceeds to update its local game state stored in the server logic accordingly.

The logic interface has a standard list of functions, shown in Table 3.2. These functions allow the game logic packet handlers to process the received data, as well as to gain access to stored game state information. For example, if the middleware was to receive a *Movement Request* packet from its client, the packet handler would be called, which in turn would use the interface to call `MoveMobile(serial)`. The function would return a response validating the requested movement. If it was a

Function	Description
InitData()	Calls the initialisation of the server logic.
NewMob(Serial, Node)	Creates a new mobile to be stored in the local game state.
Find(Serial)	Retrieves a mobile from the stored game state matching the same Serial.
Dispose(Node)	Disposes of the mobile associated with Node.
UpdateMobile(Serial, Args)	Updates parameters of a locally stored Find(Serial).
QueryMobile(Serial, Args)	Queries a parameter of a locally stored Find(Serial).
MoveMobile(Serial, Direction)	Processes a movement event of Find(Serial) in the Direction.
SendPostionUpdate(Node)	Sends an aggregated location update to the server of the specified Node's mobile.
LastLocation(Serial)	Queries the last location of Find(Serial)'s update to the server.
LastUpdate(Serial, Location)	Checks if a specified location is the same as the last update of Find(Serial) sent to server.
BroadcastUpdate(Serial)	Broadcasts a state update of Find(Serial)'s location to all neighbour peers.
GenerateMobList()	Generates a list of locally stored mobiles and their associated nodes.

Table 3.2: Logic Interface Functions

legal move, the local game logic would create its own *Movement Acknowledge* packet and return it to the client. Following this, by calling `BroadcastUpdate(serial)`, it would retrieve the new location, construct an update packet, and send it to all peers within the `NeighbourList`.

Beyond the base functionality, when a supernode receives an movement update that is applied to its replica of the game state, it also informs the IM module so that it updates its slave records of the region to perform accurate AOI calculations.



## Periodic Updates

When the server processes many of the non-distributed game mechanics, it takes a player's position into consideration. Therefore, the supernodes are required to provide the server with periodic updates of their slaves authoritative states to maintain continuity and a degree of consistency in the gameplay.

A supernode maintains an update list, and each time the supernode receives an update from a slave in its group, it adds that peer to the update list. Periodically, the supernode calls the `SendPositionUpdate(Node)` function from the logic interface which updates the server of each peer contained within update list. A 3 second interval between updates was chosen to still provide an accurate state of the virtual world to the UO server, while maximizing on the potential bandwidth reduction.

With the updates being transmitted by the supernode, the server is required to accept the information and apply any changes to the server's game state. However, after assessing how to create and transmit the periodic movement updates, it was found that the server would not accept any form of direct position updates without administration privileges. Further investigation was performed into player interactions in the hope that an event packet could be manipulated as a workaround to this problem. An example would be using the game's teleport spell packet to provide changes in a mobile's position. However, the server was secured against such exploits by requiring the player mobile to have sufficient abilities or consumable resources to perform them. With no manipulation possible, it required that the server's scripting interface be used to introduce a new, hidden command that allows a player to send a positional update of their own player mobile. Therefore, the periodic update packet is constructed within the middleware containing the necessary information and disguised as a plain text packet. While cheating mitigation and security is outside the scope of this experiment, encryption could be applied to the periodic updates in a full system to enhance their security.

## 3.6 Summary

In this chapter, the design and implementation of the middleware plugin and hybrid system was presented at various levels. It began with a high-level overview of the full hybrid system, detailing some of the major design choices that define the structure of the system to achieve the two primary goals: (1) to reduce server load and (2) to reduce player latency. This included the use of direct communication; peer grouping according to regions; the use of supernodes; how the topology is structured

to overcome the UO server limitations; and the choice of movement for distribution amongst the peers.

Following the overview, the conceptual design of the interest management scheme was presented. The scheme, based on MOPAR, divides the world into hexagonal regions, where each region forms a peer group and is administrated by a supernode. Slaves connect to the supernodes via a directory server hosted alongside the server. The supernodes is tasked with calculating the AOIs of its slaves and instructing them on who to be connected with. Finally, the 4 possible migration types were listed.

With the basic structure of the system and IM scheme covered, the conceptual design of the network architecture was discussed. Each physical system runs a middleware plugin and a game client, and the two are connected internally. The directory server acts as an entry point into the P2P segment of the network. When a player joins the game and queries the directory server, it either becomes a supernode or reroutes its connection to the server via a supernode it is assigned to. The direct communication principles allow peers to minimize latency by eliminating any additional hops.

After the high-level design of the system was covered, finer details of the implementation were presented. The implementation is divided into 6 sub-modules: core, networking, interest management, local game logic, logic interface, and server game logic. Each of these modules were presented according to their aims and design. While the majority of the functionality is shared regardless of the peer's role as supernode or slave, in some cases extended functionality was introduced to cater for a specific role. In addition, much of the functionality in some modules is dependant on other modules capabilities. Together, the modules made up the implemented middleware plugin that is investigated in the following chapter.

# Chapter 4

## Experimental Investigation

### 4.1 Introduction

This section fully details the experiments performed on the hybrid system to evaluate its performance when compared to the baseline client/server system. As part of the introduction, the metrics by which the system is measured are described, along with the method of automating the game and the motivation behind the duration of the experiments. The remains of the chapter proceed through each metric, providing details of every performed test: the motivation and configuration of them, and a discussion of the results.

#### 4.1.1 Performance Metrics

It is expected that the hybrid system will have an impact on different attributes of the original C/S system. The exact effects need to be measured through performance experiments, and thus three metrics are important to the performance of the system.

**Data Bandwidth** is defined as a measurement of the bit-rate of consumed data communication resources. In this system, bandwidth is defined as the amount of data sent and received by a node in the network summed over a duration of time. A focus is placed on two measurements of bandwidth: the total recorded bandwidth over an extended duration, and the per second bandwidth.

Bandwidth has a direct financial implication on the owner of a node. If it requires additional bandwidth then it must purchase an upgraded capability from its service provider at a higher fee. Therefore the *lower* the bandwidth experienced by a node is, the better for the owner of it.

**Latency** is defined as a measure of time delay in a system. In the context of communication networks, latency typically refers to the time delay from when a packet is sent to when it is received. In this system, latency refers to the time delay between when a packet is sent from one node, to when it is received by the node it was destined for. The constraints of a practical system means time measurements across different nodes cannot be made with a resolution less than 1ms. Therefore, it is necessary to measure latency in the form of a *Round Trip Time* (RTT). RTT is defined by the time delay between when a node sends a packet and when the same node receives a response to that packet.

Latency has a direct impact on players, and an indirect impact on the game host. Players demand low latencies to be able to play the game to its full potential. If latencies are high, the delay interrupts the gameplay, causes a disadvantage for a player and will ultimately result in their dissatisfaction. If players are dissatisfied, they leave the game resulting in lost revenue. Therefore, *low* latencies are better to facilitate the gameplay and maintain a strong player base.

**Hardware Impact** is not formally defined, but is a more subjective measure of how the software influences the hardware usage of a computer. This metric is primarily focused on the processor and memory usage of the software. Each of the hardware metrics are derived from the *Windows Diagnostics* tools built into the development platform [52]. The processor usage is recorded as the percentage of time that the processor has spent executing a non-idle thread of the process over the sample duration. Informally, it shows how much the application process is making use of the processor. Memory usage uses two measurements: the private bytes and the working set. The private bytes is a measurement of how much memory has been reserved by the process with the operating system, whereas the working set is a measurement of how much physical memory is in use by the process.

If the software is more taxing on the computer system, better hardware is required to run it. Improved hardware comes at a higher initial cost, and often has a higher running cost than low performance hardware.

The overall system performance will be evaluated according to these metrics. Bandwidth and Hardware Impact have a greater influence on the game host but also affect players of the game. Latency, in contrast, has a greater affect on the players.

Action	Description	Probability
Large Movement	Move to a random location 15 units away in x and y directions.	0.1
Small Movement	Move to a random adjacent tile.	0.6
Speak	Say something	0.3

Table 4.1: Automated Player Behaviour

### 4.1.2 Game Automation

For the real implementation of the system to be evaluated, the game client would need to perform as if it were being controlled by a real player. This involves invoking player actions on the game client at different intervals throughout the duration of play. Accurately modelling player behaviour is a complex activity, with activity being dependant on the game itself, the time of day, the length of play and the level of the character, amongst other influences [53].

To gauge the influence of this system, a simplified model of player behaviour was created to cater for the bandwidth and latency measurements, according to the expected traffic generation. It is expected that the accuracy of the player behaviour will not have a significant impact on the outcome of the experiments if the same behaviour is applied equally throughout each test across all architectures. Only two game activities were included in the player behaviour: movement and speech. Modelling movement and speech allows for respectively measuring the effect of distributed and non-distributed mechanics.

A script was written for the *EasyUO* macro application. Once the script is launched, the automated player selects and performs an action from table 4.1. When the selected action is complete, it waits a short duration and repeats the process. This continues until the user intervenes and shuts down the script. The actions were modelled to suit the style of Ultima Online, where a player will go to an area, make various small movements around it while performing actions (such as speech), and then make a large movement to move on to another area — a typical style of gameplay associated with Ultima Online.

### 4.1.3 Testing Duration

The duration of any tests that aim to emulate standard player behaviour should span around an average length that a player would play an MMOG in one sitting. The average session of a World of Warcraft player spans the duration of 2.3 hours with a median of 1.4 hours [54]. Using these as a guideline, running each test for 2

hours should suffice as an estimation for a single session of playtime.

## 4.2 Bandwidth Evaluation

### 4.2.1 Overview

The purpose of the bandwidth tests is to measure the difference in bandwidth experienced by nodes between the original Client/Server system and the new hybrid system. A baseline for comparison is established by measuring the C/S system. The following two tests are performed on the hybrid system: (1) the single region test, and (2) the inter-region test.

The single region test defines a configuration where no inter-region interest management effects take place. Each region is steady in terms of the number of peers it has, and no migrations or supernode handovers take place. The only minor effect of interest management on the single region is the definition and control of AOIs. Therefore, the single region test focuses on consistent game traffic as a result of player activity.

The inter-region test investigates traffic resulting from inter-region interest management effects. It focuses on fluctuating bandwidth as a result of migrations and IM overhead necessary to maintain the system. Players move freely between regions, causing migrations, supernode handovers and fluctuating region populations.

In this section, the test platform is described and the three practical tests are discussed with respect to their motivation, configuration and results.

### 4.2.2 Test Platform

The amount of bandwidth generated by nodes in the system is independent of the network latency, this led to the use of a local area network (LAN), which operates at 100Mbps, as a platform for testing. To limit the number of physical computers required, *VirtualBox*, a software package for running virtual x86 hardware, was utilized [55]. VirtualBox allows the creation and execution of multiple virtual machines on a single physical machine. Each virtual machine functions as if it were an independent system, with its own operating system, IP address and virtual hardware with a single core processor and 512MB of memory. Using VirtualBox, each physical computer is capable of acting as multiple machines on the network, each with their own game client and middleware running.

### 4.2.3 Test 1.1: Baseline Bandwidth

#### Motivation

For the effects of the hybrid system to be properly measured, they must be compared against how the standard C/S system baseline performs. The bandwidth for both the server and the client nodes are recorded. The movement and speech packets, as well as the total bandwidth, are measured separately to facilitate comparison with the hybrid system (where the movement packets are distributed, and the speech packets are sent to the server).

#### Configuration

This test uses the following setup:

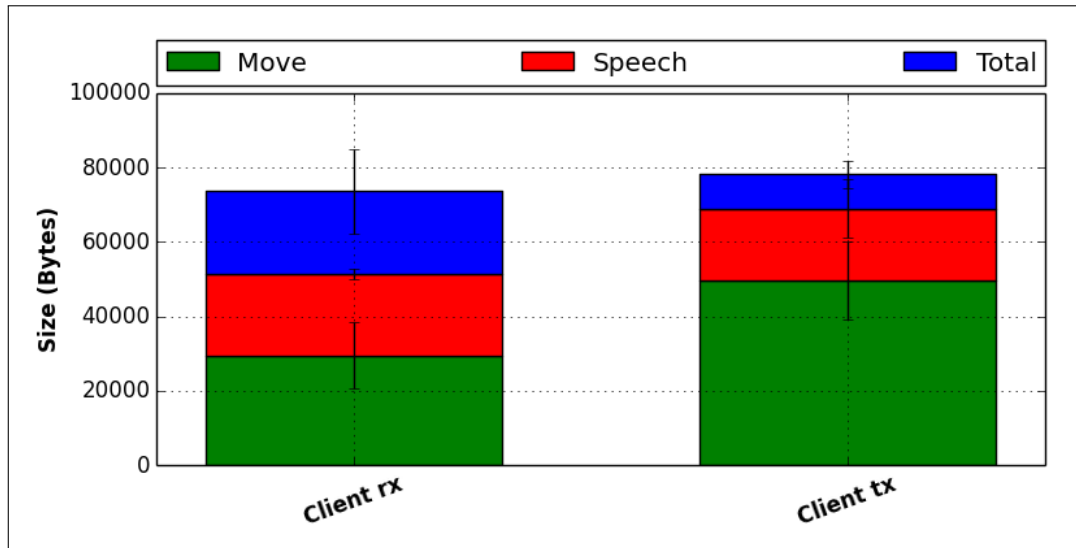
- The server is hosted on a dedicated computer.
- Fifteen clients are hosted on virtual machines spread over three physical computers.

Since the client and server software has not been modified, simplified middleware is used to record the bandwidth measurements. The client connects to the server via the middleware, and every packet is transparently passed through the middleware, without applying any game logic of modifying the packets, whilst recording the size of the packets.

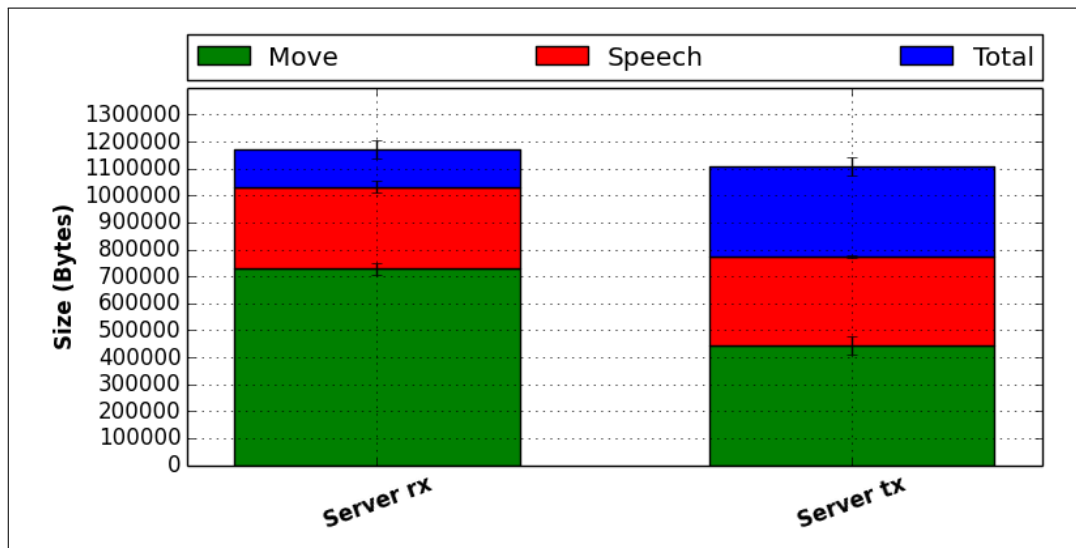
Each client executes the macro script for a period of 2 hours. All player mobiles are limited to a shared virtual area to ensure occasional interaction.

#### Results & Discussion

The recorded bandwidth values of the clients and the server were processed to produce means and standard deviations. The average amount of bandwidth experienced by a client in the C/S system is shown in Fig. 4.1a. In this, the green segments refer to the portion of traffic resulting from movement mechanics, showing the updates received and events transmitted respectively. It can be noted that the transmitted movement traffic is greater than the received traffic, suggesting that the movement events are larger than their associated confirmation updates returned from the server. Similarly, the red segments refer to the portion of traffic resulting from speech: updates received and events transmitted respectively. The remaining blue segment represents the miscellaneous game traffic, and thus makes up total traffic experienced by that node. The error bars refer to the standard deviations of each the movement, speech, and total.



(a) Client Bandwidth Totals



(b) Server Bandwidth Totals

Figure 4.1: C/S baseline results

Similarly, the bandwidth experienced by the server is shown in Fig. 4.1b. However, in this figure, the received traffic refers to events which it received from clients, and the transmitted traffic refers to the updates transmitted from the server in response to its clients. The tabulated results are available in Appendix B.

These results serve as a baseline for comparison with the hybrid system.



## 4.2.4 Test 1.2: Single Region Bandwidth

### Motivation

This test aims to capture the single region bandwidth experienced by nodes in the hybrid system. It is performed in a static, single virtual region with a single supernode for the full duration of the test. This static setup is used so that the bandwidth differences can be observed for distributed and non-distributed game mechanics, without any influence from the bandwidth as a result of peer migration, supernode handovers, or relogs.

This aims to measure the additional bandwidth per slave as a result of assuming the role of a supernode; the additional bandwidth experienced by all peers in the P2P segment; and the change in bandwidth experienced by the server.

### Configuration

Fifteen peers, each consisting of the game client, middleware and macro application, and a server are used in this test in the following setup:

- The server is hosted on a dedicated computer.
- Fifteen peers are hosted on virtual machines spread over three physical computers.
- One of the peers acts as a supernode for the complete duration of the test.

The full middleware client is used in this setup, and the full functionality of the system is available. The directory server is hosted on the same computer as the server.

In this test, the player mobiles are constrained to a single ingame region and are connected for the full duration of the test. The first player to login assumes the role of supernode and maintains that role throughout. Once the supernode is established, all other players login to the game and all macro scripts are started.

### Results & Discussion

The first comparison, shown Fig. 4.2 and tabulated in Appendix C, is between the slave nodes in the hybrid system, and the standard client nodes of the C/S system. As each slave is responsible for providing movement updates to each of its neighbours located within its area of interest, the movement bandwidth shows an increase of 23%. The increase in outgoing movement packets is not unexpected as a middleware movement update is  $9n$  bytes per move made by the player, where  $n$  is the number of nodes currently within interest range. This replaces the single 7 byte event that

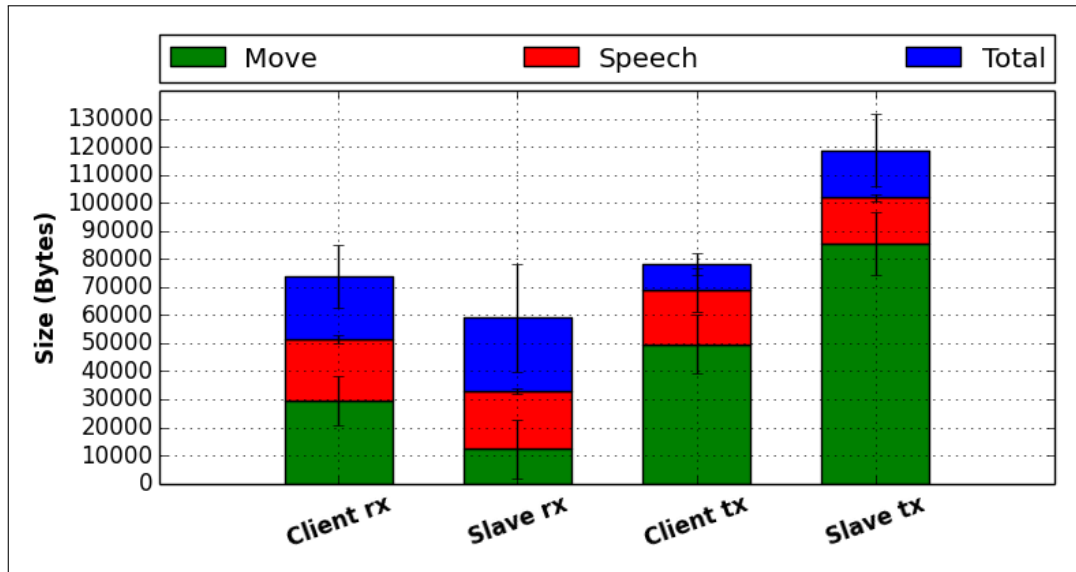


Figure 4.2: Bandwidth comparison between clients and slaves

was transmitted to the server per movement in the C/S system. The 58% decrease in incoming movement packets is attributed to the slave no longer having to receive a movement acknowledgement from the server for every move it makes.

The non-distributed speech packets see no significant change between the C/S and the hybrid system for slaves. This as a result of no real change in protocol of how they are handled, besides their rerouting through the supernode.

Overall, the 17% increase in slave bandwidth is acceptable for the purpose of the hybrid system, especially if the slaves do see a significant decrease in their latencies. This increase in slave bandwidth should translate to a reduction in server bandwidth.

The hybrid system's server bandwidth is compared with the C/S system in Fig. 4.3 and the tabulated results are available in Appendix C. The results proved to be opposite to what the predetermined goals of the system were. Instead of the reduction in bandwidth, the server experienced a 35% increase overall with the primary contributor being the 137% increase in movement traffic received by the server. This contradicts the expected reduction that periodic updating would offer.

The cause of this fault stems from the size of the periodic update packets, where every 3 seconds  $74n$  bytes of data are sent to the server, where  $n$  is the number of supernode's slaves who have moved in the previous 3 second time period. The large packets are a result of the plain text command sent to the server resulting from the UO server constraints as mentioned previously. Taking this into consideration, the

Table 4.2: Number of movement events/updates received by the server in the C/S and hybrid systems respectively.

System	Move rx	Size	Count
Client/Server	729 068	7	104 153
Middleware	1 724 977	74	23 311

number of movement updates in the hybrid system, and the number of movement events in the C/S system are shown in Table 4.2 and compared in Fig. 4.4. This shows that periodic updating indeed reduced the number of updates sent to the server by 77% from 104 153 to 23 311.

The server still redundantly processes any changes in the game state and produces updates accordingly. This results in the server sending movement updates for each periodic update it receives. However, the size of each movement update packet produced by the server is unchanged between the two systems, and thus the 28% decrease in movement traffic transmitted by the server reflects the reduction in bandwidth resulting from fewer events through periodic updates.

The changes in bandwidth for the supernode, as compared to a client in the C/S system, are shown in Table 4.3. The distribution of traffic is shown in Fig. 4.5. As expected, the bandwidth experienced by the supernode increases. The large 2 899% increase in movement traffic received is a result of all slaves transmitting every one of their movement updates to the supernode so that it is able to process AOIs accurately and produce periodic updates for the server.

In addition to the changes in movement bandwidth, it experiences a large increase of 2 575% in speech bandwidth due to the connection rerouting. Each slave that generates a speech event routes it via the supernode to the server. The supernode then receives the updates from that event from the server and sends the updates to its slaves.

While the movement and speech each show individual increases, the portion of traffic that stands out in the total distribution shown in Fig. 4.5 is the 42% of the bandwidth experienced by the supernode due to the periodic updating of player movement, including both the update commands sent from the supernode (accounting for 35%), and the redundant updates received from the server (accounting for the other 7%). While the periodic updates are necessary and were expected to increase the supernode bandwidth somewhat, they contribute significantly more than was expected due to the size of the updates.

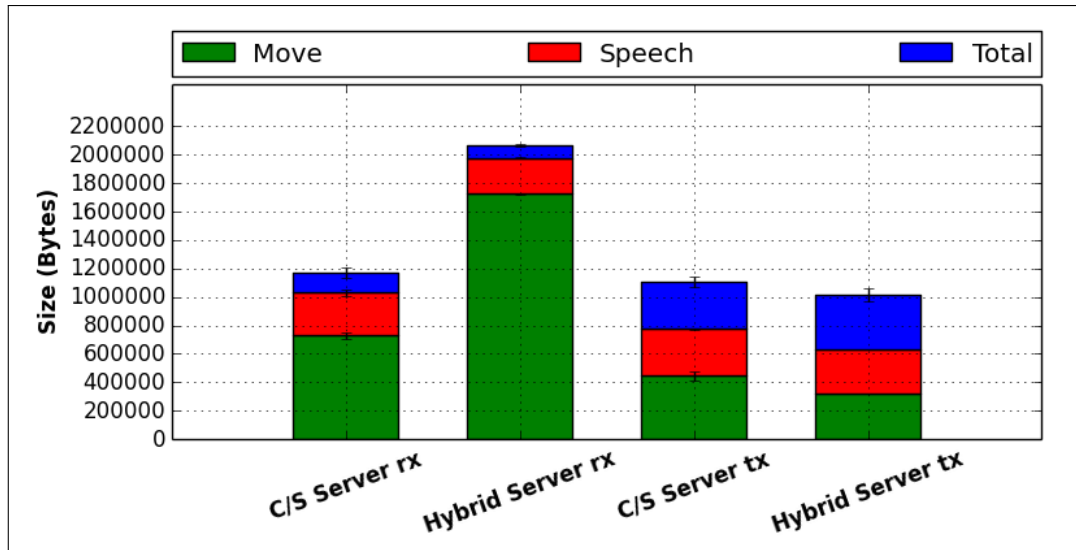


Figure 4.3: Bandwidth comparison between servers in the C/S and hybrid systems

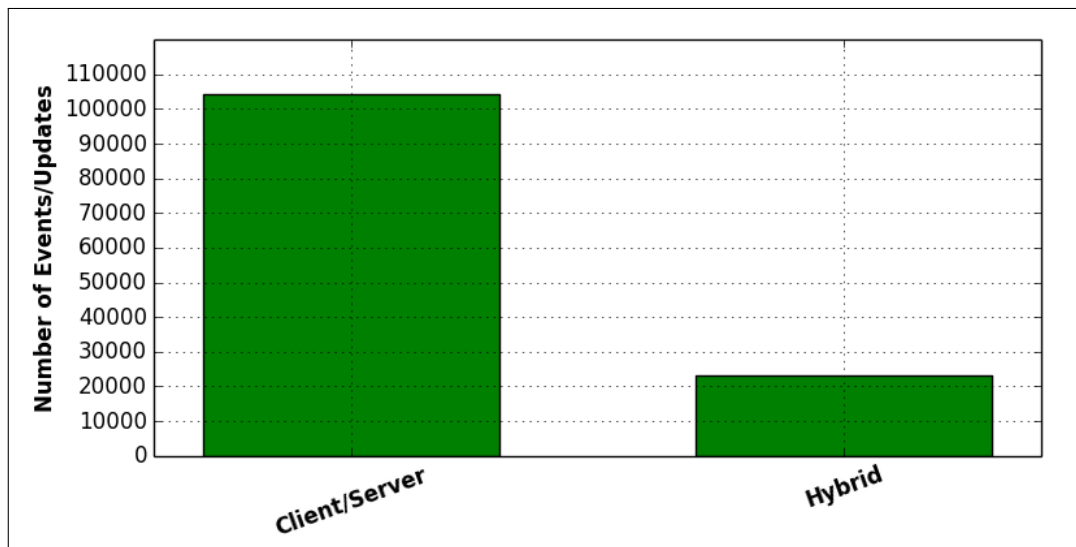


Figure 4.4: Comparison in the number of movement events in the C/S system and periodic updates in the hybrid system received by the server respectively.

Table 4.3: Bandwidth values of supernode in the single region hybrid system and change from clients in the C/S system (B)

Aspect	Mean	Std	C/S Mean	Change
Move rx	886 248	8 190	29 554	+2899%
Move tx	48 856	17 293	49 616	-1%
Move both	935 104	16 474	79 170	+1081%
Speech both	1 098 931	8 288	41 085	+2575%
Periodic upd rx	318 052	1 129	-	-
Periodic upd tx	1 724 977	6 122	-	-
Total rx	2 201 931	39 691	73 661	+2889%
Total tx	2 671 907	35 415	78 103	+3321%
Total both	4 873 838	75 106	151 764	+3111%

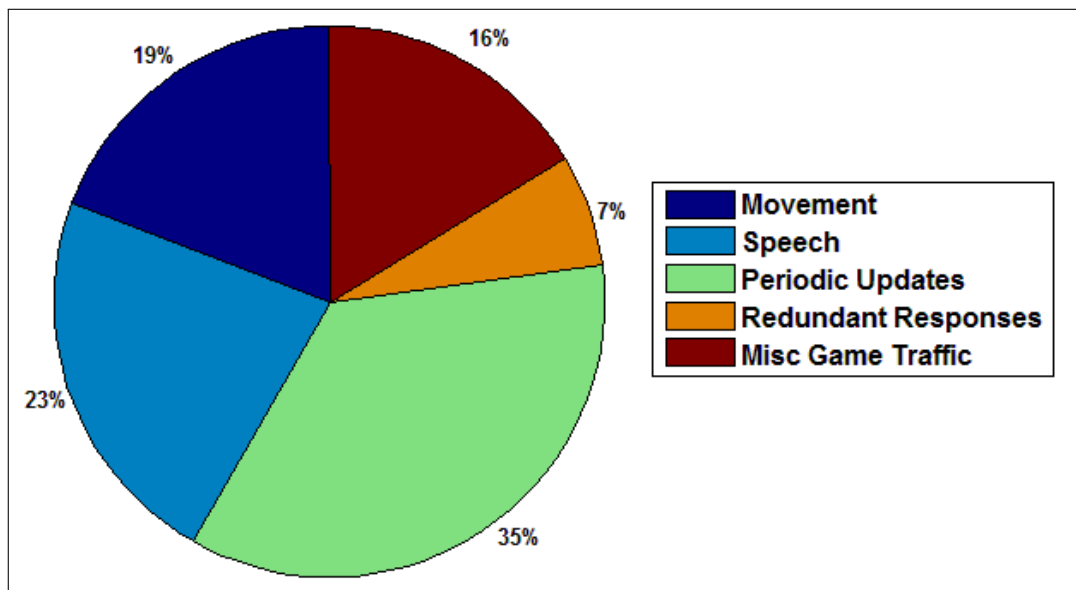


Figure 4.5: Supernode Bandwidth Distribution

Finally, the cost of hosting a slave as a supernode was investigated and is shown in Table 4.5. In the single region configuration, it shows that each slave accounts for 13.57 Bps inbound and 6.54 Bps outbound of the supernode. This excludes the additional bandwidth between the supernode and server for periodic updates. Taking periodic updates into consideration, these values increase to 16.51 Bps inbound and 22.51 Bps outbound.

With these effects established, it is worth taking into consideration the potential benefits of the system with some flexibility in the constraints of modification. The

Table 4.4: Potential comparisons between nodes with server modifications (B)

Node type	C/S	Modified	Change
Slave	151 764	177 642	+17%
Supernode	151 764	3 180 466	+1995%
Server	2 278 310	1 389 018	-39%

Table 4.5: Bandwidth cost per slave for supernode (Bps)

Direction	Mean	Std
Received	13.57	0.734
Transmitted	6.54	2.080

data represented in the current periodic update packet could be represented in a  $15n$  byte packet rather than the  $74n$  bytes it is currently using in text format. If an additional packet could be inserted into the native server code, then the amount of data as a result of periodic updates could be reduced as follows:

$$\frac{\text{total updates}}{\text{bytes/node}} * \text{potential bytes/node} = \frac{1724977}{74} * 15 = 349657.5 \text{ B} \quad (4.1)$$

If another modification could be introduced into the system to not produce redundant updates for changes received via this new packet, they could also be ignored. These two changes could produce the potential reduction in supernode and server traffic portrayed in Table 4.4. These changes are more encouraging, since the server shows a *reduction in bandwidth* of 39%.

Furthermore, if the server could be modified to accept client events from any source, the slaves could communicate directly with the server rather than routing their non-distributed traffic through the supernode. If this, combined with the previous suggestion, was implemented, the supernode could experience a total bandwidth usage of 2 118 740 B, an increase of only 1 296%. The final potential values are compared to their C/S counterparts in Fig. 4.6, highlighting the potential significant reduction in server bandwidth.

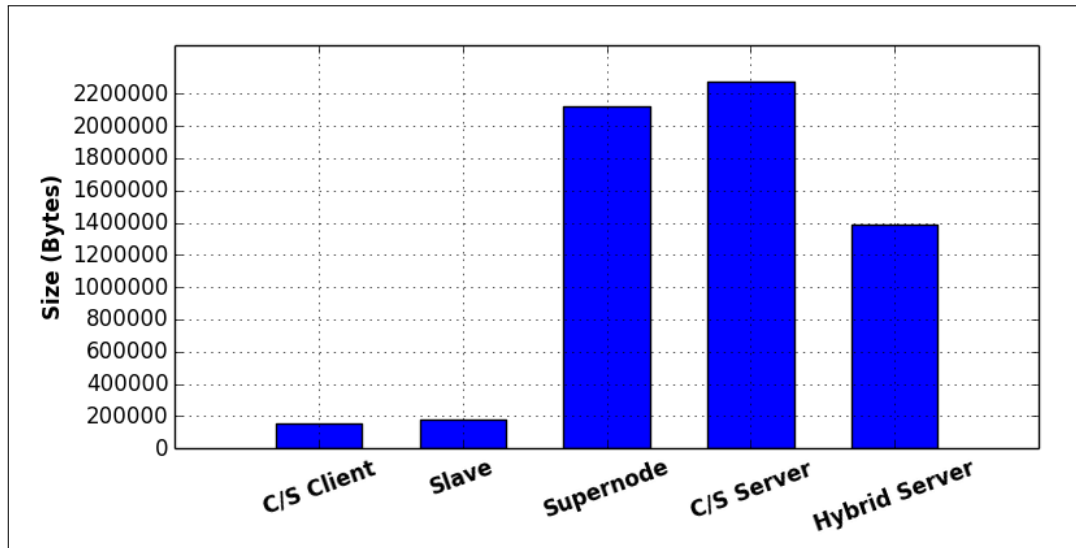


Figure 4.6: Potential bandwidth totals of hybrid nodes with client/server nodes for reference.

### 4.2.5 Test 1.3: Inter-region Bandwidth

#### Motivation

With the single region effects of the system established, the inter-region effects of the hybrid system should be observed. The IM scheme introduces migrations, supernode handovers and relogs into the system, as well as additional overhead to maintain the IM structure. The purpose of this test is to determine the effect of the inter-region IM on the bandwidth experienced by the server, clients and supernodes.

#### Configuration

Fifteen peers, each consisting of the game client, middleware and macro application, and a server are used in this test in the following setup:

- The server is hosted on a dedicated computer.
- Fifteen peers are hosted on virtual machines spread over three physical computers.

The full middleware client is used in this setup, and the full functionality of the system is available. The directory server is hosted on the same computer as the server.

In this test, the player mobiles are constrained to the shared virtual area containing multiple regions. Each client logs into the game through the middleware and the macro scripts are started.

## Results & Discussion

Two items were identified as the main components of inter-region traffic: migrations and interest management overhead. The inter-region bandwidth is dependant on a number of different factors highlighted in these results.

**Migrations** On average, a login is relatively expensive and incurs the traffic shown in Table 4.6 to the migrating peer. Because players are continuously moving, many migrations occur, causing regular transformations and connection rerouting that was described in section 3.5.3. This generates significant inter-region bandwidth, contributing negatively towards the hybrid system. An extract of the results are shown in Table 4.7. Large standard deviations exist for the role duration, as well as for the number of slaves managed per supernode, and their associated login traffic. There is a large discrepancy between the supernodes who performed their duties the longest and shortest and, similarly, there is a large discrepancy between the number of slaves handled by the supernodes. This suggests that the inter-region effects are highly dependant on player behaviour, and accurate trends might only be observable with large numbers of players over a long duration of gameplay. However, all results suggest that the relog bandwidths negatively influence the system, albeit at different degrees.

The cost of migrations in terms of login overhead has a significant impact on the supernode's cost for hosting a slave. In the previous test, the outbound cost per slave for the supernode in the single region was shown to be 6.54 Bps. The outgoing per second bandwidth is shown in Fig. 4.7 for different durations of ownership. In this figure, the large spikes in bandwidth (up to 1150Bps) for shorter durations indicates how costly owning a slave can be for a supernode if it is only owned temporarily before another migration occurs.

Over the full duration of the test, 332 migrations occurred, causing an additional 1 033 848 bytes of traffic for the server: a +45% increase to the bandwidth experienced by the C/S server seen in Test 1.1. From the mean values, a supernode experiences an average of an additional 31Bps in each direction from relog traffic alone while performing its duties.

While migrations are unavoidable, the bandwidth increase that they introduce highlights the importance of tuning the IM scheme to reduce the impact of high-cost transients. By predicting which peer will remain in a region the longest, it can be



Table 4.6: Cost of a login to the migrating node (B)

Direction	Mean	Std
Received	363.18	54.35
Transmitted	2751.30	486.22
Both	3114.48	524.53

Table 4.7: Supernode inter-region bandwidth showing migration effects according to duration as a supernode, number of slaves over the period, and the amount of re-login traffic handled.

Duration(s)		Slaves		Login(B)		Comment
<b>2155.48</b>		13		75256		Longest
1900.59		<b>28</b>		167332		Most
<b>11.05</b>		1		5984		Shortest
Mean	Std	Mean	Std	Mean	Std	-
456.6	530.4	4.8	5.6	28361	34031	-

selected as a supernode to avoid the bandwidth spikes incurred by relogs occurring from repetitive handovers. Introducing dynamic regions that scale according to how much the supernode can handle can allow regions to change in size, in some cases becoming bigger, allowing players more movement before they must migrate.

Ultimately, the server could be modified in one of two ways to eliminate the heavy migration costs: (1) the server could not produce a full login sequence in the case of a migration, but only pass the necessary information, greatly reducing the size of a relog; or (2) the server could accommodate direct connections between slaves and the server while still providing the necessary functionality, eliminating the login traffic completely along with other benefits mentioned throughout the tests.

**IM Overhead** The overhead costs for maintaining the interest management system is the other major contributor to inter-region traffic, although only to peers and not the server. The tasks for maintenance are separated into 2 portions: slave instruction and neighbour updates.

*Slave instruction* involves the supernode sending commands to its slaves on who to connect and disconnect with when other players enter and leave the slaves' areas of interest. However, the bandwidth generated by these instructions appears to be minimal. Each command is 13 bytes in size, and are only sent when there is a

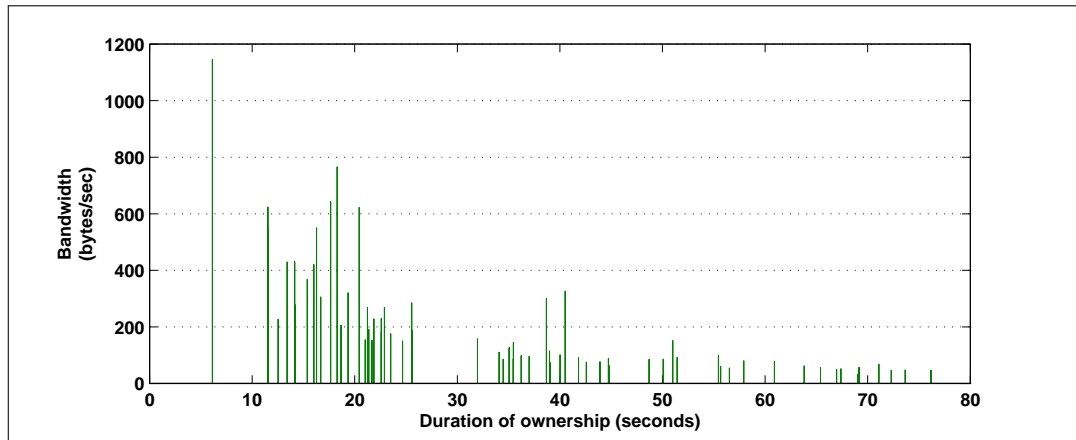


Figure 4.7: Bandwidth impact on supernode for hosting a slave at different durations

change in a player's AOI. Therefore two players remaining in a shared AOI generate no additional overhead. During the test, only 1 supernode experienced more than 1 Bps as a result of this component. All others were below 1 Bps over the full duration of their time as a supernode, for multiple slaves.

The *updating of neighbouring supernodes* contributes more towards the total overhead, but is also very dependant to the arrangement of players in the regions. The size of the updates, as previously mentioned in section 3.5.4, were shown to be dependant on the number of slaves being managed by the supernode, as well as the number of occupied immediate neighbouring regions.

An extract of bandwidth resulting from IM maintenance are shown in Table 4.8. In this table, the amount of neighbour update traffic experienced throughout the duty as supernode is shown in both directions. Again, the standard deviations are seen to be large in comparison to the mean values, suggesting large variations in each supernode's experience. An example of these discrepancies is shown by the supernode who received the most updates (23 836 B) and must have been neighbouring a heavily populated region, while its own region remained sparsely populated, resulting in it only transmitting 1 495 B to its neighbours. Also shown in the table is the amount of slave instruction traffic sent by a supernode throughout its duty and the large discrepancies of recorded values between 0 and 1 339 B.

Again, these figures illustrate that the inter-region traffic is highly dependant on player behaviour, and accurate trends might only be observable with large numbers of players over a long duration of gameplay. The IM overhead bandwidths differ

Table 4.8: Supernode inter-region bandwidth showing overhead effects according to duration as a supernode, number of slaves over the period, the amount of neighbour update traffic experienced inbound and outbound, and the amount of AOI instruction sent.

Duration(s)		Slaves		Upd rx(B)		Upd tx(B)		Instruct(B)		Comment
<b>2155.48</b>		13		14378		807		0		Longest
1900.59		28		15320		<b>35003</b>		<b>1339</b>		Most
1190.87		7		<b>23836</b>		1495		138		Most
Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	-
456.56	530.36	4.8	5.6	2956.5	5136.2	2843	6581.1	96.18	246.09	-

Table 4.9: Supernode inter-region bandwidth makeup of total traffic throughout duration

Component	Portion	
	Mean	Std
Migration	16.88%	8.04%
Overheads	4.70%	4.05%
Combined	21.58%	9.20%

greatly depending on the current state of the originating and neighbouring regions.

Table 4.9 shows a summary of how much inter-region bandwidth constitutes the total bandwidth of supernodes for the duration of their duty. While the migration traffic constitutes a significant portion in the bandwidth experienced by supernodes at 16.88%, the overhead for managing the IM scheme is reasonable at only 4.7%.

No other significant trends could be established from this test due to the volatility in the results. It is recommended that this test is run at a much larger scale, with hundreds of *real* players over a long duration, so that more significant details about the inter-region bandwidths can be determined.

## 4.2.6 Conclusion

The measurement of bandwidth data revealed a number of valuable points relating to the bandwidth changes in the hybrid system. The middleware plugin, as it currently stands, shows that all nodes in the system experience an increase in their bandwidth.

The increase seen in slaves is within an acceptable level, seeing only a minor growth of 17% in the single region, plus small overhead for interest management

instructions. The slave only experiences poor bandwidth performance when it is regularly migrating between regions, generating relog traffic between short intervals.

As one of the primary goals of the system was to reduce server bandwidth, the resulting increase is an unexpected outcome. The three causes of the recorded increase are the large periodic updates sent by the supernode, the redundant updates returned to the supernode, and the large login packets that are exchanged as a result of migrations.

The supernode experiences a huge increase in its bandwidth over a client in the standard system, servicing more traffic for its region than the server does. This will become a definite problem if a region becomes densely populated to the point of overloading its supernode's capabilities. The causes of the increase are the same as those of the server, coupled with the need for the slave connections to be routed through the supernode.

However, the unexpected outcomes are all shown to be a consequence of the server constraints. The following three server modifications should result in the expected decrease in server bandwidth: (1) Introduce a custom packet for aggregated updates, thereby reducing the required supernode/server bandwidth. (2) Eliminate the redundant server updates that are returned from the server from the aggregated periodic updates. (3) Accept client events from multiple sources, which allows slaves to connect directly to the server and send non-distributed packets directly. This modification eliminates the need for the supernode to forward non-distributed traffic. Finally, it will also eliminate bandwidth resulting from relogs.

The current system does not perform as expected and does not achieve its bandwidth objectives. However, the expected change in bandwidth that could be achieved with minor modifications to the server are promising and show that a middleware plugin can reduce the server bandwidth while not over-burdening the clients.

## 4.3 Latency Evaluation

### 4.3.1 Overview

The purpose of the latency tests is to measure the difference in latencies for nodes in the network between the original Client/Server system and the hybrid system.

In this section, the protocol modifications necessary to measure latencies are detailed; the test platform used for the tests is described; the three practical tests that were performed are discussed with respect to their motivation, configuration

and results.

### 4.3.2 Protocol Modification

This test measures the time delay between when packets are sent and received. As it is not possible to ensure that the clocks of different nodes are in synchronisation, the test rather measures RTT to use only the source node's clock. This requires that packet responses are introduced into the protocol for packets that are usually only one directional, such as a movement update. Furthermore, as multiple packets of the same type are transmitted at any time, it is necessary to distinguish between them, requiring that some form of identification between packets is incorporated.

This leads to the following modifications to the protocol. Each movement update packet broadcast from a peer to his neighbour list is affixed with a unique ID number. When a peer receives a movement update packet, a response containing the ID is immediately returned to the original sender. Similarly, when a speech packet is sent, the speech text is set to a unique ID number and transmitted to the server. The server returns the speech, including the number, as an update to the client. This allows an affixed ID to be transmitted to the server without any packet modification.

Whenever an ID appended packet is sent or received, an appropriate event is recorded with a timestamp by the statistics module built into the application. This allows the difference in time to be calculated as the round trip time of the packet delivery.

### 4.3.3 Test Platform

For the effects on latency to be properly gauged, nodes must be situated at different distances away from the server and each other. This will highlight any significant differences that would be invisible if the latency was negligibly small. This led to the use of Amazon Web Services (AWS) as a testing platform [56]. AWS allows a user to launch virtual machine instances hosted by Amazon in the cloud, and provides the user with access to the system. A user may then use the virtual machines to perform computations, host a server, or execute any software that would be run on a regular operating system.

AWS offers different global regions around the for virtual instances to be hosted in, providing the user with a public IP to access the virtual instance over the Internet. Three AWS regions were chosen to offer varying delays between each: US West (Oregon), US East (North Virginia) and Ireland.

Table 4.10: Round Trip Time to Server per Region (ms)

Origin of Event					
US West		US East		Ireland	
Mean	Std	Mean	Std	Mean	Std
6.457	20.136	94.725	37.811	196.964	74.996

### 4.3.4 Test 2.1: Latency Baseline

#### Motivation

The purpose of this test is to establish a baseline with the C/S system to which the hybrid system can be compared. It requires that nodes simply have their RTTs recorded between their own location and the server to derive an average latency between each region and the server.

#### Configuration

Using 15 clients spread evenly between the 3 regions and the server hosted in 1 of the regions, the node distribution is as follows:

- The server is hosted in the US West region.
- Five clients are situated in the US West region.
- Five clients are situated in the US East region.
- Five clients are situated in the Ireland region.

Since the client and server software has not been modified, simplified middleware is used to record the latency measurements. The client connects to the server via the middleware, and every packet is transparently passed through the middleware, without applying any game logic, whilst inserting packet identifiers and recording the timestamps as the packets are transmitted.

Each client executes the macro script for a period of 2 hours. All player mobiles are limited to a shared virtual area to ensure occasional interaction.

#### Results & Discussion

After computing the round trip time for each recorded packet, the mean and standard deviation of each region were calculated and appear in Table 4.10.

As expected, the nodes located in the same region as the server experience the lowest latencies. It is also clear that as the distance from the server increases, so does the latency. These results serve as a basis for comparison with the hybrid system.

### 4.3.5 Test 2.2: Latency with Supernode situated near server

#### Motivation

This setup aims to capture latency information of movement and speech packets for the hybrid system. This test is to measure (1) the RTT of movement updates originating from peers to update their neighbours, ie the time delay in when a distributed update is perceived, and (2) the RTT of speech events being sent to the server and returned as an update to the originating node.

This test, performed in conjunction with Test 2.3, aims to reveal if the selection and placement of supernodes influences the latencies experienced by different nodes in the system. This is the first of the two extremes, where the supernode is located in the same region as the server, rather than in the furthest region. In order to measure reliable results, the test is run in a static single virtual hexagonal region with only one supernode for the duration of the test. This allows for definite results regarding the supernode placement to be derived.

#### Configuration

Using the same amount of clients as the client/server system, the supernode is placed in the same region as the server. The node distribution is as follows:

- The server is hosted in the US West region.
- One supernode is situated in the US West region.
- Four slaves are situated in the US West region.
- Five slaves are situated in the US East region.
- Five slaves are situated in the Ireland region.

The full middleware client is used in this setup, and the full functionality of the system is available. The directory server is hosted on the same virtual instance as the server.

In this test, the player mobiles are constrained to a single virtual region and are connected for the full duration of the test. The first player to login assumes the role of supernode and maintains that role for the full duration of the test. Therefore, the first node to login is in the same region as the server. The rest of the configuration is the same as for Test 2.1.

Table 4.11: Average RTT of speech (non-distributed) with the supernode in USWest Region (ms)

Origin of Event					
US West		US East		Ireland	
Mean	Std	Mean	Std	Mean	Std
15.03	38.99	115.32	105.74	213.95	105.91

Table 4.12: Average RTT of movement (distributed) with the supernode in USWest region (ms)

	Destination					
	US West		US East		Ireland	
Origin	Mean	Std	Mean	Std	Mean	Std
US West	14.05	74.38	129.50	176.72	300.37	191.80
US East	114.66	124.59	27.78	248.74	145.04	216.50
Ireland	269.92	226.95	134.42	164.2	11.67	33.01

## Results & Discussion

The speech data was processed to produce round trip times for each recorded packet that was transmitted from the originating region, via the supernode, to the server. The round trip times experienced by each region were averaged and the standard deviation taken of each: the results are shown in Table 4.11 and compared with the C/S results in Fig. 4.8. As expected, the round trip times do not show a significant difference when compared to the baseline. Each value is roughly the original C/S RTT of that region with the addition of one US West C/S RTT resulting from the extra hop.

The movement data was processed to produce round trip times for each recorded packet and the destination region of the packet. The round trip times experienced by each region were averaged and the standard deviation taken of each: the results are shown in Table 4.12 and the best case internal-region latencies are compared to the C/S system in Fig. 4.9.

Regions far from the server see significant internal improvements in the time between when an event is produced to when the updated state is perceived by interested parties. This is particularly noticeable in Ireland where the original C/S RTT of 196.96ms is reduced by 94% to the internal region RTT of 11.67ms. Based on the premise that friends play together, players who are located nearby geographically



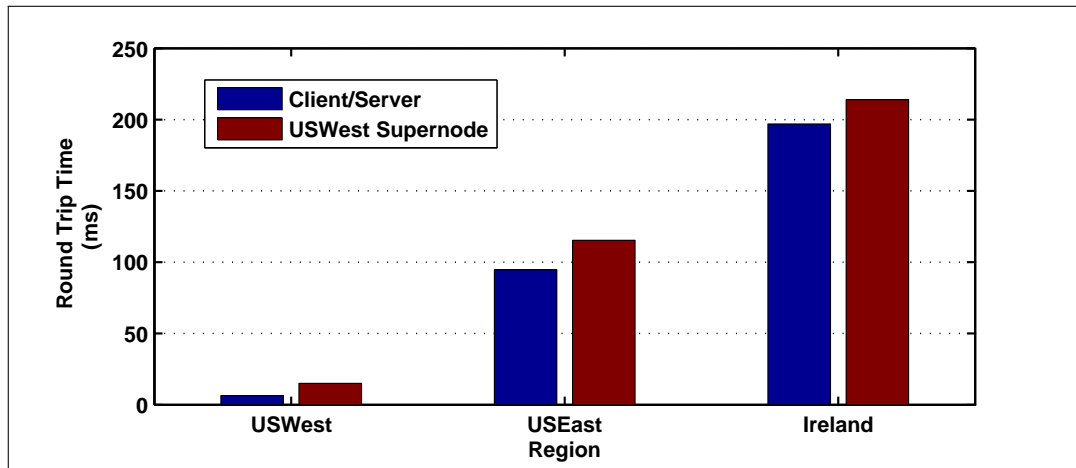


Figure 4.8: Comparison of Round Trip Times for clients sending non-distributed events and receiving their associated updates between the C/S system and hybrid system with a supernode located in the US West region.

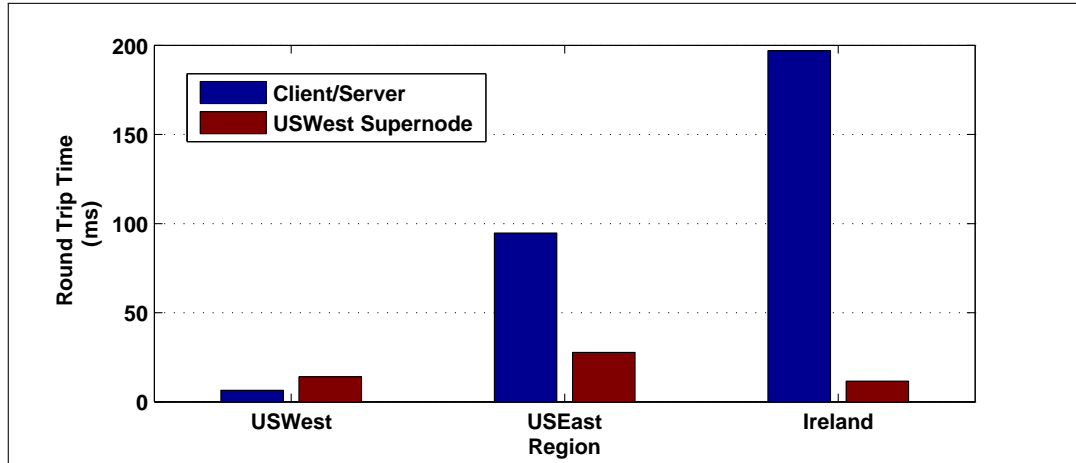


Figure 4.9: Comparison of distributed latencies for peers communicating within the same region between the C/S system and hybrid system with a supernode located in the US West region.

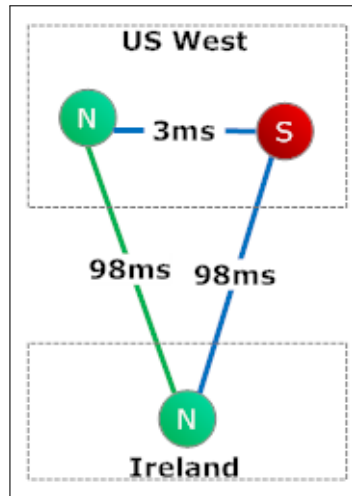


Figure 4.10: Comparison of distributed mechanic latencies between peers where green connections indicate hybrid latencies, and blue indicate C/S latencies.

will see significant improvements to their gameplay experience.

For nodes located near the server, the reduction is not quite as significant. A US West node experiences average movement RTTs of around 14ms, close to its baseline RTT of 6ms. However, the latency does become independent of server load, thus a heavily loaded server will no longer affect the movement latencies experienced by players, even if they are located nearby the server.

In the worst case, where two nodes are located geographically far apart, there is still a reduction over the standard C/S system. If the one-way latency between Irish and US West regions can be approximated to  $1/2$  of the RTTs measured in the C/S system, then the latencies of 2 distant nodes and the server can be described as in Fig. 4.10. This shows that the hybrid system offers a latency of 98ms between the nodes as opposed to the 101ms in the C/S system — a reduction of 3.1%, albeit minor.

While these are promising results, further investigation into the effects of super-node placement is necessary.

Table 4.13: Average RTT of speech (non-distributed) with the supernode in Ireland region (ms)

Origin of Event					
US West		US East		Ireland	
Mean	Std	Mean	Std	Mean	Std
390.80	196.72	293.09	93.54	218.77	211.01

### 4.3.6 Test 2.3: Latency with Supernode situated far from server

#### Motivation

This setup follows closely on Test 2.2 with the aim of gathering the same information. The test also runs in a single in-game region occupied by one supernode and 14 slaves. However, in this test, the supernode is deployed in the furthest AWS region from the server, in order to determine if the selection and placement of supernodes influences the latency experienced by nodes in the system.

#### Configuration

Using the same amount of clients as the base case, the supernode is placed in the furthest region from the server. The node distribution is as follows:

- The server is hosted in the US West region.
- Five slaves are situated in the US West region.
- Five slaves are situated in the US East region.
- One supernode is situated in the Ireland region.
- Four slaves are situated in the Ireland region.

This test follows the same configuration as Test 2.2 with the difference that the first node to login is from the Irish region.

#### Results & Discussion

The speech data was processed according to the same method as the previous test: the results are shown in Table 4.13. A comparison of the results to the C/S system and the hybrid configuration of Test 2.2 are shown in Fig. 4.11. In this instance, the significant impact of supernode placement can be observed. There is an insignificant change in latencies experienced by nodes in the supernode's region, where speech latencies change from 196.97ms to 218.77ms, while nodes located in other regions are affected poorly.

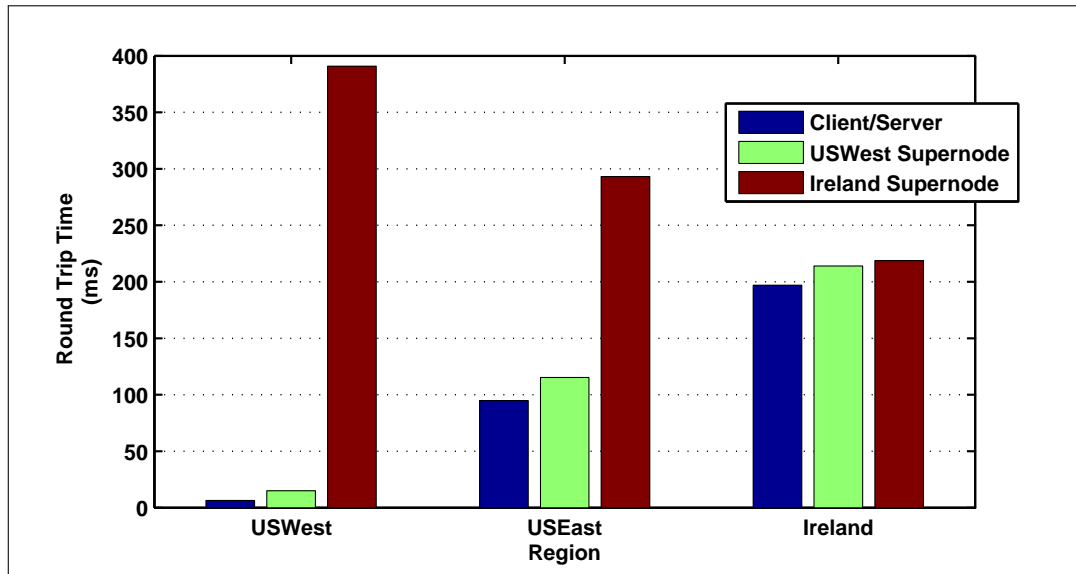


Figure 4.11: Comparison of Round Trip Times for clients sending non-distributed events and receiving their associated updates between the C/S system and hybrid system configurations.

Over the standard C/S setup, the nodes located in the US East region experience an increase in latency of 209% from 94.73ms to 293.09ms, and the nodes located in the US West region experience an increase of 5952% from 6.46ms to 390.8ms. This is a significantly negative impact of the hybrid system on non-distributed game mechanics. If players who are located near to the server are unfortunate enough to get paired in a region with a distant supernode, it will result in a serious degradation in the gameplay experience. Even in the case where the supernode is the originator of the event, a distant node will experience a bigger latency, as shown in Fig. 4.12 where an event originating from the supernode in Ireland is only perceived by the US West node 294ms after it was generated, rather than 101ms in the C/S system. This highlights the importance of supernode placement within the P2P network, and presents an opportunity to define criteria for supernode selection as the player in the region who is nearest to the server. However, this still only serves as a way to minimize the negative impact, but not eliminate it. The severe impact of the supernode situated in Ireland is clearly visible in Fig. 4.11 where all nodes experiencing their highest RTT of non-distributed latencies in that setup. Also visible in the figure is the negligible impact that a supernode in the same region as the server has over the C/S system: where the US West supernode and Client/Server RTTs are similar.

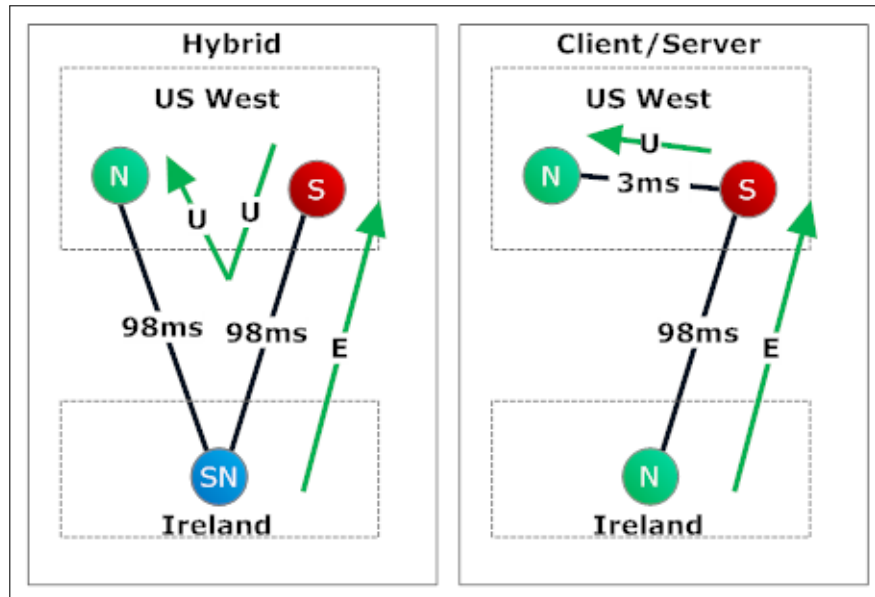


Figure 4.12: Comparison of non-distributed mechanic latencies of updates between regions in the hybrid and C/S systems

Table 4.14: Average RTT of movement (distributed) with the supernode in Ireland region (ms)

	Destination					
	US West		US East		Ireland	
Origin	Mean	Std	Mean	Std	Mean	Std
US West	13.34	83.65	115.18	131.20	270.98	169.79
US East	134.97	325.20	14.02	98.78	155.63	354.69
Ireland	296.70	288.92	147.22	102.60	24.04	258.26

The movement data was processed according to the same method as the previous test: the results are shown in Table 4.14. The best case latencies are shown in Fig. 4.13 where the movement RTTs are compared between the standard C/S system and each of the hybrid system configurations when communicating within the same region. Comparing this to the results of the previous test, it can be concluded that the placement of supernode has an insignificant effect on the latencies belonging to the distributed mechanics. In all instances, the RTTs show an insignificant change between the configurations, such as 134.97ms compared to 114.66ms when movement updates originate from a US East node and are perceived by a US West node.

Overall, there is a significant reduction in distributed mechanic latencies, espe-

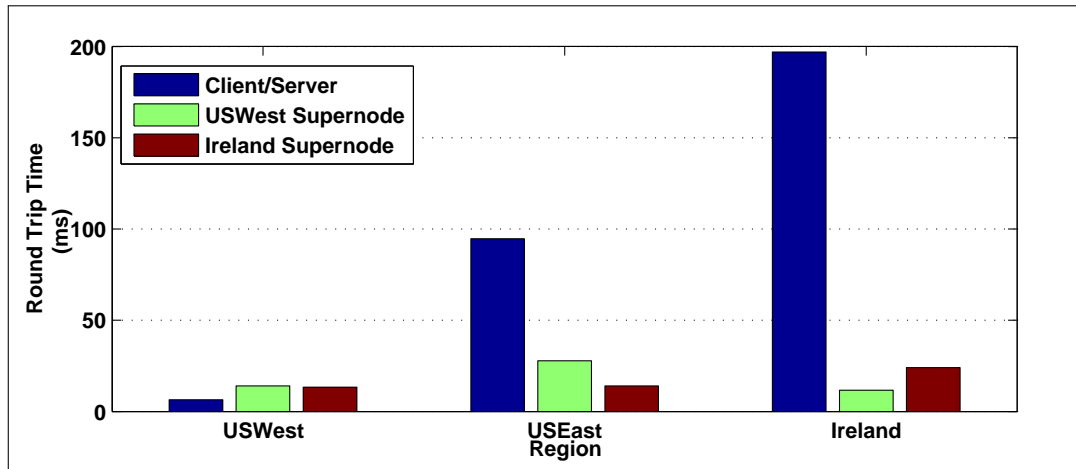


Figure 4.13: Comparison of distributed latencies for peers communicating within the same region between the C/S system and hybrid system configurations

cially noticeable in the Ireland region.

### 4.3.7 Conclusion

The outcomes of the tests showed that the expected latency results can be achieved in all but one aspect of the game for the hybrid system.

The distributed movement latencies showed a decrease in all regards, with significantly large reductions for players who are distant from the server but local to each other. This is a very positive outcome for the system. Based on the premise that groups of friends from the real world play together as groups in the virtual world, the latency reductions can provide them with a highly improved gameplay experience. Many game servers are international and have players from distant geographical locations connecting to them, and with the middleware they would no longer suffer from international latencies with distributed mechanics.

Non-distributed mechanics result in a range of different latency changes depending on the location of the supernode. In the worst case it was observed that a player who is nearby the server can experience an unacceptable increase of nearly 6000% in its latency. However, since the goal of the system is to distribute most, if not all, of the game mechanics, this increase in latency can be minimised when non-distributed mechanics are in the minority. The priority at which mechanics are distributed can be related to their latency requirements — where mechanics requiring low delays are prioritised, and non-time sensitive mechanics are not.

However, once again the poor performance is a result of the UO server constraints. If modification to the server is allowed, using the same modifications that were mentioned in the bandwidth tests, the effects of selecting a distant supernode can be entirely eliminated by allowing slaves to connect directly to the server. This would result in slaves experiencing similar latencies for non-distributed mechanics as clients do in the baseline C/S system.

The current system achieves one of its objectives by showing that distributed mechanics can hugely benefit from large decreases in latency for players located nearby one another, and be roughly equal to the C/S system when they are not located nearby. Non-distributed mechanics show that poor supernode selection can result in negative effects on the system, but can be eliminated by modifying the server. This supports the hypothesis that the hybrid system improves network latencies as experienced by the players.

## 4.4 Hardware Impact Evaluation

### 4.4.1 Overview

The purpose of the hardware performance tests is to measure the hardware usage of the middleware software, as well as comparing the hardware usage of the server software between the two systems. Investigation into this will show whether the hardware requirements of the server will be lowered by using the system, and how the middleware scales when performing supernode tasks.

In this section, the test platform used for the tests is summarised; the two practical tests that was performed are discussed with respect to their motivation, configuration and results.

### 4.4.2 Test Platform

The platform for testing is the same as described for bandwidth tests, using a number of VirtualBox Virtual Machines spread over multiple physical computers connected over a LAN.

For the correct perspective to be established, the same dedicated computer must be used by the software module under investigation. The specifications of the computer used in this test are shown in Appendix D.

The software's data capturing module samples the CPU utilization, Virtual Bytes and Working Set of the process every 1 second.

### 4.4.3 Test 3.1: Hardware Impact of server software

#### Motivation

For the effects of the system to be properly measured, they must be compared against C/S system baseline performance. It requires that the server records its CPU and memory usage over the duration of the test while an increasing number of clients connect to it, maintain their presence ingame and then disconnect.

With the baseline established, the middleware should be incorporated and the same procedure followed to record the CPU and memory usage of the server in the hybrid system.

#### Configuration

As the server is under investigation, it is hosted on the computer dedicated to measurements. The client peers are located on virtual machines. The test follows the following configuration:

- The server is hosted on the dedicated measurement computer.
- Fifteen clients are hosted on virtual machines spread over 3 physical computers.

With the server started, each client logs into the game with a 2 minute delay between entries. The client runs their macro scripts while additional clients are connected. When all 15 clients are connected, the configuration is left to run for 4 minutes. Each client is then logged out with a 2 minute delay between departures.

As measuring the hardware impact is more subjective, the test focuses on examining the changing trends of hardware usage for varying numbers of players, rather than the exact usage figures. Therefore, a shorter test duration is used to simplify the test, while still allowing enough time for each client to perform a set of actions.

Following this, the middleware is introduced to each of the clients and the directory server is run alongside the server. The same procedure is followed as before.

#### Results & Discussion

The collected results were processed to produce an average and standard deviation of each of the hardware attributes for each of the number of clients connected to



the system at that time. The tabulated results are shown in Appendix E while the summarised results are presented below.

The differences in CPU utilization were shown to be not significant, as seen in Fig. 4.14a. This shows that the middleware plugin does not reduce CPU utilization of the server, contrary to what was expected. As the server is still processing all of the non-distributed mechanics, handling the overhead of hosting players, as well as servicing the periodic updates and producing redundant updates for them, it is within reason as to why no change has been observed.

The differences observed in memory usage are shown in Fig. 4.14b and Fig. 4.14c. Neither the private bytes nor the working set show a significant change in memory usage, but only minor variations. The memory usage does not seem to change significantly between the number of clients that are connected to the game for a small number of players.

In all of these aspects, the true changes in hardware usage might only become evident when many more players are connected to the game, perhaps in the range of hundreds. In addition, the predicted benefits to the server might only come into effect if the proposed modifications from the previous tests are incorporated into the server.

#### 4.4.4 Test 3.2: Hardware Impact of middleware software

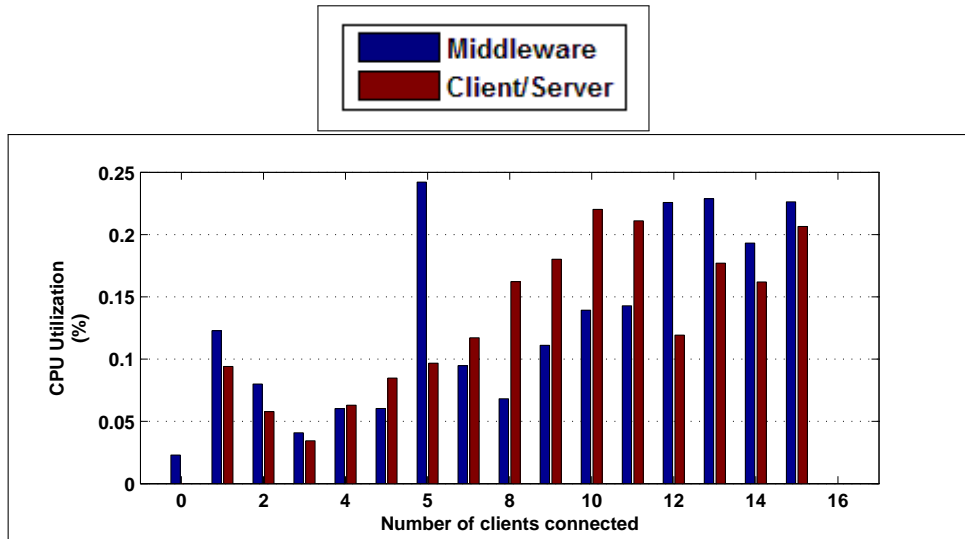
##### Motivation

The aim of this test is to judge the impact and scalability of the middleware software. By recording the hardware usage over time as more slaves connect through the supernode's middleware, it is possible to predict how significantly hosting more slaves will impact the system hardware.

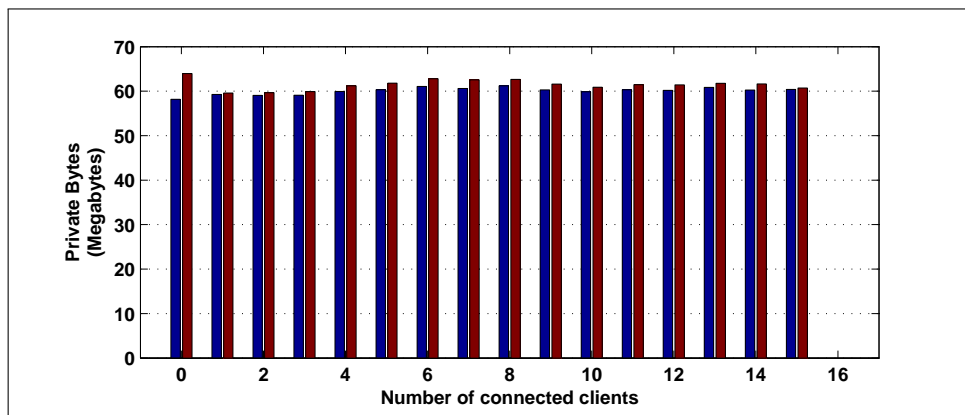
##### Configuration

The configuration is modified slightly from to appear as follows:

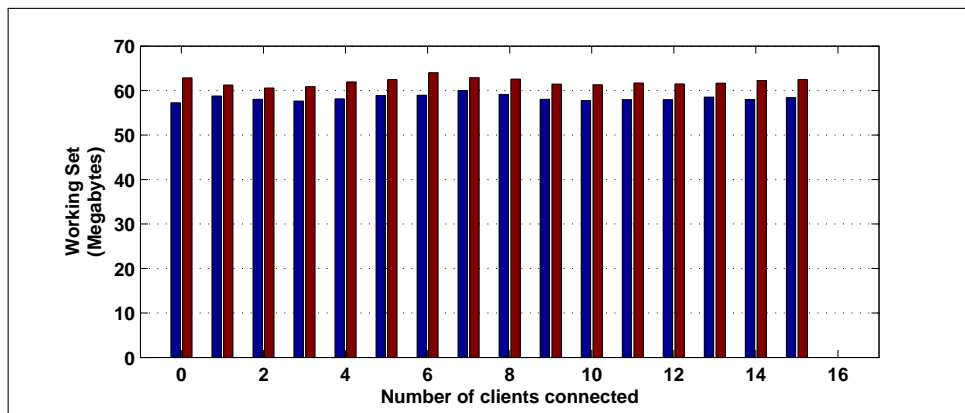
- A supernode is hosted on the dedicated measurement computer.
- The server is hosted on a dedicated computer.
- Fourteen peers are hosted on virtual machines spread over 3 physical computers acting as slaves.



(a) Comparison of CPU utilization between C/S and hybrid system.



(b) Comparison of Private Bytes between C/S and hybrid system.



(c) Comparison of Working Set between C/S and hybrid system.

Figure 4.14: Hardware Usage of server software between systems.

The directory server is hosted alongside the server, and the same test procedure is followed as in Test 3.1.

## Results & Discussion

With the data collected, it was processed to produce a mean and standard deviation of hardware usage for different numbers of slaves connected through the supernode. The raw results are shown in Appendix E and summarised below.

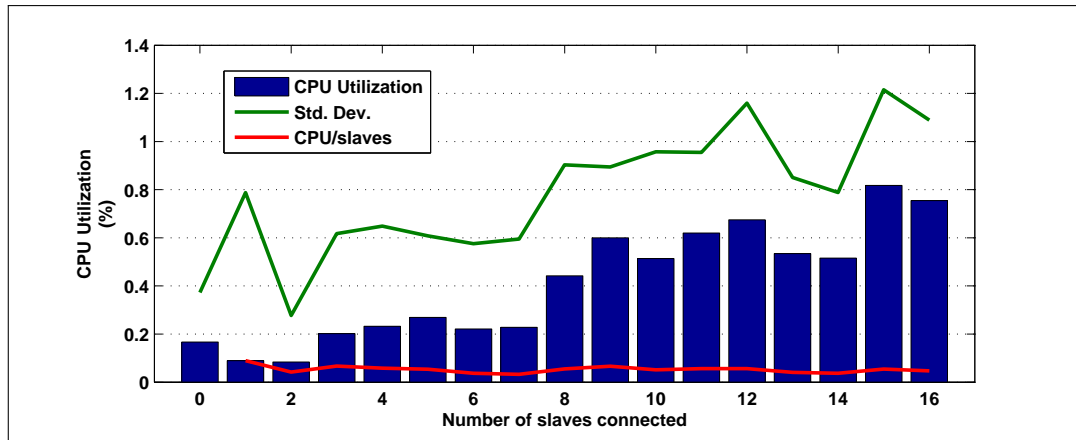
The CPU utilization is shown in Fig. 4.15a. In this it can be seen that CPU utilization is low in the case where the system is only hosting itself as 1 slave. By analysing the usage as the number of slaves increases, it appears to show that CPU utilization increases roughly linearly with the number of slaves that join. The standard deviation of the measurements also increase with the number of slaves, suggesting that larger spikes in usage also begin to appear as more slaves connect. Therefore, it can be concluded that if the supernode must host large numbers of slaves in well-populated areas, it will eventually experience a significant impact on its system CPU usage, and thus might begin to affect the gameplay itself by limiting the resources available to the game.

The memory usage, shown in Fig. 4.15b, shows that the system uses around 40MB at startup, and then increases to around 55MB as the player connects to the game, a 37% increase. Once assuming supernode duties with a few slaves connected, the memory usage stabilises at around 60MB and appears to proceed unchanged from that point, regardless of how many slaves are added.

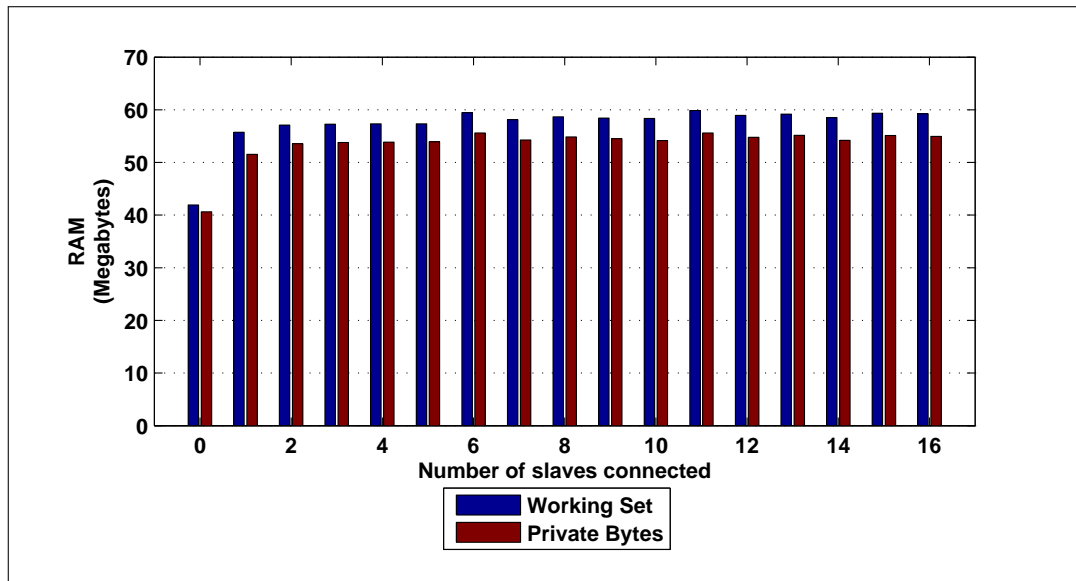
While these trends might hold true for smaller numbers of slaves, they might reveal different trends with significantly larger numbers of players.

### 4.4.5 Conclusion

The hybrid system appears to have a minimal effect on the server's hardware usage for the number of players in the tests. The CPU utilization appears unaffected by the system, and the memory usage shows insignificant variations. However, the effects of the hybrid system might become more visible with larger numbers of players. The potential benefits, such as reducing in the frequency of CPU usage spikes or the average CPU usage, might only take effect if the proposed server modifications from the previous tests are applied.



(a) CPU utilization of the middleware plugin.



(b) Private Bytes and Working Set of the middleware plugin.

Figure 4.15: Hardware Usage of the middleware software.

The middleware's hardware usage shows a linear increase in CPU utilization for small numbers of slaves. It is possible that with larger numbers of slaves, the CPU utilization linear scale might vary, causing faster or slower changes. However, the change in CPU utilization suggests that the supernode will need to have sufficient hardware capabilities to manage larger numbers of slaves without overloading the physical hardware. The memory usage appears to be constant for small numbers of slaves, and while it's possible that it could change with more slaves, it suggests that the CPU usage will be the limiting factor for scalability of supernodes.

Overall, the middleware plugin does not appear to be hardware intensive for slaves, or for supernodes hosting small numbers of slaves. The results show that the middleware is lightweight and does not affect the performance of the game. Further tests, at a much larger scale, should be done to determine how the server is affected by the hybrid system.

## 4.5 Summary

In this chapter, three performance metrics were measured and compared between the baseline client/server system and the hybrid system: bandwidth, latency and hardware impact.

The bandwidth tests revealed that the hybrid system with no modifications to the server could not provide the desired reductions. However, proposed modifications were presented that would effectively reduce the bandwidth of the server, at the cost of minor increases to the slaves and a significant increase to the supernodes.

The latency tests revealed how significantly distributed latencies can be reduced between nodes located nearby to each other geographically. This offers an improvement in the gameplay experience to players. The non-distributed latencies suffer from poor supernode placement. However, similar proposed modifications to the server can eliminate the negative effects while maintaining the improvement in distributed latencies.

The hardware impact tests revealed that the current hybrid system has no significant effect on the server software for small numbers of players. The middleware software has roughly constant memory usage, and the CPU utilization scales linearly with the number of slaves connected to it. Further tests should be performed with greater numbers of players to establish more effects on hardware usage.

# Chapter 5

## Simulation

### 5.1 Introduction

The purpose of the simulations are for verification of the trends that were obtained and observed from the practical system. Two simulated models have been built in OMNeT++, one representing the Client/Server system, and one for the hybrid system.

OMNeT++ is a discrete event simulator, used for the simulation of network models. Its website describes it as “an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators”. The open-source platform is free for academic purposes and has a wide variety of uses [57].

This chapter discusses the requirements of the simulation models; the basic components of the OMNeT++ platform; and the design of the modelled nodes, the traffic generation and the model configurations. Following this, the simulations are presented with regard to their motivation, configuration and their results. Finally, the chapter is concluded with a summary of the simulation outcomes.

### 5.2 Requirements

The simulation models are based upon the practical system, and therefore must follow the architectural design of the practical system as closely as possible. The bandwidth generated by each node in the model should be similar to the bandwidth generated by the real world nodes for a server, client, slave and supernode. However, as not all the data generated by the nodes in the real system is necessary to display the difference in performances of the systems, only the relevant mechanics should

be included in the models. The model should be measurable with respect to the different network metrics as they are detailed in Section 4.1.1.

## 5.3 Platform

Understanding the basic building blocks of the platform aids in the description of the design. OMNeT++ has four basic building blocks for network simulations: `modules`, `channels`, `gates` and `packets`. Using these components, a model of a network can be built and simulated accordingly. Each component can be extended with user-defined functionality to act as necessary in the simulation.

A `packet` is an object that holds data and has an associated size. `Packets` are able to travel on `channels`, the communication medium of OMNeT++. `Channels` can be defined to have specific properties, such as a data rate or delay. A `channel` can be connected between any two `gates`, allowing `packets` to travel from one `gate` and arrive at another. `Gates` belong to `modules` which are blank slates equipped with basic functionality for sending and receiving `packets` through `gates`. `Modules` can also be compounded to allow multiple sub-`modules` to be combined into a more complex `module`.

These components are only the very base of OMNeT++'s capabilities, but are customizable and versatile enough to create complex systems.

## 5.4 Design

### 5.4.1 Node

The simulation is made up of a number of interconnected nodes modelled to perform like the real-world system. Each node is a compound module made up of 3 sub-module types: buffer, router and processor. Put together, they provide the node with the necessary functionality to generate the correct traffic and direct it to the correct place in the network. An example of the node model is shown in Fig. 5.1. Every node in the system has an address, akin to an IP address, and packets are sent by address to their destinations.

The networking capabilities of each node are implemented by a combination of buffers and a router. The buffers are the basic component for communications, receiving a packet on the incoming channel and holding it in a queue until the outgoing channel is free. Once it detects the channel is free, it sends the packet.

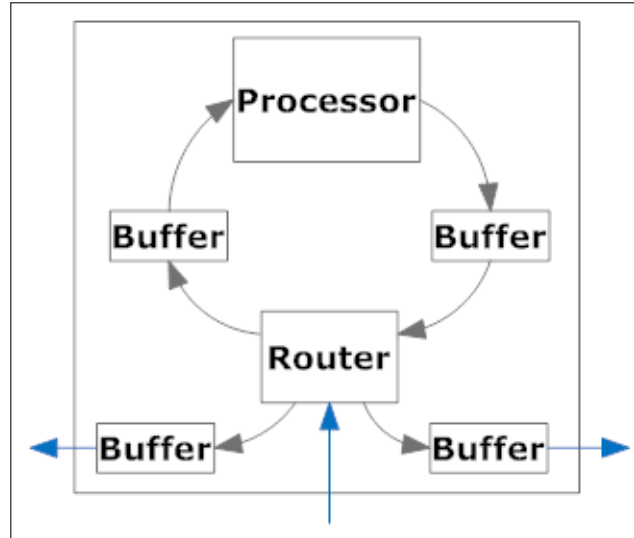


Figure 5.1: Simulation Node Model

The router functions similarly to a real-world router, storing a record of which communication channel is associated with which node address. When the router receives a packet from the processor, it forwards it along the correct channel to an output buffer, which sends it when it is able to. Similarly, if the router receives a packet from the outside destined for the node it is part of, it forwards the packet to the processor.

The communication channels running between the router and processor allow the networking capacity to be defined for the node, namely the upstream and downstream bandwidth, and any base delay on the connection. Individual delays between specific nodes are defined on the external connections between their routers.

The processor holds the node logic, allowing it to act as a server, client, slave, or supernode in the simulation. The processor acts as a backbone for deciding which packet to generate and who it must be sent to. Extensive logic is added to the processor module to make it function similarly to the real-world system. Upon receiving a packet, it processes it in a similar fashion to the PacketHandlers described in the software design, where custom handlers are defined to process the different mechanics being simulated.

### 5.4.2 Traffic

The traffic generation performed by the processor is modelled as an approximation of the physical system. As the purpose of the simulation is to confirm what is



measured in the practical system, only specific cases of traffic are implemented in the simulation: a distributed game mechanic, a non-distributed game mechanic, and interest management traffic. Each of these cases have a number of parameters associated with them, which can be altered to tune the packet sizes of the simulation.

Simulating the traffic of events and updates requires that the virtual world be modelled, and virtual areas of interest be defined for different nodes. This leads to the definition of a *virtual region* and a *virtual area*. The virtual regions represent the hexagonal regions in the real system. Each virtual region contains a number of virtual areas which approximate the effects of areas of interest in the real world game. As nodes are initialised, they are placed in an area which might be shared with other nodes. Nodes in the same area are considered to be within each others areas of interest. Three categories of areas exist: light, medium and dense. The number of areas per type and the probability of selecting an area of a certain type are defined as simulation parameters. When an event originates from a node in an area, all nodes in that area receive the update to it. Functionality is built onto this to allow nodes to change regions and areas as required.

The actions taken by a node are decided by a state machine, shown in Fig. 5.2. The state machine continues indefinitely throughout the simulation, beginning in the decide action state. A probability is assigned as a parameter for a large move, a small move and a speech action. Once decided, the next state generates the traffic that would be caused by the action, and waits the appropriate amount of time before returning back to the decision state. In the case of a large move, another check is performed to decide whether a migration occurs, also defined by a parameter. In the migration state, one final check is performed to decide whether an area or region migration occurs, and the appropriate traffic is generated and actions executed. Additionally, the state machine is extended for a supernode to produce periodic updates for the server every 3 simulated seconds, to send a summarised neighbour update every 5 seconds, and to send a full neighbour update every 30 seconds.

### 5.4.3 Configurations

Using the designed components, two models were implemented in different configurations. In both configurations, one node is designated as the server and maintains that role throughout. Examples of both models are shown in Fig. 5.3.

The first configuration models the client/server system where each node, other than the server, acts as a client and is connected by a channel directly to the server. Each client generates traffic according to the state machine which it follows. On

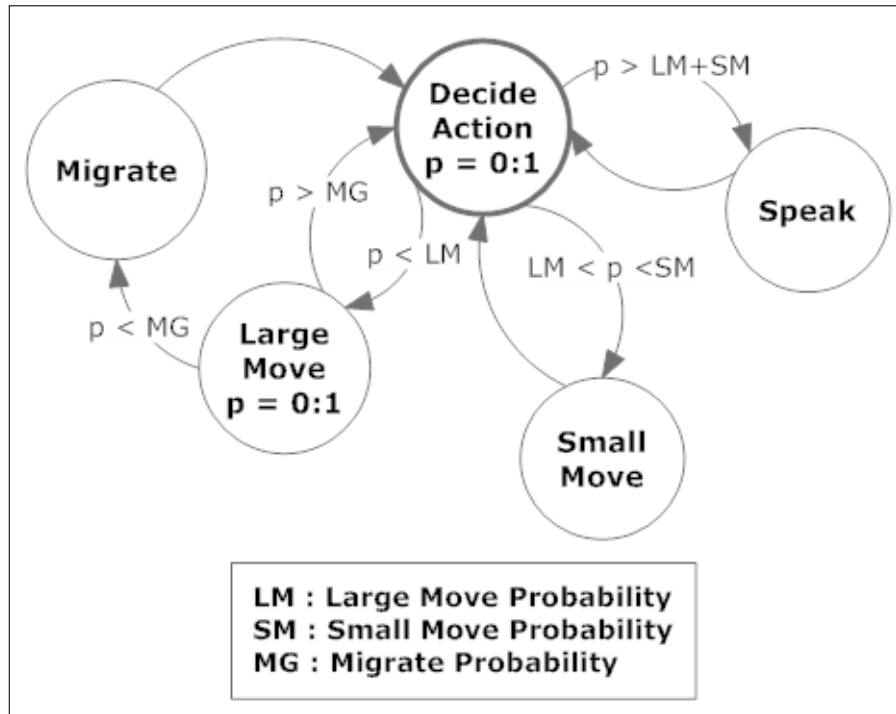


Figure 5.2: Simulated Node AI State Machine

startup, the server initialises the virtual regions and areas, and clients are able to migrate between them as the simulation runs. Migrations do not cause any changes in the communication channels, but clients only receive updates from the server pertaining to other clients in their virtual area.

The second configuration models the hybrid system, where each node, other than the server, can act as either a supernode or a slave. Supernodes connect directly to the server, while slaves route their connections via the supernode. Again, each supernode and slave generates traffic according to the state machine it follows. On startup, the server initialises the virtual regions and areas, and nodes are able to migrate between them as the simulation runs. The same 4 cases of migrations as in the practical system can occur in the simulation, and the nodes react accordingly. The supernode sends AOI instructions to the slaves when they share a virtual area, and the slaves adjust their connectivity. From this, updates for a slave's distributed mechanics are communicated directly with other nodes in the slaves AOI. A slave's non-distributed traffic is communicated to the server via the supernode, and the updates are returned similarly.

Each of these configurations are implemented to resemble the real C/S and hybrid systems. Both models have been used in the simulations performed in sec-

tions 5.5 and 5.6.

## 5.5 Simulation: Bandwidth

### 5.5.1 Motivation

The simulation serves as a verification of the correctness in the trends observed in the practical testing. The first simulation models the baseline C/S system similar to Test 1.1. Only the hybrid system without inter-region interest management, in its single region configuration, is simulated since the inter-region IM results are highly dependant on the model of player activity.

### 5.5.2 Configuration

The models implemented in OMNeT++ are used for this simulation. In the single region configuration of the hybrid system, Node 14 is selected as the supernode. Both models are simulated for 2 hours.

### 5.5.3 Results & Discussion

The simulated bandwidth results were processed in the same fashion as the practical tests. A comparison between the clients in the simulated C/S system, and the slaves in the simulated hybrid system are shown in Fig. 5.4. Similar changes to those seen in the practical system are visible. The slave experiences a reduction of 68% in received movement traffic as it processes its own events and no longer receives movement acknowledgements from the server. This also leads to the increase in movement traffic transmitted by the slave of 48% as it has to update the supernode of every move, as well as any other nodes in its AOI. As expected, the speech traffic shows negligible changes between the systems. In addition, the slaves see a small increase in the miscellaneous traffic received due to the IM overhead transmitted to it by the supernode. Each of these trends match those observed in the practical testing.

The comparison in simulated bandwidth between the servers of the C/S and hybrid systems is shown in Fig. 5.5. This figure illustrates the negligible change in speech traffic between systems, but highlights the effects of distributing movement traffic. The significant increase of 277% in movement traffic received in the hybrid system is a result of the periodic updating by the supernode using the large text

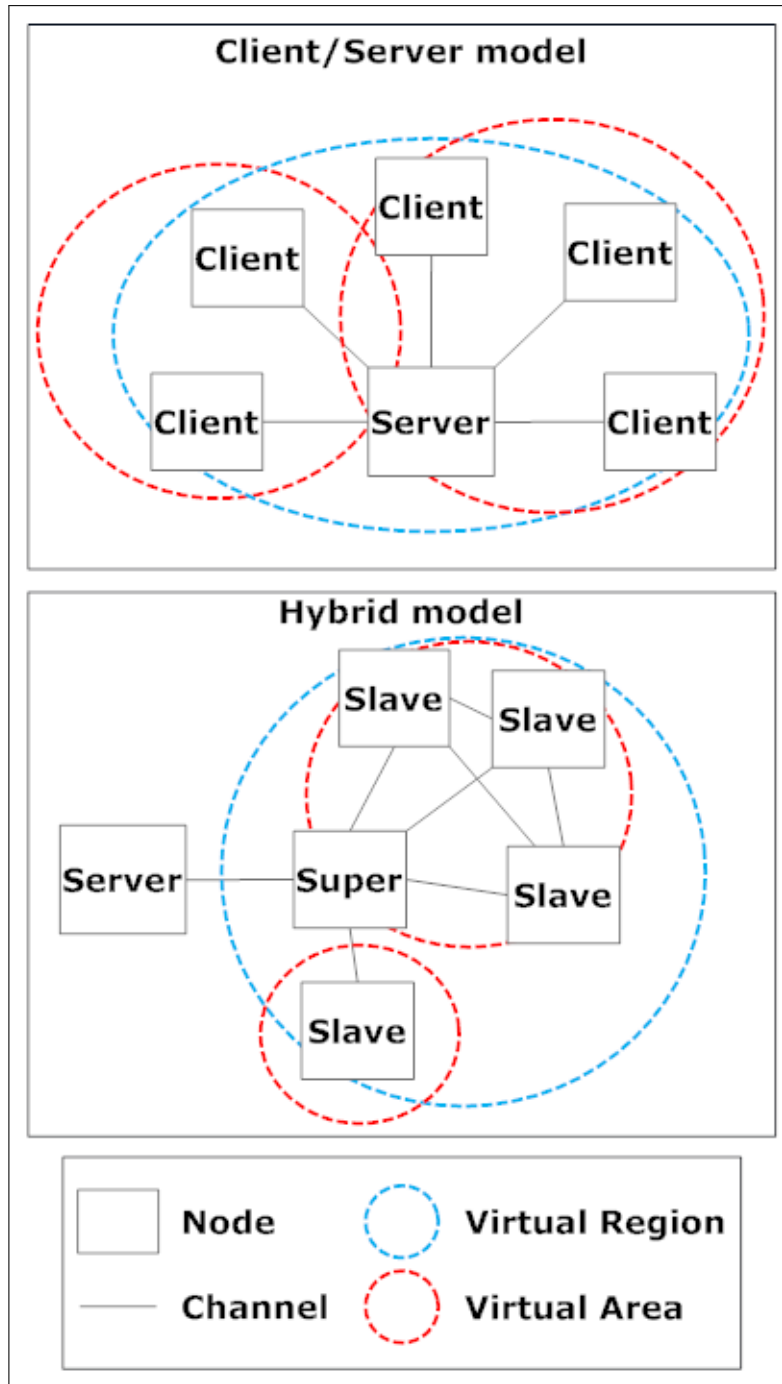


Figure 5.3: Simulation Node Model

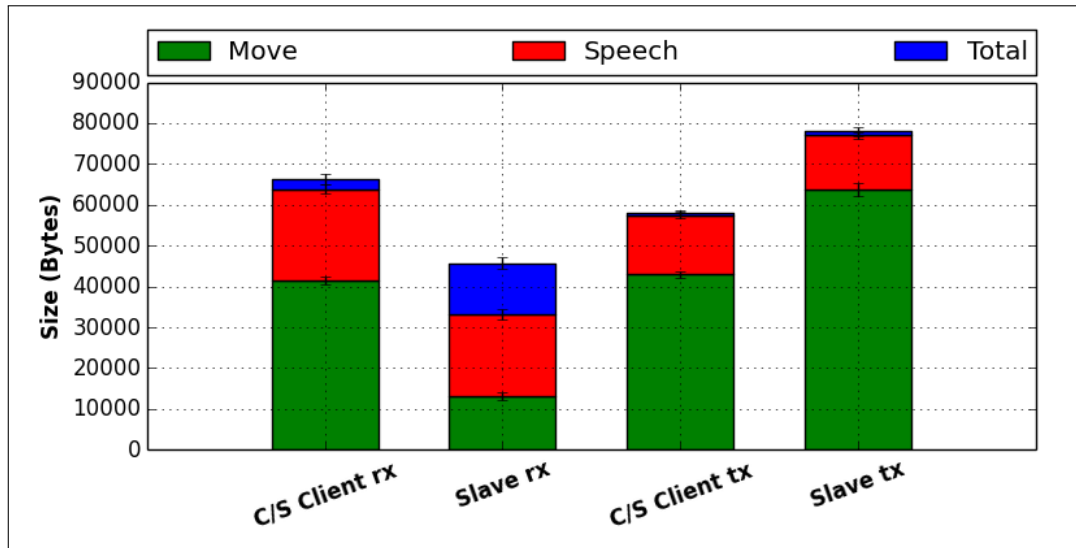


Figure 5.4: Bandwidth comparison between simulated clients and slaves

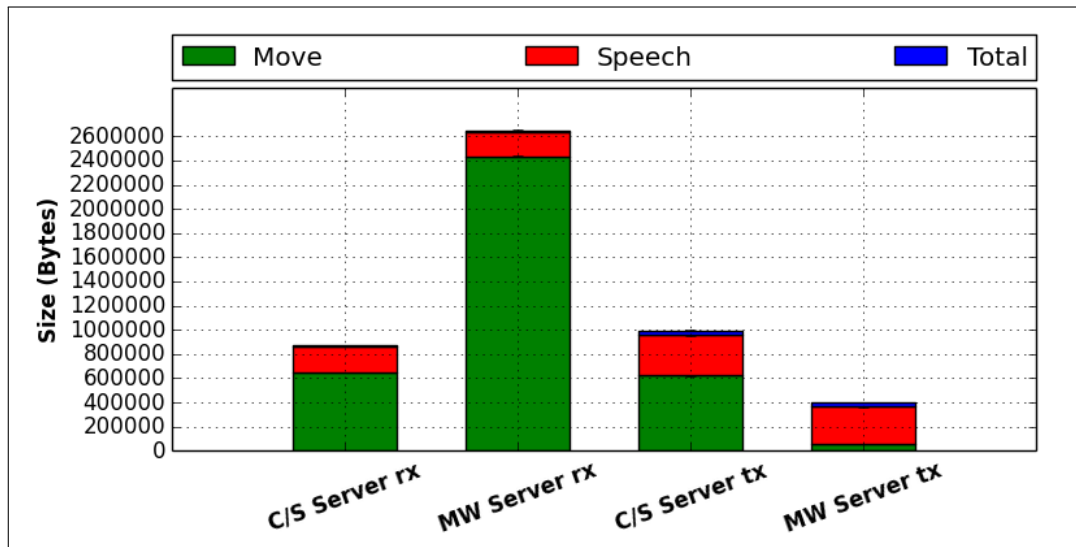


Figure 5.5: Bandwidth comparison between servers in the C/S and hybrid systems

command. As the server is only receiving the movement data periodically, the server produces fewer movement updates than in the C/S system. Therefore, the server experiences a decrease in the movement traffic it transmits of 90%. Overall, the simulated hybrid system's server experiences a 64% increase in bandwidth over its C/S system counterpart due to the periodic movement updates. These trends agree with those observed in the practical testing.

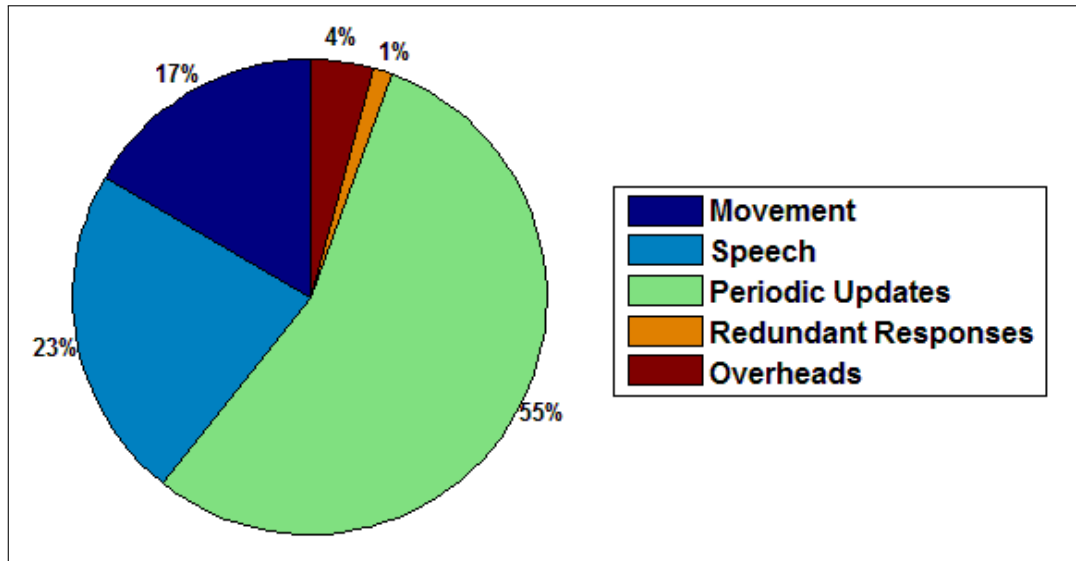


Figure 5.6: Simulated Supernode Bandwidth Distribution

The supernode's bandwidth distribution is shown in Fig. 5.6. The supernode experiences an increase of 3457% in its bandwidth over a client in the C/S system. The increase in movement traffic is attributed to receiving all the slaves' updates, while the decrease results from only updating nodes in the supernode's AOI. Similar to the practical system, the majority of supernode bandwidth (55% of it) is spent on the expensive periodic updates. From this, a similar predication can be made that reducing the periodic update packet size and removing the redundant updates will improve the hybrid system performance. Furthermore, the supernode's speech bandwidth can be reduced by allowing the slaves to connect directly with the server. The bandwidth observed on the supernode is similar to that of the practical system, and the same predicted modifications to the server should improve performance.

The full results of simulated clients and server in the C/S system; and slaves, supernode and server in the hybrid system are shown in Appendix F. The magnitude of values and the their associated changes are not strictly accurate to the practical system due to the approximation of the virtual environment in the simulation. However, the trends in the effects of the hybrid system are consistent with those observed in the practical tests, and therefore serve to verify the general effects of the hybrid system on bandwidth.

## 5.6 Simulation: Latency

### 5.6.1 Motivation

The simulation serves as a verification of the correctness in the trends observed in the practical testing. The first simulation models the baseline C/S system similar to Test 2.1. Using the model of the hybrid system, a single region is simulated with a similar number of nodes located at different distances from the server, and a supernode located somewhere between the two extremes. This will highlight the effects of the hybrid system on both distributed and non-distributed game mechanics.

### 5.6.2 Configuration

The models implemented in OMNeT++ are used for this simulation. The simulated nodes are situated a distance away from each other, linearly proportional to the difference in their node ID numbers. The server is considered to have an ID of 0, therefore Node 1 is nearest the server, and Node 15 is the furthest. In the hybrid system, Node 14 is selected as the supernode for the duration of the test. Both models are simulated for 2 hours.

### 5.6.3 Results & Discussion

The speech and movement data for each node was processed according to the same method as the practical system. The average latencies of the nodes located nearest (Node 1) and furthest (Node 15) from the server are shown in Fig. 5.7.

The speech latencies of Node 1 are shown in Fig. 5.7a for when Node 1 generates an event and the update is perceived by the listed node. The trends observed in this figure are similar to what was experienced by the US West nodes in the practical system where the supernode was located further from the server than themselves. In the C/S system, nodes closer to the server perceive Node 1's actions sooner than those located further from the server. In the hybrid system, where Node 1 has to route its non-distributed traffic through Node 14, the speech latencies increase significantly. This is attributed to the additional latency introduced by the extra hop. In the worst case, where the Node 1's action is perceived via the supernode by Node 2 which is located near the server, the latency results in a 3900% increase over the C/S system.

The speech latencies of Node 15, shown in Fig. 5.7b, show similar trends to those experienced by the practical nodes located in Ireland. As Node 15 is located near

to the supernode, it does not experience significant changes in latencies to nearby nodes, such as between itself and Node 13. However, for nodes located near to the server, the latencies experience an increase compared to the C/S system. Node 1 only perceives Node 15's actions around 400ms later than in the C/S system, an increase of around 190%.

The non-distributed latencies in the simulated system show similar trends to those observed in the practical system. In the hybrid system, nodes located nearby the server are impacted negatively by supernodes located further from the server than themselves, while nodes located further away from the server are impacted to a lesser extent depending on their proximity to the supernode, but still produce negative results.

The distributed movement latencies of Node 1 are shown in Fig 5.7c. The changes in latencies between the C/S and hybrid system are minimal, as they are only reduced by the additional hop between Node 1 and the server, which is negligible due to the proximity of Node 1 to the server. This trend is similar to the one experienced by the US West nodes in the practical system.

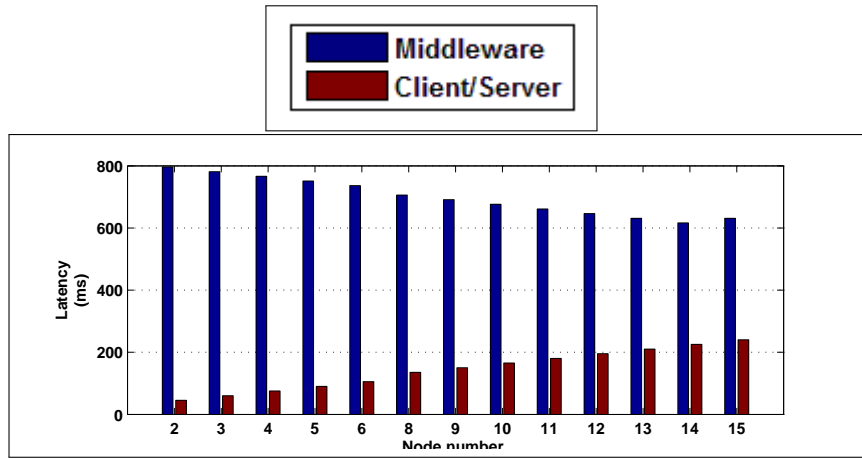
The movement latencies of Node 15 are shown in Fig. 5.7d. These trends are similar to those experienced by the practical Irish nodes. The change in latency between Node 15 and Node 1 is negligible as only the hop between the server and Node 1 is avoided. However, as the nodes move progressively closer to Node 15, the latencies continue to decrease. The biggest benefit is seen between Node 15 and Node 14, where they experience a reduction of 2100% in distributed latency.

The distributed latencies in the simulated system also show similar trends to those observed in the practical system. In all cases, the distributed latencies are improved, especially to a large extent between nodes that are located far away from the server, but near each other. Therefore, the outcomes of the simulated latency tests verify what was observed in the practical latency tests.

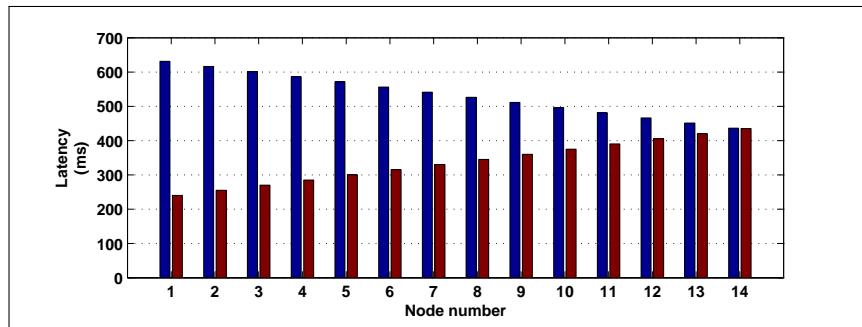
## 5.7 Conclusion

In this chapter, the simulation platform was examined, the design of the simulation was discussed, and the simulations themselves presented. Two tests were performed to examine the bandwidth and latency of the hybrid system in comparison to a C/S baseline. The results showed the model of the virtual environment to not be accurate to the practical implementation, but still capable of revealing the effects of

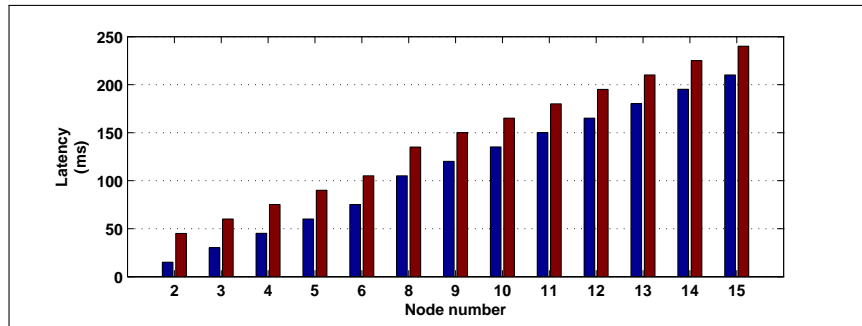




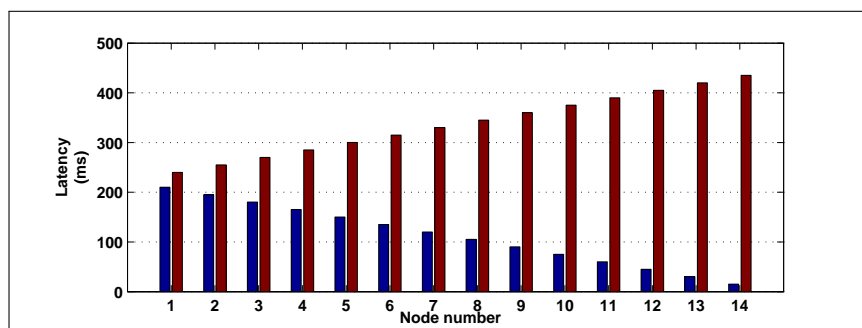
(a) Node 1: comparison of speech latency between C/S and hybrid system.



(b) Node 15: comparison of speech latency between C/S and hybrid system.



(c) Node 1: comparison of movement latency between C/S and hybrid system.



(d) Node 15: comparison of movement latency between C/S and hybrid system.

Figure 5.7: Simulated Latency Results

the hybrid system.

In the bandwidth simulation, the effects of the simulated hybrid system were similar to those observed in the practical tests. The non-distributed traffic was unchanged for both the server and slaves, while the supernode saw a significant increase due to the connection rerouting. Slaves experienced a reduction in distributed traffic they received, but an increase in distributed traffic they transmitted. The server experienced an increase in bandwidth due to the large periodic updates, but transmitted fewer updates than in the C/S system. The supernode experienced an increase in bandwidth resulting from distributed mechanics and periodic updating. While the model of the virtual environment could not produce perfect results, the trends observed in the simulation verified those observed in the practical tests.

The latency simulation revealed similar trends to those observed in the practical tests. The effects of the hybrid system on distributed latencies were verified to be positive in all regards, reducing movement latencies between all nodes, and significantly between nodes located nearby to each other, but distant from the server. The effects of the hybrid system on non-distributed mechanics were verified to be mostly negative, where the additional hops between (1) the node and the supernode, and (2) the supernode and the server each negatively impacted the nodes. This is significant in cases where slaves are nearer to the server than the supernode.

Overall, the outcomes of the simulations resulted in the trends observed in the practical results to be verified as accurate.

# Chapter 6

## Conclusion & Recommendations

### 6.1 Introduction

In an effort to promote commercial P2P adoption, this thesis introduced the concept of a hybrid peer-to-peer middleware plugin for an unmodified, existing C/S MMOG as a solution to some of the shortcomings of C/S and pure P2P architectures. With the groundwork laid through an examination of MMOGs, existing architectures, and P2P concepts, the designs of the hybrid system and middleware plugin were presented. The designed system was implemented and tested in a practical setup, rather than in the traditional theoretical or simulated fashion, and results of its performance were discussed.

This chapter provides a summary of the performance results, concludes on the feasibility of the system, compares the results to related systems, and lists opportunities for future work.

### 6.2 System Performance

Testing this specific system's performance produced a combination of both desirable and undesirable results. The bandwidth in the hybrid system showed an acceptable increase for slaves of only 17% in the single region configuration. The supernode showed an unacceptably large increase of 3 111%, and the server showed an increase of 35% instead of the desired decrease. Inter-region bandwidths showed that migrations are expensive and increased the servers load by an additional 45%, while the overhead for interest management did not contribute significantly towards total bandwidth experienced by slaves or supernodes. From these results, predictions with

minor modifications to the server were made which showed that the server could have its bandwidth reduced by 39%, while the supernode experiences a smaller increase of 1 995%. The potential results were shown in Table 4.4 of Chapter 4.

The latency results showed improvements in the distributed mechanics for all setups, offering up to an 88.3% decrease in movement latencies. The resulting latencies of non-distributed speech traffic varied according to the placement of the supernode, but in the worst case increased latencies almost 6000%. While the decrease in distributed latencies could significantly improve the players gameplay experience, the increase in non-distributed latencies could detract from that. However, predictions were made that with modifications to the server, the negative latency results could be eliminated entirely, resulting in a 0% change in non-distributed latencies and maintaining the decreases in distributed mechanics. The best case latencies were shown in Fig. 4.13 of Chapter 4

The hardware results revealed no noticeable change in the hardware impact of the hybrid system on the server's CPU utilization or memory usage for small numbers of players. Investigation into the middleware plugin itself predicted that CPU utilization, not memory usage, would become the limiting factor as a supernode hosts more slaves. The middleware, when operating in a slave capacity, was shown to have a relatively constant low utilization of resources, thus it would not starve the game client.

### 6.3 Feasibility Outcomes

Due to the negative results, it can be concluded that a middleware plugin with *no modification* of the server or client software is not a viable solution as it does not achieve the desired results of reducing server load while improving player latencies. The negative results can be attributed to two constraints of the server:

1. The server only accepts events from the same client that is identified during the login process. This means another peer in the network cannot send an update on behalf of another player. The constraint resulted in the design decision to reroute all server bound client traffic through the supernode, significantly increasing the bandwidth that has to be handled by a supernode. Routing server-bound traffic through a supernode also significantly increased the latency experienced by players, especially if the supernode's latency to the server was large.
2. To maintain consistency between the server state and the state stored in the

P2P segment, the supernode needs to periodically send aggregated movement updates to the server. The server protocol does not by default allow for such aggregated updates, therefore a custom player command script packet was used for sending aggregated updates. The size of the custom packet is roughly 5 times larger than normal movement packets, resulting in the number of movement bytes received by the server in the hybrid system to be significantly more as compared to the number of movement bytes received by the server in the C/S system.

However, with two primary modifications performed on the server, it is expected that the hybrid system will be able to achieve the expected performance increases:

1. Introducing a packet natively into the server code to provide a means for the supernode to periodically update the server will reduce the current bandwidth associated with transmitting periodic updates by almost 80%. Furthermore, removing the redundant updates being transmitted from the server in response to the periodic updates will reduce the servers outgoing movement traffic by 100% and the supernodes traffic by 7% of its current value. These modifications alone will result in a 39% decrease in server bandwidth from the C/S system, and the supernode experiencing a smaller increase of 1 995%.
2. Introducing a reduced login sequence, specifically for migrations, would reduce the large bandwidth spikes occurring during migrations. However, by modifying the server to accept events and updates from any peer in the network, the supernode routing would no longer be required. Doing this would almost entirely eliminate both the migration overhead, and the increase in latency for non-distributed packets. In addition, the supernode would no longer have to manage non-distributed traffic for its slaves. Combining this with the other primary modification, a supernode would experience an increase of only 1 296% over a C/S client.

Care needs to be taken to ensure the security of the architecture not be compromised. This could be achieved by using some form of encryption on the client events so that the server can uniquely identify the origin of the client events and updates.

Therefore, with minor server modification the expected performance increase can be achieved and the middleware presents itself as a feasible solution to reduce the risk of P2P adoption for existing systems by introducing the benefits of P2P networks without the requirement of fully re-engineering an established system.

## 6.4 Related Work

A 2003 paper by Pellegrino and Dovrolis discusses the effects on bandwidth and latency by a custom hybrid model that resembles the middleware-plugged hybrid system, but does not function as a plugin [58]. They find that player latencies are reduced due to the direct communication between nodes. The bandwidth requirements of the server are reduced in the hybrid system where they increase linearly with the number of players, rather than the quadratic increase in the C/S system. However, the server's CPU and memory utilization remain unchanged between C/S and hybrid systems. These results agree with those of the middleware-plugged hybrid system.

*Kiwano*, the system which modifies a C/S architecture by distributing movement amongst multiple servers, uses a proxy application similar to the middleware software in this system. In their tests, they found that their proxy slightly reduced the bandwidth of a player over the standard C/S system, while increasing the CPU and memory utilization. Overall, their full system experienced a net increase in bandwidth usage, but the additional load is managed by additional independent nodes at the expense of the game's host [59]. Therefore, the scalability offered by their system comes at additional financial cost to the host, unlike the middleware-plugged hybrid system which makes use of existing resources offered by the peers.

## 6.5 Future Work

A number of possibilities exist for future work from this point:

- The hybrid system provides a platform for experimentation of other P2P aspects. It can be used to implement and test: cheating mitigation techniques; supernode selection algorithms; different P2P overlays by replacing the directory server; or different interest management schemes such as introducing dynamic regions.
- The middleware plugin as it stands can be extended to distribute more game mechanics. Once distributed, the different mechanics can be analysed individually to gauge whether different mechanics produce different effects on the performance of the system.
- The hybrid system can be applied to different existing MMOGs. Analysis can be done into if and how the different systems are influenced by the middleware.

Building on this, game agnostic mechanics can be identified, and a generic way of handling game agnostic components can be implemented.

- The current middleware plugin has no recovery mechanisms, thus it can be enhanced to include fault tolerance against network errors and dropped nodes.
- Experimentation is required to measure at what point a supernode will become overloaded and how it will affect the hybrid system. From this, how to balance and manage supernode load should be investigated to aid the system's scalability and robustness.
- The performed tests can be enhanced by using legitimate player traces rather than an approximation of player behaviour. Similarly, the tests can be performed at much larger scale with hundreds of players to better understand the full effects of the hybrid system.
- The simulation model can be improved to more accurately model the virtual environment, its regions and the players' areas of interest.
- A practical experiment can be performed to verify that the proposed modifications result in the predicted results.
- If server modifications are allowed, a study can be performed into how extensively a server must be modified to produce improved results. Finding an optimal balance between modification and performance will allow for middleware-type solutions to be better tailored to meeting those requirements.
- The middleware plugin principle shown to be feasible in this thesis can be applied to non-MMOG client/server systems as well. Identification of systems suited to this transformation and implementing a middleware plugin for them could provide similar benefits.

# Bibliography

- [1] Yahyavi, A. and Kemme, B.: Peer-to-peer architectures for massively multiplayer online games. *ACM Computing Surveys*, vol. 46, no. 1, pp. 1–51, October 2013. ISSN 03600300.
- [2] Schiele, G., Suselbeck, R., Wacker, A., Hahner, J., Becker, C. and Weis, T.: Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. In: *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pp. 773–782. IEEE, May 2007. ISBN 0-7695-2833-3.
- [3] Richard Bartle: MUD Hitsory, Who Invented MUD's, How MUD's Were Invented. [Online] Available: [http://www.livinginternet.com/d/di\\_major.htm](http://www.livinginternet.com/d/di_major.htm), March 2012.
- [4] The Original Neverwinter Nights 1991 - 1997. [Online] Available: <http://www.bladekeep.com/nwn/index2.htm>, March 2012.
- [5] Achterbosch, L., Pierce, R. and Simmons, G.: Massively multiplayer online role-playing games. *Computers in Entertainment*, vol. 5, no. 4, p. 1, March 2008. ISSN 15443574.
- [6] MMOData.net: Subscriptions and Active Accounts with a peak above 1m. [Online] Available: <http://users.telenet.be/mmodata/Charts/Subs-1.png>, March 2012.
- [7] MMOData.net: Peak Concurrent Users with a Peak above 500k. [Online] Available: <http://users.telenet.be/mmodata/Charts/PCU-1.png>, March 2012.
- [8] Jon Radoff: Anatomy of an MMORPG. *PlayerVox*, March 2007.
- [9] Raph Koster: On “Pay To Play” Or, MMORPG Business Models 101. [Online] Available: <http://www.raphkoster.com/gaming/busmodels.shtml>, March 2012.



- [10] Daniel Reynolds: The Cost to Make a Quality MMORPG. [Online] Available: <http://www.whatmmorpg.com/cost-to-make-a-quality-mmorpg.php>, June 2010.
- [11] Pantel, L. and Wolf, L.C.: On the impact of delay on real-time multiplayer games. In: *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video - NOSSDAV '02*, p. 23. ACM Press, New York, New York, USA, 2002. ISBN 1581135122.
- [12] Fritsch, T., Ritter, H. and Schiller, J.: The effect of latency and network limitations on MMORPGs. In: *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games - NetGames '05*, p. 1. ACM Press, New York, New York, USA, 2005. ISBN 1595931562.
- [13] Chen, K.-t., Huang, P., Huang, C.-y. and Lei, C.-l.: Game traffic analysis: an MMORPG perspective. In: *Proceedings of the international workshop on Network and operating systems support for digital audio and video - NOSSDAV '05*, p. 19. ACM Press, New York, New York, USA, 2005. ISBN 158113987X.
- [14] Kinicki, J. and Claypool, M.: Traffic analysis of avatars in Second Life. In: *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video - NOSSDAV '08*, p. 69. ACM Press, New York, New York, USA, 2008. ISBN 9781605581576.
- [15] Miller, J.L. and Crowcroft, J.: The near-term feasibility of P2P MMOG's. In: *2010 9th Annual Workshop on Network and Systems Support for Games*, vol. 1, pp. 1–6. IEEE, November 2010. ISBN 978-1-4244-8356-3.
- [16] Jeffrey Kesselman: Server Architecture for Massively Multiplayer Online Games. in *JavaOne conference - Session TS-1351*, 2004.
- [17] Nae, V., Prodan, R., Iosup, A. and Fahringer, T.: A new business model for massively multiplayer online games. In: *Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering - ICPE '11*, p. 271. ACM Press, New York, New York, USA, 2011. ISBN 9781450305198.
- [18] BigWorld Technology: BigWorld. [Online] Available: <http://www.bigworldtech.com>, October 2013.

- [19] Diaconu, R. and Keller, J.: Kiwano: A Scalable Distributed Infrastructure for Virtual Worlds. [Online] Available: <http://download.hybridearth.net/kiwano-hpcs2013.pdf>, October 2013.
- [20] Doval, D. and O'Mahony, D.: Overlay networks a scalable alternative for p2p. *IEEE Internet Computing*, vol. 7, no. 4, pp. 79–82, July 2003. ISSN 1089-7801.
- [21] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H.: Chord. *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, October 2001. ISSN 01464833.
- [22] Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Schenker, S.: A scalable content-addressable network. *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 161–172, October 2001. ISSN 01464833.
- [23] Zhao, B., Kubiatowicz, J. and Joseph, A.: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Tech. Rep. April, University of California, 2001. CSD-01-1141.
- [24] Rowstron, A.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Middleware 2001*, , no. November 2001, 2001.
- [25] Castro, M., Druschel, P., Kermarrec, A.-M. and Rowstron, A.: Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, October 2002. ISSN 0733-8716.
- [26] Druschel, P. and Rowstron, A.: PAST: a large-scale, persistent peer-to-peer storage utility. In: *Proceedings Eighth Workshop on Hot Topics in Operating Systems*, pp. 75–80. IEEE Comput. Soc, 2001. ISBN 0-7695-1040-X.
- [27] Knutsson, B. and Hopkins, B.: Peer-to-peer support for massively multiplayer games. In: *IEEE INFOCOM 2004*, vol. 1, pp. 96–107. IEEE, 2004. ISBN 0-7803-8355-9.
- [28] Hampel, T., Bopp, T. and Hinn, R.: A peer-to-peer architecture for massive multiplayer online games. In: *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06*, pp. 48–es. ACM Press, New York, New York, USA, 2006. ISBN 1595935894.

- [29] Hu, S.-Y. and Liao, G.-M.: Scalable peer-to-peer networked virtual environment. In: *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04 Network and system support for games - SIGCOMM 2004 Workshops*, p. 129. ACM Press, New York, New York, USA, 2004. ISBN 158113942X.
- [30] Chan, L., Yong, J., Bai, J., Leong, B. and Tan, R.: Hydra : A Massively-Multiplayer Peer-to-Peer Architecture for the Game Developer. In: *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games - NetGames '07*, pp. 37–42. ACM Press, New York, New York, USA, 2007. ISBN 9780980446005.
- [31] Chen, A. and Muntz, R.R.: Peer clustering: a hybrid approach to distributed virtual environments. In: *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06*, p. 11. ACM Press, New York, New York, USA, 2006. ISBN 1595935894.
- [32] Buyukkaya, E., Abdallah, M. and Cavagna, R.: VoroGame: A Hybrid P2P Architecture for Massively Multiplayer Games. In: *2009 6th IEEE Consumer Communications and Networking Conference*, pp. 1–5. Ieee, January 2009. ISBN 978-1-4244-2308-8.
- [33] Carter, C., Rhalibi, A.E., Merabti, M. and Bendiab, A.T.: Hybrid Client-Server, Peer-to-Peer framework for MMOG. In: *2010 IEEE International Conference on Multimedia and Expo*, pp. 1558–1563. IEEE, July 2010. ISBN 978-1-4244-7491-2.
- [34] Varvello, M., Diot, C. and Biersack, E.W.: P2P Second Life: Experimental Validation Using Kad. In: *IEEE INFOCOM 2009 - The 28th Conference on Computer Communications*, pp. 1161–1169. IEEE, April 2009. ISBN 978-1-4244-3512-8.
- [35] Gilmore, J.S. and Engelbrecht, H.A.: A Survey of State Persistency in Peer-to-Peer Massively Multiplayer Online Games. *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–19, 2011. ISSN 1045-9219.
- [36] Fan, L.: *Solving Key Design Issues for Massively Multiplayer Online Games on Peer-to-Peer Architectures*. Ph.D. thesis, Heriot-Watt University, 2009.
- [37] Yu, A.P. and Vuong, S.T.: MOPAR. In: *Proceedings of the international workshop on Network and operating systems support for digital audio and video*

- *NOSSDAV '05*, p. 99. ACM Press, New York, New York, USA, 2005. ISBN 158113987X.
- [38] Boulanger, J.-S., Kienzle, J. and Verbrugge, C.: Comparing interest management algorithms for massively multiplayer games. In: *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06*, p. 6. ACM Press, New York, New York, USA, 2006. ISBN 1595935894.
- [39] Bharambe, A., Douceur, J.R., Lorch, J.R., Moscibroda, T., Pang, J., Seshan, S. and Zhuang, X.: Donnybrook. In: *Proceedings of the ACM SIGCOMM 2008 conference on Data communication - SIGCOMM '08*, p. 389. ACM Press, New York, New York, USA, 2008. ISBN 9781605581750.
- [40] Jesi, G., Montresor, A. and Babaoglu, O.: Proximity-aware superpeer overlay topologies. *Self-Managed Networks, Systems, and ...*, pp. 43–57, 2006.
- [41] Fan, L., Taylor, H. and Trinder, P.: Mediator. In: *Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games - NetGames '07*, pp. 43–48. ACM Press, New York, New York, USA, 2007. ISBN 9780980446005.
- [42] Lo, V. and GauthierDickey, C.: Scalable Supernode Selection in Peer-to-Peer Overlay Networks. In: *Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pp. 18–27. IEEE, 2005. ISBN 0-7695-2417-6.
- [43] Mitra, B., Peruani, F., Ghose, S. and Ganguly, N.: Analyzing the vulnerability of superpeer networks against attack. In: *Proceedings of the 14th ACM conference on Computer and communications security - CCS '07*, p. 225. ACM Press, New York, New York, USA, 2007. ISBN 9781595937032.
- [44] Aurenhammer, F.: Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, September 1991. ISSN 03600300.
- [45] Hu, S.-Y., Chang, S.-C. and Jiang, J.-R.: Voronoi State Management for Peer-to-Peer Massively Multiplayer Online Games. *2008 5th IEEE Consumer Communications and Networking Conference*, pp. 1134–1138, 2008.
- [46] Krause, S.: A case for mutual notification. In: *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games - NetGames '08*, p. 28. ACM Press, New York, New York, USA, 2008. ISBN 9781605581323.

- [47] Fiedler, S., Wallner, M. and Weber, M.: A communication architecture for massive multiplayer games. In: *Proceedings of the 1st workshop on Network and system support for games - NETGAMES '02*, pp. 14–22. ACM Press, New York, New York, USA, 2002. ISBN 1581134932.
- [48] Mythic Entertainment: Ultima Online. [Online] Available: <http://www.uo.com>, October 2013.
- [49] The RunUO Team: RunUO. [Online] Available: <http://www.runuo.com/>, March 2012.
- [50] Iris2 Team: Iris2. [Online] Available: <http://www.iris2.de>, October 2013.
- [51] EasyUO Team: EasyUO. [Online] Available: <http://www.easyuo.com>, October 2013.
- [52] Microsoft: Windows Performance Counters: Process Object. [Online] Available: <http://technet.microsoft.com/en-us/library/cc780836%28WS.10%29.aspx>, October 2013.
- [53] Suznjevic, M., Stupar, I. and Matijasevic, M.: MMORPG player behavior model based on player action categories. In: *2011 10th Annual Workshop on Network and Systems Support for Games*, pp. 1–6. IEEE, October 2011. ISBN 978-1-4577-1934-9.
- [54] Kihl, M., Aurelius, A. and Lagerstedt, C.: Analysis of World of Warcraft traffic patterns and user behavior. *International Congress on Ultra Modern Telecommunications and Control Systems*, pp. 218–223, October 2010.
- [55] Oracle: VirtualBox. [Online] Available: <https://www.virtualbox.org>, October 2013.
- [56] Amazon.com: Amazon Web Services. [Online] Available: <http://aws.amazon.com>, October 2013.
- [57] OMNeT++ Community: OMNeT++. [Online] Available: <http://www.omnetpp.org/>, March 2012.
- [58] Pellegrino, J.D. and Dovrolis, C.: Bandwidth requirement and state consistency in three multiplayer game architectures. In: *Proceedings of the 2nd workshop on Network and system support for games - NETGAMES '03*, pp. 52–59. ACM Press, New York, New York, USA, 2003. ISBN 1581137346.

- [59] Diaconu, R., Keller, J. and Valero, M.: Manycraft: Scaling Minecraft to Millions. In: *Proceedings of the 12th workshop on Network and system support for games - NETGAMES '13*. 2013.  
Available at: <http://blog.hybridearth.net/wp-content/uploads/2013/11/netgames2013-1.pdf>
- [60] Thomas Jahn: Coordinates in Hexagon-Based Tile Maps. [Online]  
Available: [http://www.gamedev.net/page/resources/\\_/technical/game-programming/coordinates-in-hexagon-based-tile-maps-r1800](http://www.gamedev.net/page/resources/_/technical/game-programming/coordinates-in-hexagon-based-tile-maps-r1800),  
April 2002.

# Appendix A

## Hexagonal Co-ordinate System

The hexagonal division algorithm is available from [60] and summarised below.

A regular hexagon is shown in fig. A.1 with a side length  $s$ . Given that the hexagon is regular, all other parameters can be calculated in accordance with equations (A.1–A.4). If a known number of identical regular hexagons are placed together to form a grid such as in fig. A.2a, and the size of the grid can be derived from the hexagon parameters. From the hexagonal grid that is created, rows are created in an alternating pattern and repeating rectangles are created in each of the rows, where each rectangle approximates a hexagon. The result of this is shown in fig. A.2b. Henceforth, even rows are known as an  $A$  rows, and odd rows as  $B$  rows. From this, a continuous 2D point can be mapped onto a hexagon. As the 3rd dimension is negligible for achieving the desired functionality from the interest management, the vertical attribute of a player's position is dropped.

First, it must be established which rectangle the 2D point is located in using equations (A.5) and (A.6). With the approximate hexagon found, the inside of the rectangle must be considered as it contains parts of three surrounding hexagons. Using equations (A.7) and (A.8), the internal location of the point relative to the top-left of the rectangle is found. From this, it can be identified how to offset the approximate rectangle co-ordinate to find the exact hexagonal co-ordinate. This is done differently for  $A$  or  $B$  rows respectively.

In an  $A$  row, the majority of the rectangle is filled with the appropriate hexagon, correlating to the correct co-ordinate. In the case where the point is located in the upper-left portion, x- and y-coordinates are both reduced by one. Similarly, if the point is in the top-right section, only the y-coordinate is reduced by one.

In a  $B$  row, the rectangle co-ordinate correlates to the right-side hexagon. If the point is located in that part of the rectangle, the co-ordinates remain unchanged. If the point is located in the left-side section, the x-coordinate is reduced by one.

Finally, if the point is in the top section then the y-coordinate is reduced by one.

All incomplete hexagons located around the borders of the rectangle grid are treated as part of their neighbouring hexagons. These coordinate adjustment algorithms are presented in pseudocode below.

<pre> if(TYPE A RECTANGLE) if( x &gt; r )     if( y &lt; (h/r)x - h )         hex_X = rect_X         hex_Y = rect_Y - 1     else         hex_X = rect_X         hex_Y = rect_Y     else if( y &lt; (h/r)x )     hex_X = rect_X - 1     hex_Y = rect_Y - 1     else         hex_X = rect_X         hex_Y = rect_Y                 </pre>	<pre> if(TYPE B RECTANGLE) if( x &gt; r )     if( y &lt; 2h - (h/r)x )         hex_X = rect_X         hex_Y = rect_Y - 1     else         hex_X = rect_X         hex_Y = rect_Y     else if( y &lt; (h/r)x )     hex_X = rect_X     hex_Y = rect_Y - 1     else         hex_X = rect_X - 1         hex_Y = rect_Y                 </pre>
---	--

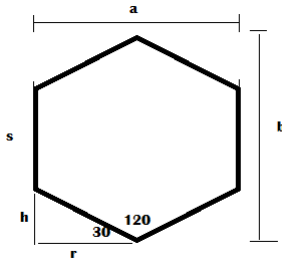


Figure A.1: Hexagon

$$h = \sin(30 \text{ deg}) \times s \quad (\text{A.1})$$

$$r = \cos(30 \text{ deg}) \times s \quad (\text{A.2})$$

$$b = s + 2 \times h \quad (\text{A.3})$$

$$a = 2 \times r \quad (\text{A.4})$$

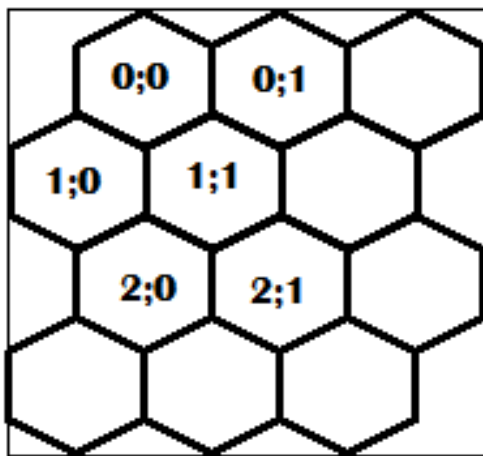
$$x_{rect} = x_{point} \div 2r \quad (\text{A.5})$$

$$y_{rect} = y_{point} \div (h + s) \quad (\text{A.6})$$

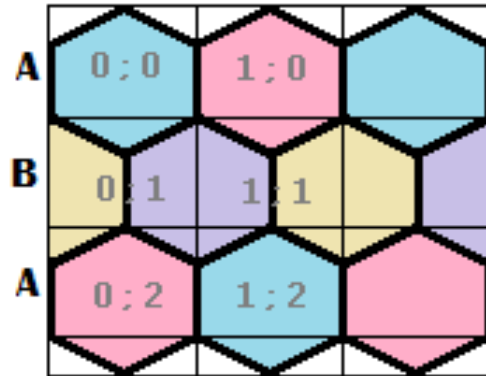
$$x_{local} = x_{point} - x_{rect} \times 2r \quad (\text{A.7})$$

$$y_{local} = y_{point} - y_{rect} \times (h + s) \quad (\text{A.8})$$

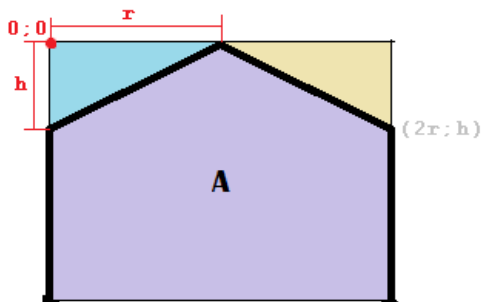




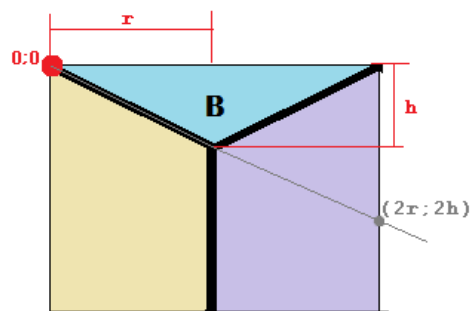
(a) Hexagon Grid



(b) Rectangle Divisions



(c) A-type Rectangle



(d) B-type Rectangle

Figure A.2: Hexagonal Grid Co-ordinate Lookup

# Appendix B

## Test 1.1: C/S Baseline Bandwidth Results

Table B.1: Bandwidth values of clients in the C/S system (B)

<b>Aspect</b>	<b>Mean</b>	<b>Std</b>
Move rx	29 554	8 753
Move tx	49 616	10 557
Move both	79 170	17 888
Speech rx	21 765	1 459
Speech tx	19 320	7 928
Speech both	41 085	7 648
Total rx	73 661	11 286
Total tx	78 103	3 749
Total both	151 764	12 546

Table B.2: Bandwidth totals of server in the C/S system (B)

<b>Aspect</b>	<b>Total</b>	<b>Std</b>
Move rx	729 068	21 678
Move tx	444 314	32 768
Move both	1 173 382	42 836
Speech rx	302 719	22 333
Speech tx	329 031	6 606
Speech both	631 750	25 082
Total rx	1 169 809	34 071
Total tx	1 108 501	34 039
Total both	2 278 310	44 519

# Appendix C

## Test 1.2: Hybrid Single Region Bandwidth Results

Table C.1: Bandwidth values of slaves in the single region hybrid system and change from clients in the C/S system (B)

Aspect	Mean	Std	C/S Mean	Change
Move rx	12 265	10 635	29 554	-58%
Move tx	85 469	11 415	49 616	+72%
Move both	97 733	19 789	79 170	+23%
Speech rx	20 675	1 177	21 765	-5%
Speech tx	16 531	1 222	19 320	-14%
Speech both	37 205	2 195	41 085	-9%
Total rx	59 004	19 357	73 661	-19%
Total tx	118 637	12 883	78 103	+52%
Total both	177 642	29 589	151 764	+17%

Table C.2: Bandwidth values of server in the single region hybrid system and change from server in the C/S system (B)

Aspect	Mean	Std	C/S Mean	Change
Move rx	1 724 977	6 122	729 068	+137%
Move tx	318 052	1 129	444 314	-28%
Move both	2 043 029	7 251	1 173 382	+74%
Speech rx	252 616	5 572	302 719	-16%
Speech tx	315 452	1 428	329 031	+4%
Speech both	568 068	5 861	631 750	-10%
Total rx	2 069 693	8 792	1 169 809	+77%
Total tx	1 012 697	46 752	1 108 501	-9%
Total both	3 082 390	37 960	2 278 310	+35%

# Appendix D

## Measurement Computer Specifications

Table D.1: Measurement Computer Specifications

<b>Component</b>	<b>Specification</b>
CPU	Processor Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz, 3301 Mhz, 4 Core(s), 4 Logical Processor(s)
RAM	8.00GB
Virtual Memory	16.00GB
OS	Windows 7 Professional 64bit
OS Version	6.1.7601 Service Pack 1 Build 7601
Storage	OCZ Vertex 4 256GB Solid State Drive
Display	NVIDIA GeForce GTX 650 Ti

# Appendix E

## Hardware Usage Results

Table E.1: Server without middleware

Slaves	CPU(%)		Working Set(MB)		Private Bytes(MB)	
	Mean	Std	Mean	Std	Mean	Std
1	0.094	0.476	61.2	3.5	59.6	4
2	0.058	0.323	60.6	3.6	59.6	4.4
3	0.034	0.182	60.9	3.3	59.9	3.5
4	0.063	0.409	62	2.8	61.2	3
5	0.085	0.658	62.5	2.5	61.8	2.2
6	0.097	0.296	64	2.9	62.8	1.6
7	0.117	0.397	62.9	2.3	62.6	2.4
8	0.162	0.494	62.6	1.5	62.7	1.4
9	0.18	0.506	61.5	2.7	61.6	2.6
10	0.22	0.476	61.3	3.3	60.9	3.6
11	0.211	0.576	61.7	2.6	61.5	2.7
12	0.119	0.324	61.5	2.9	61.4	2.6
13	0.177	0.446	61.7	2.5	61.8	2.4
14	0.162	0.5	62.3	2.3	61.6	2.5
15	0.207	0.479	62.5	1.6	60.7	1

Table E.2: Server with middleware

Slaves	CPU(%)		Working Set(MB)		Private Bytes(MB)	
	Mean	Std	Mean	Std	Mean	Std
1	0.123	0.609	58.8	2.5	59.3	2.5
2	0.08	0.44	58	2.1	59.1	2
3	0.041	0.198	57.7	1.8	59.1	1.9
4	0.06	0.238	58.1	1.5	59.9	1.2
5	0.06	0.329	58.9	2.2	60.3	0.9
6	0.242	1.002	59	1.3	61.1	0.9
7	0.095	0.461	60	2.6	60.6	1.1
8	0.068	0.252	59.1	2.6	61.2	2.4
9	0.111	0.314	58	0.3	60.3	0.6
10	0.139	0.467	57.8	0.6	59.9	0.9
11	0.143	0.35	58	0.3	60.3	0.4
12	0.226	0.705	58	0.3	60.2	0.4
13	0.229	0.577	58.5	1.1	60.9	1.1
14	0.193	0.508	58	0.5	60.3	0.7
15	0.226	0.593	58.4	1.5	60.4	0.5

Table E.3: Middleware software

Slaves	CPU(%)		Working Set(MB)		Private Bytes(MB)	
	Mean	Std	Mean	Std	Mean	Std
0	0.167	0.373	41.9	3.7	40.6	2.2
1	0.09	0.787	55.7	2.2	51.5	1.7
2	0.084	0.278	57.1	1.9	53.5	1
3	0.202	0.617	57.3	1.4	53.8	0.5
4	0.232	0.649	57.3	1.7	53.8	0.5
5	0.269	0.608	57.3	1.3	53.9	0.3
6	0.221	0.576	59.4	1.3	55.6	1.1
7	0.228	0.595	58.1	1.8	54.2	1.1
8	0.442	0.903	58.6	1.5	54.9	1
9	0.6	0.894	58.4	1.5	54.5	0.8
10	0.514	0.957	58.4	1.4	54.2	0.2
11	0.62	0.955	59.8	0.6	55.6	1.2
12	0.675	1.159	58.9	1.6	54.8	1
13	0.535	0.851	59.2	1.5	55.1	1.2
14	0.515	0.788	58.5	1.8	54.2	0.3
15	0.817	1.215	59.4	1.7	55.1	1.3
16	0.754	1.089	59.2	1.8	55	1.2

# Appendix F

## Simulation: Bandwidth Results

Table F.1: Bandwidth of clients in simulated C/S system (B)

Aspect	Mean	Std
Move rx	41426	961
Move tx	42956	836
Move both	84382	1509
Speech both	37015	2020
Total rx	66378	1248
Total tx	58018	542
Total both	124396	1528

Table F.2: Bandwidth of peers in simulated single region hybrid system compared to to C/S system(B)

Aspect	Slave			Supernode		
	Mean	Std	Change	Mean	Std	Change
Move rx	13092	926	-68%	721452	277	+1641%
Move tx	63687	1528	+48%	13122	930	-69%
Move both	76779	2089	-9%	734574	808	+770%
Speech both	33440	2092	-9%	1006616	2513	+2619%
Total rx	45720	1388	-45%	1312938	2908	+1877%
Total tx	77961	1042	+25%	3111473	4664	+5263%
Total both	123682	2039	+0%	4424412	4644	+3457%



Table F.3: Bandwidth of server in hybrid and C/S systems (B)

<b>Aspect</b>	<b>C/S</b>		<b>Hybrid</b>		
	<b>Mean</b>	<b>Std</b>	<b>Mean</b>	<b>Std</b>	<b>Change</b>
Move rx	644338	1847	2433564	5420	+277%
Move tx	621385	1783	60158	1721	-90%
Move both	1265723	2568	2493722	3700	+97%
Speech both	555219	6720	503309	2513	-9%
Total rx	870275	686	2647900	6053	+204%
Total tx	995666	4612	398650	2350	-60%
Total both	1865942	5032	3046550	4361	+63%