5-2012

Peer-to-Peer Simulation of Massive Virtual Environments

James Dean Mathias
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

Part of the Computer Sciences Commons

Recommended Citation
Mathias, James Dean, "Peer-to-Peer Simulation of Massive Virtual Environments" (2012). All Graduate Theses and Dissertations. 1164.
https://digitalcommons.usu.edu/etd/1164

This Dissertation is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.
PEER-TO-PEER SIMULATION OF MASSIVE VIRTUAL ENVIRONMENTS

by

James Dean Mathias

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Dr. Daniel Watson
Major Professor

Dr. Donald Cooley
Committee Member

Dr. Nicholas Flann
Committee Member

Dr. Ming Li
Committee Member

Dr. Jacob Gunther
Committee Member

Dr. Mark R. McLellan
Vice President for Research and
Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah
2012
ABSTRACT

Peer-to-Peer Simulation of Massive Virtual Environments

by

James Dean Mathias, Doctor of Philosophy
Utah State University, 2012

Major Professor: Dr. Daniel Watson
Department: Computer Science

Massively multiplayer online environments continue to grow in popularity, with current technical designs based upon a well-proven client-server model. This approach has some inherent limitations, high costs to provision server resources for peak demands and restriction of the maximum number of concurrent participants within a virtual environment. Incorporating peer-to-peer (P2P) techniques provides developers the opportunity to significantly reduce costs, while also breaking through the barrier of the number of concurrent participants within a single virtual environment. This dissertation presents a hybrid P2P design incorporating a managed server along with a Voronoi-based P2P overlay for the development of massive virtual environments. In this design, the managed server ensures a secure computing environment and long-term persistent storage, with the virtual environment simulation distributed among the peers, ensuring computational scalability.

(211 pages)
PUBLIC ABSTRACT

James Dean Mathias

Massively multiplayer games make up a large and growing segment of the computer game industry. One of the best known examples of these games is World of Warcraft, developed and published by Activision Blizzard. World of Warcraft boasts a subscriber base of over eleven million active subscribers, earning an estimated $1 billion dollars in 2010.

Some of the core issues for companies that publish these games are the cost of the computers, Internet bandwidth usage, and supporting technical staff. These costs easily reach hundreds of thousands of dollars each month, and in the case of World of Warcraft, millions of dollars. A highly successful game can generate enough revenue to offset these costs, but they represent a high risk venture for any developer, and a significant barrier to entry into the marketplace for smaller developers. Furthermore, even successful companies demand ways to reduce costs as competition for subscribers heats up.

Another issue is that not all players can participate in the same gameplay environment. Current technical hurdles prevent more than a few thousand players from participating together. This results in the subscriber base being split among many different environments, with each environment a copy and completely separate from all others.

The research presented in this dissertation proposes and demonstrates a new technical design that overcomes the limitations noted above. The design dramatically reduces the large computing load and support resources demanded by the games, while also enabling all game players to participate in the same environment, rather than being split among independent copies. The design also ensures security of sensitive player information, such as usernames and passwords, along with providing authenticity of a player’s identity.
# CONTENTS

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th>iii</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUBLIC ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ACRONYMS</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Peer-to-Peer Networks</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Organizing Peer-to-Peer Overlays</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Peer-to-Peer Promise</td>
<td>8</td>
</tr>
<tr>
<td>1.5 Research Scope</td>
<td>8</td>
</tr>
<tr>
<td>2 RELATED WORK</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Historical Progression</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Voronoi Models</td>
<td>16</td>
</tr>
<tr>
<td>2.4 Security</td>
<td>18</td>
</tr>
<tr>
<td>3 THE AUDREY MODEL</td>
<td>24</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Voronoi Connectivity</td>
<td>25</td>
</tr>
<tr>
<td>3.3 Server</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Peer</td>
<td>28</td>
</tr>
<tr>
<td>3.5 Security</td>
<td>33</td>
</tr>
<tr>
<td>4 IMPLEMENTATION</td>
<td>35</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>35</td>
</tr>
<tr>
<td>4.2 Development Tools</td>
<td>36</td>
</tr>
<tr>
<td>4.3 Execution Platform</td>
<td>37</td>
</tr>
<tr>
<td>4.4 Voronoi Overlay</td>
<td>37</td>
</tr>
<tr>
<td>4.5 Processing Architecture</td>
<td>38</td>
</tr>
<tr>
<td>4.6 Networking</td>
<td>40</td>
</tr>
<tr>
<td>4.7 Dropped Packets</td>
<td>44</td>
</tr>
<tr>
<td>4.8 Security</td>
<td>44</td>
</tr>
<tr>
<td>4.9 Data Collection and Visualization</td>
<td>46</td>
</tr>
</tbody>
</table>
5 EXPERIMENTAL SETUP ........................................... 51
  5.1 Introduction ............................................. 51
  5.2 Performance Measures ................................... 51
  5.3 Primary Experiments ..................................... 54
  5.4 Scenario Parameters - Fixed ............................. 57
  5.5 Scenario Parameters - Variable ......................... 60

6 RESULTS ...................................................... 64
  6.1 Introduction ............................................. 64
  6.2 Data Collection and Aggregation ....................... 65
  6.3 Experiment 1 - Bandwidth Scalability for Constant Density .... 65
  6.4 Experiment 2 - Bandwidth Scalability for Increasing Density .... 67
  6.5 Experiment 3 - Model Characterization ................. 69
  6.6 Experiment 4 - Selected Scenarios .................... 81
  6.7 Analysis .................................................. 86

7 CONCLUSIONS ................................................ 88
  7.1 Introduction ............................................. 88
  7.2 Contributions .......................................... 88
  7.3 Future Work ............................................ 89

APPENDICES .................................................... 101
  Appendix A Messaging Benchmark .......................... 102
  Appendix B Detailed Parameters ........................... 120
  Appendix C Comprehensive Results ........................ 124
  Appendix D Reprint Permission ............................ 197

CURRICULUM VITAE ............................................ 199
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Fixed Parameters - Experiments 1, 2, and 3.</td>
<td>58</td>
</tr>
<tr>
<td>5.2</td>
<td>Fixed Parameters - Experiment 4.</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>Parameters - Experiment 1.</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>Parameters - Experiment 2.</td>
<td>60</td>
</tr>
<tr>
<td>5.5</td>
<td>Parameters - Experiment 3.</td>
<td>61</td>
</tr>
<tr>
<td>5.6</td>
<td>Parameters - Experiment 4.</td>
<td>62</td>
</tr>
<tr>
<td>A.1</td>
<td>Simulation Parameters.</td>
<td>113</td>
</tr>
<tr>
<td>A.2</td>
<td>Fixed AOI - Number of Hops.</td>
<td>114</td>
</tr>
<tr>
<td>A.3</td>
<td>Dynamic AOI - Number of Hops.</td>
<td>114</td>
</tr>
<tr>
<td>A.4</td>
<td>Peer Messages - Fixed AOI.</td>
<td>115</td>
</tr>
<tr>
<td>A.5</td>
<td>Peer Messages - Dynamic AOI.</td>
<td>115</td>
</tr>
<tr>
<td>A.6</td>
<td>Spatial Messages.</td>
<td>115</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Voronoi diagram.</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Voronoi neighbors.</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>Typical peer lifecycle.</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Data parallel, fan out, task scheduler.</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>Protocol container.</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Example user account.</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>Example session certificate.</td>
<td>45</td>
</tr>
<tr>
<td>4.5</td>
<td>Network message XML record example.</td>
<td>46</td>
</tr>
<tr>
<td>4.6</td>
<td>Peer state XML record example.</td>
<td>47</td>
</tr>
<tr>
<td>4.7</td>
<td>AudreyViz execution replay.</td>
<td>48</td>
</tr>
<tr>
<td>4.8</td>
<td>AudreyViz bandwidth visualization.</td>
<td>49</td>
</tr>
<tr>
<td>4.9</td>
<td>AudreyViz consistency visualization.</td>
<td>50</td>
</tr>
<tr>
<td>4.10</td>
<td>AudreyViz heatmap visualization.</td>
<td>50</td>
</tr>
<tr>
<td>5.1</td>
<td>Scenario configuration example.</td>
<td>57</td>
</tr>
<tr>
<td>6.1</td>
<td>Bandwidth scalability for constant density.</td>
<td>66</td>
</tr>
<tr>
<td>6.2</td>
<td>Bandwidth scalability for increasing density.</td>
<td>68</td>
</tr>
<tr>
<td>6.3</td>
<td>Bandwidth for a single peer.</td>
<td>69</td>
</tr>
<tr>
<td>6.4</td>
<td>Peer bandwidth - No movement.</td>
<td>70</td>
</tr>
<tr>
<td>6.5</td>
<td>Peer consistency - No movement.</td>
<td>71</td>
</tr>
<tr>
<td>6.6</td>
<td>Peer bandwidth - Slow movement.</td>
<td>72</td>
</tr>
<tr>
<td>6.7</td>
<td>Peer consistency - Slow movement.</td>
<td>72</td>
</tr>
</tbody>
</table>
6.8 Missing peers - Slow movement. ........................................ 74
6.9 Bandwidth - No message or halt failures. .......................... 74
6.10 Consistency - No message or halt failures. ....................... 75
6.11 Bandwidth - Low message and halt failures. ...................... 76
6.12 Consistency - Low message and halt failures. .................... 76
6.13 Missing Peers - Low message and halt failures. ............... 77
6.14 Bandwidth - Movement, message, and halt failures. .......... 78
6.15 Consistency - Movement, message, and halt failures. ......... 78
6.16 Missing Peers - Movement, message, and halt failures. ...... 79
6.17 Bandwidth - Combined message and halt failures. ............. 80
6.18 Consistency - Combined message and halt failures. .......... 80
6.19 Bandwidth - Combined message and halt failures. ............. 81
6.20 Consistency - Combined message and halt failures. .......... 82
6.21 Bandwidth - Scenario 1. ............................................. 82
6.22 Consistency - Scenario 1. .......................................... 83
6.23 Missing Peers - Scenario 1. ....................................... 84
6.24 Consistency - Scenario 2. .......................................... 85
6.25 Missing Peers - Scenario 2. ....................................... 85
A.1 Greedy forwarding. .................................................... 108
A.2 Fixed AOI messages. (a) Best case (b) Pure greedy (c) FIFO (d) Random (e) Working set random (f) Working set recent (g) Working set proportional (h) Grid recent ......................................................... 116
A.3 Dynamic AOI messages. (a) Best case (b) Pure greedy (c) FIFO, (d) Random (e) Working set random (f) Working set recent (g) Working set proportional (h) Grid recent ............................... 117
C.1 Bandwidth - Move = 0.00, Message = 0.00, Halt = 0.00000. ..................................................... 125
C.2 Bandwidth - Move = 0.10, Message = 0.00, Halt = 0.00000.

C.3 Bandwidth - Move = 0.20, Message = 0.00, Halt = 0.00000.

C.4 Bandwidth - Move = 0.30, Message = 0.00, Halt = 0.00000.

C.5 Bandwidth - Move = 0.40, Message = 0.00, Halt = 0.00000.

C.6 Bandwidth - Move = 0.50, Message = 0.00, Halt = 0.00000.

C.7 Bandwidth - Move = 0.00, Message = 0.01, Halt = 0.00000.

C.8 Bandwidth - Move = 0.10, Message = 0.01, Halt = 0.00000.

C.9 Bandwidth - Move = 0.20, Message = 0.01, Halt = 0.00000.

C.10 Bandwidth - Move = 0.30, Message = 0.01, Halt = 0.00000.

C.11 Bandwidth - Move = 0.40, Message = 0.01, Halt = 0.00000.

C.12 Bandwidth - Move = 0.50, Message = 0.01, Halt = 0.00000.

C.13 Bandwidth - Move = 0.00, Message = 0.05, Halt = 0.00000.

C.14 Bandwidth - Move = 0.10, Message = 0.05, Halt = 0.00000.

C.15 Bandwidth - Move = 0.20, Message = 0.05, Halt = 0.00000.

C.16 Bandwidth - Move = 0.30, Message = 0.05, Halt = 0.00000.

C.17 Bandwidth - Move = 0.40, Message = 0.05, Halt = 0.00000.

C.18 Bandwidth - Move = 0.50, Message = 0.05, Halt = 0.00000.

C.19 Bandwidth - Move = 0.00, Message = 0.25, Halt = 0.00000.

C.20 Bandwidth - Move = 0.10, Message = 0.25, Halt = 0.00000.

C.21 Bandwidth - Move = 0.20, Message = 0.25, Halt = 0.00000.

C.22 Bandwidth - Move = 0.30, Message = 0.25, Halt = 0.00000.

C.23 Bandwidth - Move = 0.40, Message = 0.25, Halt = 0.00000.

C.24 Bandwidth - Move = 0.50, Message = 0.25, Halt = 0.00000.

C.25 Bandwidth - Move = 0.00, Message = 0.00, Halt = 0.000417.

C.26 Bandwidth - Move = 0.10, Message = 0.00, Halt = 0.000417.
C.27  Bandwidth - Move = 0.20, Message = 0.00, Halt = 0.000417.

C.28  Bandwidth - Move = 0.30, Message = 0.00, Halt = 0.000417.

C.29  Bandwidth - Move = 0.40, Message = 0.00, Halt = 0.000417.

C.30  Bandwidth - Move = 0.50, Message = 0.00, Halt = 0.000417.

C.31  Bandwidth - Move = 0.00, Message = 0.01, Halt = 0.000417.

C.32  Bandwidth - Move = 0.10, Message = 0.01, Halt = 0.000417.

C.33  Bandwidth - Move = 0.20, Message = 0.01, Halt = 0.000417.

C.34  Bandwidth - Move = 0.30, Message = 0.01, Halt = 0.000417.

C.35  Bandwidth - Move = 0.40, Message = 0.01, Halt = 0.000417.

C.36  Bandwidth - Move = 0.50, Message = 0.01, Halt = 0.000417.

C.37  Bandwidth - Move = 0.00, Message = 0.05, Halt = 0.000417.

C.38  Bandwidth - Move = 0.10, Message = 0.05, Halt = 0.000417.

C.39  Bandwidth - Move = 0.20, Message = 0.05, Halt = 0.000417.

C.40  Bandwidth - Move = 0.30, Message = 0.05, Halt = 0.000417.

C.41  Bandwidth - Move = 0.40, Message = 0.05, Halt = 0.000417.

C.42  Bandwidth - Move = 0.50, Message = 0.05, Halt = 0.000417.

C.43  Bandwidth - Move = 0.00, Message = 0.25, Halt = 0.000417.

C.44  Bandwidth - Move = 0.10, Message = 0.25, Halt = 0.000417.

C.45  Bandwidth - Move = 0.20, Message = 0.25, Halt = 0.000417.

C.46  Bandwidth - Move = 0.30, Message = 0.25, Halt = 0.000417.

C.47  Bandwidth - Move = 0.40, Message = 0.25, Halt = 0.000417.

C.48  Bandwidth - Move = 0.50, Message = 0.25, Halt = 0.000417.

C.49  Bandwidth - Move = 0.00, Message = 0.00, Halt = 0.00100.

C.50  Bandwidth - Move = 0.10, Message = 0.00, Halt = 0.00100.

C.51  Bandwidth - Move = 0.20, Message = 0.00, Halt = 0.00100.
C.52 Bandwidth - Move = 0.30, Message = 0.00, Halt = 0.00100.

C.53 Bandwidth - Move = 0.40, Message = 0.00, Halt = 0.00100.

C.54 Bandwidth - Move = 0.50, Message = 0.00, Halt = 0.00100.

C.55 Bandwidth - Move = 0.00, Message = 0.01, Halt = 0.00100.

C.56 Bandwidth - Move = 0.10, Message = 0.01, Halt = 0.00100.

C.57 Bandwidth - Move = 0.20, Message = 0.01, Halt = 0.00100.

C.58 Bandwidth - Move = 0.30, Message = 0.01, Halt = 0.00100.

C.59 Bandwidth - Move = 0.40, Message = 0.01, Halt = 0.00100.

C.60 Bandwidth - Move = 0.50, Message = 0.01, Halt = 0.00100.

C.61 Bandwidth - Move = 0.00, Message = 0.05, Halt = 0.00100.

C.62 Bandwidth - Move = 0.10, Message = 0.05, Halt = 0.00100.

C.63 Bandwidth - Move = 0.20, Message = 0.05, Halt = 0.00100.

C.64 Bandwidth - Move = 0.30, Message = 0.05, Halt = 0.00100.

C.65 Bandwidth - Move = 0.40, Message = 0.05, Halt = 0.00100.

C.66 Bandwidth - Move = 0.50, Message = 0.05, Halt = 0.00100.

C.67 Bandwidth - Move = 0.00, Message = 0.25, Halt = 0.00100.

C.68 Bandwidth - Move = 0.10, Message = 0.25, Halt = 0.00100.

C.69 Bandwidth - Move = 0.20, Message = 0.25, Halt = 0.00100.

C.70 Bandwidth - Move = 0.30, Message = 0.25, Halt = 0.00100.

C.71 Bandwidth - Move = 0.40, Message = 0.25, Halt = 0.00100.

C.72 Bandwidth - Move = 0.50, Message = 0.25, Halt = 0.00100.

C.73 Consistency - Move = 0.00, Message = 0.00, Halt = 0.00000.

C.74 Consistency - Move = 0.10, Message = 0.00, Halt = 0.00000.

C.75 Consistency - Move = 0.20, Message = 0.00, Halt = 0.00000.

C.76 Consistency - Move = 0.30, Message = 0.00, Halt = 0.00000.
C.102 Consistency - Move = 0.50, Message = 0.00, Halt = 0.000417.
C.103 Consistency - Move = 0.00, Message = 0.01, Halt = 0.000417.
C.104 Consistency - Move = 0.10, Message = 0.01, Halt = 0.000417.
C.105 Consistency - Move = 0.20, Message = 0.01, Halt = 0.000417.
C.106 Consistency - Move = 0.30, Message = 0.01, Halt = 0.000417.
C.107 Consistency - Move = 0.40, Message = 0.01, Halt = 0.000417.
C.108 Consistency - Move = 0.50, Message = 0.01, Halt = 0.000417.
C.109 Consistency - Move = 0.00, Message = 0.05, Halt = 0.000417.
C.110 Consistency - Move = 0.10, Message = 0.05, Halt = 0.000417.
C.111 Consistency - Move = 0.20, Message = 0.05, Halt = 0.000417.
C.112 Consistency - Move = 0.30, Message = 0.05, Halt = 0.000417.
C.113 Consistency - Move = 0.40, Message = 0.05, Halt = 0.000417.
C.114 Consistency - Move = 0.50, Message = 0.05, Halt = 0.000417.
C.115 Consistency - Move = 0.00, Message = 0.25, Halt = 0.000417.
C.116 Consistency - Move = 0.10, Message = 0.25, Halt = 0.000417.
C.117 Consistency - Move = 0.20, Message = 0.25, Halt = 0.000417.
C.118 Consistency - Move = 0.30, Message = 0.25, Halt = 0.000417.
C.119 Consistency - Move = 0.40, Message = 0.25, Halt = 0.000417.
C.120 Consistency - Move = 0.50, Message = 0.25, Halt = 0.000417.
C.121 Consistency - Move = 0.00, Message = 0.00, Halt = 0.00100.
C.122 Consistency - Move = 0.10, Message = 0.00, Halt = 0.00100.
C.123 Consistency - Move = 0.20, Message = 0.00, Halt = 0.00100.
C.124 Consistency - Move = 0.30, Message = 0.00, Halt = 0.00100.
C.125 Consistency - Move = 0.40, Message = 0.00, Halt = 0.00100.
C.126 Consistency - Move = 0.50, Message = 0.00, Halt = 0.00100.
C.127 Consistency - Move = 0.00, Message = 0.01, Halt = 0.00100. . . . . . . . . 188
C.128 Consistency - Move = 0.10, Message = 0.01, Halt = 0.00100. . . . . . . . . 188
C.129 Consistency - Move = 0.20, Message = 0.01, Halt = 0.00100. . . . . . . . . 189
C.130 Consistency - Move = 0.30, Message = 0.01, Halt = 0.00100. . . . . . . . . 189
C.131 Consistency - Move = 0.40, Message = 0.01, Halt = 0.00100. . . . . . . . . 190
C.132 Consistency - Move = 0.50, Message = 0.01, Halt = 0.00100. . . . . . . . . 190
C.133 Consistency - Move = 0.00, Message = 0.05, Halt = 0.00100. . . . . . . . . 191
C.134 Consistency - Move = 0.10, Message = 0.05, Halt = 0.00100. . . . . . . . . 191
C.135 Consistency - Move = 0.20, Message = 0.05, Halt = 0.00100. . . . . . . . . 192
C.136 Consistency - Move = 0.30, Message = 0.05, Halt = 0.00100. . . . . . . . . 192
C.137 Consistency - Move = 0.40, Message = 0.05, Halt = 0.00100. . . . . . . . . 193
C.138 Consistency - Move = 0.50, Message = 0.05, Halt = 0.00100. . . . . . . . . 193
C.139 Consistency - Move = 0.00, Message = 0.25, Halt = 0.00100. . . . . . . . . 194
C.140 Consistency - Move = 0.10, Message = 0.25, Halt = 0.00100. . . . . . . . . 194
C.141 Consistency - Move = 0.20, Message = 0.25, Halt = 0.00100. . . . . . . . . 195
C.142 Consistency - Move = 0.30, Message = 0.25, Halt = 0.00100. . . . . . . . . 195
C.143 Consistency - Move = 0.40, Message = 0.25, Halt = 0.00100. . . . . . . . . 196
C.144 Consistency - Move = 0.50, Message = 0.25, Halt = 0.00100. . . . . . . . . 196
ACRONYMS

**ALM**  Application Level Multicast. Instead of using native IP multicast, primarily due to lack of vendor support, multicast is supported by an application itself. As an application receives a datagram, it multicasts this datagram to any interested receivers.

**AOI**  Area of Interest. Identifies the virtual environment space a peer can see. The AOI may be of a constant fixed size throughout the peer's lifetime, or it may be dynamic, changing in response to its neighborhood or the environment.

**DHT**  Distributed Hash Table. Provides the same key-value pair lookup service as a typical hash table, with the difference being the hash table is stored over a group of distributed computers.

**Hybrid P2P**  Hybrid Peer-to-Peer. A P2P network that includes managed coordination within the overlay. There is no single definition of the exact role of what is managed; the specifics of the hybrid nature are distinct to each technical design.

**MMO**  Massively Multiplayer Online. A virtual environment that is capable of allowing large numbers of concurrent participants, on the order of hundreds or thousands. An MMO is by definition both a VE and an NVE, but an NVE is not necessarily an MMO. This paper primarily uses NVE, intending to communicate that the proposed framework scales from a few participants to many thousands, or more.

**MMORPG**  Massively Multiplayer Online Role Playing Game(s). The vast majority of MMO games are role playing games, typically utilizing a fantasy or science fiction theme.

**NVE**  Networked Virtual Environment. A virtual environment that is shared between networked computers.

**NPC**  Non-Player Character(s). A non-player character refers to a virtual environment participant that is not under the control of any player. An NPC’s actions are programmed, either as scripted events in response to environmental triggers, or through more sophisticated Artificial Intelligence techniques.

**P2P**  Peer-to-Peer. Describes the network organization of the participants, with the purpose that each participant shares computing resources without central coordination.

**VE**  Virtual Environment. An environment which exists as a computer-based simulation. It may exist on a single computer, or be shared between networked computers.
CHAPTER 1
INTRODUCTION

1.1 Introduction

Massively multiplayer online (MMO) environments have become a significant component of the computer game industry, including titles such as Ultima Online, Everquest, Eve Online and, of course, World of Warcraft. These represent only a few names, among a growing and popular landscape of MMOs. World of Warcraft dominates, with Blizzard boasting an active subscriber base of 12 million [22], while other popular MMOs have active subscriber bases in the range of several hundred thousand. MMOs represent a unique opportunity for the entertainment industry; namely, the ability to create persistent virtual worlds with participation counted in the thousands, contrasted with standard multi-player games that have no persistence and allow, at best, a few dozen participants in the same virtual environment (VE). While MMOs are tremendously popular, current commercial deployments are constrained by various technical limitations, affecting both VE design and cost to deliver and maintain.

The technical design of MMOs like World of Warcraft (WOW) is a client-server architecture. The VE simulation is performed at a server, or more specifically, a server farm. Every participant in the VE connects through a computer, with this computer acting as a client. The client renders the VE from the perspective of the participant, collects input, sends this input to the server, monitors for updates from the server, and re-renders the VE based upon updated state information. The server continuously receives inputs from possibly many thousands of connected clients, and manages these inputs to update the VE simulation. The server is the authoritative source for the current state of the VE simulation, disseminating updates to any connected clients.
At the Austin Game Developer Conference in September of 2009, Blizzard discussed the backend requirements used to support World of Warcraft [25]. They report 13,250 server blades running 75,000 CPU cores, spread across an unstated number of server farms. Estimating 200 CPU cores per server farm, this is on the order of 375 server farms, or server instances, as multiple server farms may be located in the same physical location. Using this number and working from their reported subscriber base, we can estimate a maximum allowable number of registered users per server instance, on the order of 50,000, with each instance allowing perhaps up to 10,000 active participants. Another popular MMO, Eve Online [14], makes claim to having the largest number of simultaneous users on a single server (farm) instance of just over 56,000 [15]. These numbers indicate a high cost to purchase and maintain these computing resources, along with a limited number of concurrent users within a single VE. There is a high cost to purchasing these computers, physically hosting them around the world, and paying for network bandwidth, power, cooling, human technical support, as well as other expenses simply to have this level of managed physical resources. Of course, a company with a paying subscriber base of 12 million can afford these physical resources, but even large companies like to find ways to reduce their expenses. More importantly, smaller companies, or even individuals, need to find a way to reduce the upfront financial barrier.

Fundamentally, a client-server technical design results in a high cost risk for any organization, in addition to representing a barrier to entry for smaller organizations. As an organization underestimates or overestimates demand, over-allocation or under-allocation of resources follows. In the case of underestimating demand, customers are frustrated because of poor performance, or an inability to even access the MMO. In the case of overestimating demand, money is unnecessarily wasted in providing physical hosting resources, very likely resulting in a sparsely-populated MMO, making it undesirable for a customer to continue participating.

Another limitation imposed by the client-server architecture is the design of the VE. There is a limitation on the number of simultaneous users on any particular server instance.
This technical limitation, leads to a VE design requirement: The VE is designed to accommodate a tiny fraction of the total subscriber base, leading to the deployment of multiple VE instances in order to accommodate the total subscriber base. Removing this technical limitation frees designers to contemplate new VEs enabling hundreds of thousands, potentially millions, of simultaneous users.

One approach to alleviating the technical limitations imposed by a client-server design is to use peer-to-peer (P2P) network design elements. P2P networks enable an application to distribute computing tasks among the participating clients. As long as each connected client provides more computing resources than it consumes, the potential for a scalable computing environment exists. Traditionally, P2P networks have been used for content distribution, but that is not their only use. They can be utilized for computational tasks, with the computational tasks originating and computed without centralized global coordination.

1.2 Peer-to-Peer Networks

Peer-to-Peer networks are composed of clients connected to clients in an ad hoc networking strategy, relying upon the cumulative resources of participants to provide computing, storage, and bandwidth services, versus a traditional client-server relationship. On top of a standard P2P communication layer, developers define application specific overlay protocols that enable a higher level management of peers within the network. P2P networks are then organized via overlay protocols and utilized for any number of computing tasks, including sharing and distribution of content such as files, relaying email, text messages, news servers, voice over IP (VOIP), and others [21,52,57]. The concept of P2P networking is one with no central, or at best a loosely defined, authority.

Three primary components describe the fundamental nature of a P2P network [46]:

1. Resource Sharing: Each node in a P2P network may provide both client and server functionality. Each node provides services and makes requests of the ad hoc P2P network.
2. Decentralization: No central authority exists that coordinates the activities among the participants in the P2P network.

3. Autonomy: Each node within the network is given responsibility as to what services it provides and what requests it makes of the P2P network, with an expectation of fair usage.

P2P network applications have their origins in one of the earliest widespread uses of the Internet (ARPANET) in Usenet [55]. Usenet is a distributed Internet discussion network, with each server acting as a peer to the service, and with no central controlling authority coordinating the activity among the servers. The basic protocol is that when a server receives a new message, it forwards that message to its neighboring servers who have not yet received the message. In this way, the ad hoc network acts as a message relay service, with each server maintaining a complete copy of all discussions.

Most P2P systems differ substantially from the Usenet model in that no single node in the network contains a complete copy of all content, instead content is distributed throughout the network. As needed, nodes make requests to the P2P network, using an overlay protocol, to obtain desired content or computing resources.

1.3 Organizing Peer-to-Peer Overlays

Three primary approaches to organizing peers into a P2P overlay for use in massive networked virtual environments (NVE) have been proposed, with variations and combinations on these three approaches.

1. Distributed Hash Table

2. Application Level Multicast

3. Mutual Notification

*Distributed hash tables (DHT)* provide the same key-value pair lookup service as a typical hash table, with the difference being the hash table is stored over a group of distributed
computers. A DHT uses a keyspace of some number of bits, for example 128. The set of peer computers in the network are organized into an overlay wherein each peer maintains a routing table of its X closest neighbors in the keyspace. Upon receipt of a service request (lookup or storage), a peer examines the key. If the key matches the peer, the request is serviced at the peer. If the key does not match the peer, it queries its routing table for the neighbor with the key closest to, or matching, the key and forwards the request to that peer. Using this basic scheme, computers can be organized into a P2P network, with no central control and no global knowledge, with the ability to efficiently and robustly service content storage and lookup requests.

*Application level multicast (ALM)* is related to native IP multicast with the difference lying in the location in which the multicast operation is performed. IP multicast is an Internet protocol for sending the same datagram to a group of receivers interested in the same data transmission. The typical uses for this are streaming media, such as Internet based audio and video broadcasts and financial applications, such as stock tickers. In ALM, the multicast is supported by the application itself, rather than the transport medium. As an application receives a datagram, it multicasts this datagram to any interested receivers. Due to the lack of widespread deployment of native IP multicast throughout the Internet, ALM has become the popular choice for multicast.

*Mutual notification* is an approach in which peers maintain a list of neighboring peers and those peers directly communicate with each other. In a naive P2P network, all peers send all messages to all other peers, with the consequence of this being a network that can not scale beyond a few dozen peers at best. In more sophisticated mutual notification schemes, peers are organized using a data structure of some kind (e.g., quad-tree) wherein range queries can be performed to maintain only a localized list of the neighbors within NVE space. Peers in this approach rely upon neighboring peers to communicate relevant events. The research presented in this dissertation is based on mutual notification.
1.3.1 Voronoi Characteristics

Through the use of the a Voronoi diagram peers are able to make direct connections to their area of interest (AOI) neighbors, creating the lowest possible latency, unlike a DHT overlay wherein messages sent to neighbors are routed through many peers first. The computational complexity of the Voronoi diagram is $O(N\log N)$, ensuring the maintenance cost to recompute on a frequent basis is low, and ensuring scalability as the number of neighboring peers increases. These, along with other characteristics, have led to the rising interest in this diagram for organizing P2P NVE overlays.

Further validation of the Voronoi overlay approach is provided by Krause. His work reviews three primary protocols for P2P NVE models, comparing the performance of each under different scenarios to evaluate their performance. The three approaches are: 1) application layer multicast, 2) supernode, and 3) mutual notification.

ALM protocols divide the VE into a number of subregions which are further organized into a multicast group. As events occur, all peers interested in that event are notified through the multicast group. A peer’s interest is defined by a unique AOI. Similar to ALM protocols, supernode protocols also divide the VE into some number of subregions, either a fixed number or one dynamically adjusted at runtime. Each subregion is assigned a node that becomes responsible for its maintenance and messaging, a supernode. Peers register with this supernode as the region enters their AOI. As events occur in the subregion, the supernode is responsible for notifying all registered peers that have an AOI visibility to the event. Mutual notification protocols differ from both ALM and supernode in that they do not divide the VE into subregions. Instead, peers are organized into an overlay using a spatial data structure, such as a Voronoi diagram or QuadTree, and then are responsible for direct communication, notifying each other of events as they occur.

Krause compares these protocols against each other in terms of how long it takes a message to travel from one peer to the destination peer and the bandwidth required at each peer in support of the overall protocol. With respect to message travel time, the mutual notification scheme showed the best performance in all scenarios. With respect to bandwidth
consumption, again, the mutual notification scheme showed the best results. Krause’s work shows that ALM protocols are not affected by local peer density, but greatly affected by global peer density. Supernode protocols are affected by both local and global peer density, with mutual notification protocols not affected by global peer density but affected by local peer density. The author concludes by suggesting that mutual notification protocols are the most promising candidates for low-delay NVEs, with the caveat that concerns with respect to overlay stability and consistency still need to be addressed, and doing so will impact bandwidth.

1.3.2 Hybrid Peer-to-Peer

Real-world deployments of NVEs have several key elements that must be designed into the system. These include the ability to collect and validate payment for authorized users, secure player and VE data storage, management of the VE content, and providing peer executable updates, among others. It is well understood that pure P2P environments are notoriously insecure [54], making a pure P2P system impractical. Therefore, in order to take advantage of the inherent scalability offered by P2P, some elements of standard client-server systems must be incorporated, forming a hybrid P2P framework.

As a peer joins a VE, an account is required in order to authenticate participation. This account must be securely maintained by a managed server. Details of payment, credit card numbers, expiration dates, payment status, and other sensitive private information simply can not be stored throughout a P2P content framework; instead, it requires a physically managed server.

While peers interact, they must have a way to verify others peers’ authenticity. In order to do this, certificates created and signed by a managed server are exchanged and verified among participating peers. Upon verification the peers may now trust each other and freely interact. A managed server makes it possible to provide this kind of service.

Massively multiplayer environments are long lived, requiring a steady stream of content and client executable updates, and P2P-based systems are no different. Companies need access to a physically managed server through which new content can be added and
disseminated. Similarly, when a new client executable is required, it needs to be distributed from a secure source. A managed server makes both of these easily possible. As a peer logs into the server for authentication, new content and client updates can be made at that time.

As players participate in the NVE, data regarding their interactions and character advancement is generated. These data must be securely stored on a managed server. While P2P content distribution systems exist [47], they do not provide the secure and guaranteed online storage requirements of a 24x7 uptime NVE. The only reasonable assurance of security and uptime is through a managed server.

These requirements demand the need for a system that incorporates both a managed server and a P2P computing framework. The server is responsible for account security, content/client updates and secure data storage. The active peers are used to distribute the simulation and communicate with each other. This is an architecture that is neither client-server or pure P2P, it is a hybrid design.

1.4 Peer-to-Peer Promise

Peer-to-peer networked virtual environment systems borrow from many fields of computing, including peer-to-peer systems, distributed computing, simulation, security, networking, and databases. It is the application of these fields in a unique combination that makes P2P NVE research a distinct activity. The promise of P2P versus client-server networks includes improved scalability, lower cost of ownership, improved fault tolerance, better network utilization, improved response to network resources, and new opportunities for massive virtual environment designs.

1.5 Research Scope

To date, no P2P model that has been proposed, and crucially, none demonstrated, meets the requirements necessary to create an interactive and secure computational platform that is capable of simulating massive virtual environments. This dissertation presents a hybrid model that is primarily a P2P network, but incorporates the use of a centrally
managed server. The primary purpose of the server is to provide authentication and credentials to peers requesting to participate in the virtual environment. Once authenticated and credentialed, no further interaction with the server is necessary, peers then interact only with each other. The purpose of this research is to define the framework for such a model and demonstrate it is capable of meeting the requirements for an interactive massive virtual environment. This critical step will provide a new paradigm to the entertainment software industry that has the potential to dramatically alter the way in which large simulation environments are deployed, with the benefit of moving the cost of server-based models away from the simulation provider.

The research was conducted in three phases. The first was the development of a simulation to demonstrate the feasibility of the proposed hybrid model, along with data collection and visualization techniques to characterize its performance. The second phase was the development of an operational implementation of the model, also including a data collection and visualization application used to aid in the characterization of its performance. The simulation is a single process, multi-threaded application, capable of simulating thousands of peers, using a lockstep timing protocol. The operational implementation was developed on a modest LAN of inexpensive, small form factor computers. The third, and final, phase was a set of experiments, using the operational implementation, with the purpose of characterizing performance of the model and implementation.

The major contribution of this work is the definition and demonstration of a new hybrid P2P model for the simulation of massive virtual environments. The hybrid model is composed of a set of distributed peers, authenticated through a centrally managed server, all connected through a Voronoi diagram overlay. During the development of this model, various other contributions were made, including: distributed P2P data collection and analysis techniques, networking protocols, benchmarking approaches (refer to Appendix A for a full writeup), and security techniques.
CHAPTER 2
RELATED WORK

2.1 Introduction

There is a large volume of research regarding P2P networks, with a growing segment devoted to P2P-based NVEs. The research covers topologies for organizing peers into a network, messaging, routing, security, and a number of other topics. The current P2P NVE literature is overwhelmingly concerned with proposals regarding approaches for the organization of a large set of peers into an overlay model and how to manage object state and communication throughout the overlay, once organized. Issues such as security and reliability are addressed, but at this time, are not yet of primary research focus. While the research is active, it remains an immature area, with no clear winning strategy and no commercial deployment of a P2P-based MMO.

A client-server based NVE has a server that maintains global knowledge of the current state of the VE. All movement and state changes are communicated to the server where the VE is simulated and new states returned to each client. A significant issue a P2P system must overcome is communicating those movement and state changes among a distributed set of peers. Whether explicitly stated or not, the fundamental problem being addressed by the current P2P NVE research is related to ensuring that all peers are able to send and receive to and from those other relevant peers, do it within reasonable network bandwidth limits, and trust those communications. A client-server system can easily do this because the server maintains global knowledge of all connected clients. P2P systems face a much greater challenge by their distributed nature, no single device in the network has global knowledge of the VE. This chapter discusses the current state of research as it relates to
P2P systems for massive NVEs, other P2P systems, such as those for file sharing, or content
distribution are not considered in this review. The discussion is divided into three sections.
Section 2.2 discusses historical progression, Section 2.3 discusses research regarding the use
of the Voronoi diagram as an overlay, and Section 4.8 presents a review of security issues.

2.2 Historical Progression

In order for a P2P system to work towards the common goal of providing a service or
networking environment, the individual peers must be organized into an overlay, or model,
through which they can communicate and share resources. This section provides a detailed
chronological review of the various proposed and demonstrated models for organizing peers
into an NVE.

The earliest proposed P2P NVE model is Solipsis [39] in 2003. The authors propose an
approach for a globally-connected network, including a login and teleportation protocol for
finding the peer closest to a desired destination. An interesting feature of the Solipsis design
is that of representing the VE as a two-dimensional torus, versus a typical approach of a
rectangular region. The advantages of the torus are the improvement in global connectivity,
because there are no peers on the edge of a rectangle, and the average distance between
any two peers within the VE is reduced. For example, two peers on opposite sides of
a rectangle are neighbors on a torus, at least in terms of an overlay topology, and the
maximum distance between two arbitrary peers is one-half the maximum dimension of the
torus. In 2008, Frey et al. present a new architecture, under the name of Solipsis [24], it
is a pure P2P approach, based upon an n-dimensional Voronoi overlay. Computationally
intensive tasks such as physics and collision detection are distributed throughout the overlay
instead of a central server; the architecture, unfortunately, does not consider the primary
issue of security.

In 2004, Knutsson et al. present the design and results from their P2P research exper-
iment, SimMud [40]. The SimMud architecture is built upon the Pastry [48] foundation,
utilizing the Scribe [11] application level multicast infrastructure. The authors claim their
results show it is feasible to create a P2P-based massive NVE appropriate for the type of
FreeMMG [16] is the first to suggest a hybrid approach, combining both a centrally managed server and a P2P overlay. This model proposes the server to be responsible for player authentication, session tracking, player area of interest, along with data storage and backup. The game simulation is executed by the clients using the P2P lockstep model as implemented by the Age of Empires II game [43]. The server ensures a cheat resistant authentication infrastructure, while the lockstep networking and simulation model ensures a consistent distributed game state. The authors demonstrate this framework, through simulation, with up to 300 peers. While this model is shown to be adequate for relatively small numbers of clients, it remains to be seen if it can work with thousands or hundreds of thousands of peers. In particular, the ability of the lockstep model is highly suspect in its ability to provide interactive performance scalability beyond a few hundred clients.

Another model introduced in 2004 organizes the VE into a set of federated zones [35]. In this model, global states are assigned to zones. As a peer wants to modify one of these global states it can make a request to the zone owner to modify the state. The zone owner judges conflicts and communicates the updated states. Initially, all zones do not have owners, but over time, peers can make requests to become zone owners, whereby they then become responsible for all states within that zone. Peers can also voluntarily resign from being a zone owner. If a peer leaves the overlay, all global states it is responsible for are also lost. The communication strategy employed by this model is based upon a distributed hash table structure, like that proposed by Castro et al. [10], for discovery of zone owners; after that, it is direct communication. The authors demonstrate this model being effective for small NVEs of less than 500 peers.

The first work from the Voronoi adaptive scalable transfer (VAST) project was also published in 2004 [18,32,36]. VAST is an ongoing P2P NVE research project that is based around the use of a Voronoi diagram to construct and maintain a P2P network overlay. The group has identified several fundamental operations peers may use for overlay maintenance, including join, move, and leave, along with proposals for managing game state among the
P2P overlay. The authors demonstrated these three procedures, within a Voronoi-based overlay, provide a scalable solution as the number of peers increases. The project maintains a continually-updated, open source implementation of a P2P NVE library and simulation.

An unusual, and unique, approach to deploying NVEs over a P2P network is to embrace the nature of a P2P network, as opposed to emulating a client-server model. In 2005 Hughes et al. presented the results of their work [34]. They begin with the assumption that P2P networks are unpredictable. Specifically, they focus on P2P networks without a central authority. Their suggestion is to embrace this in the design of the game world, rather than force predictability over the network. Their VE design is intimately tied to the underlying peers participating, with each peer hosting a room. As long as a peer is active, the room is available, when a peer fails to respond, that room is not accessible. Ultimately, the game is about exploration of the P2P environment, versus a goal-directed game. While interesting, this technique is not of value to the vast majority of NVE designs.

In 2006, Chen and Muntz present peer clustering [17], another hybrid approach. Their model uses a managed server for secure storage of things like usernames, passwords and long term object persistence. The VE is divided into regions small enough for a peer to manage and utilizes a distributed hash table approach for dividing the region responsibilities among the active peers. Communication between peers is handled through the use of a Pastry based routing scheme [48].

Another DHT based approach was developed by Hampel et al. in 2006 [31]. They utilize a Pastry [48] network and Past storage architecture [21]. The VE is divided into arbitrarily hexagonal sized regions, with each region being assigned to a single region controller. In addition to the assigned region controller, several backup controllers are also assigned, in an effort to mitigate cheating and network failures.

Continuing with the theme of using DHT communication to form a P2P overlay, in 2007 Fan et al. presented Mediator [23]. Mediator uses a structured P2P overlay, formed through a DHT, along with application level multicasting and direct P2P communication. The VE is broken down into a number of rectangular zones based upon load management.
needs. These zones incorporate one or more super-peers, or mediators, each of which may take on different roles, which then have responsibility to maintain the overlay, VE simulation and communication activities. The design of this framework proposes to use EigenTrust [38] reputation and the DCRS anti-free-riding algorithm [29] to construct a reward and reputation-based load balancing computational network that self-organizes into zones of simulation activity.

In 2007, Backhaus and Krause provided a review of the VAST approach, evaluating how well the architecture supports bandwidth requirements and maintains neighborhood consistency under two different models of player movement [4]. Under standard random waypoint movement both bandwidth and neighborhood consistency are well within acceptable limits. Using group-based random waypoint, a flocking behavior, bandwidth is again acceptable but neighborhood consistency begins to fail as time progresses. The authors attribute this to the problem of non-bilateral neighbors, where peer A sees peer B as a neighbor, but peer B does not see peer A as a neighbor. This must be a mistake, as this breaks the definition of neighbors in the model. By definition, if peer A sees peer B as an AOI; enclosing or boundary neighbor, peer B will see peer A as a neighbor also. This does not negate their results, but it does mean they misunderstand the problem and therefore, the solutions they recommend do not necessarily apply.

Two years later, in 2009, Ahmed et al. propose a hybrid P2P architecture for zonal (region) based NVEs called massively multiuser virtual simulation architecture (MM-VISA) [1]. This model divides the VE into hexagonal zones, with each zone assigned a master peer. The master peer is responsible for coordinating the activities of all peers within its zone of responsibility. A differentiating feature of this model is that in addition to the VE zoning, peers of different movement classes are clustered together. Peers within the VE may have different movement characteristics, some typically move slower or faster than others, and the clusters are based upon these movement characteristics. Based upon the division of the VE into zones and the categorization of peers into movement clusters, they invoke a rather complicated communications strategy to disseminate events, ensuring delivery of
those messages and maintenance of the overlay. The model also introduces an intelligent zone switching mechanism, to combat the problem of peers that move along the boundary of a zone, generally requiring the peer to frequently change the master node they work with. To do this, they introduce an overlapping boundary between zones, within which, peers do not switch master nodes, which significantly reduces the problem of constant master node switching.

In the same year, Buyukkaya et al. proposed to combine both DHT and a Voronoi overlay in VoroGame [8]. They use a Voronoi overlay in the same manner as VAST to support direct communication and game event dissemination. The interesting addition is the additional use of a DHT to support data management. The peers are also organized through the DHT for data management and persistence. Therefore, time critical events are still communicated directly through the Voronoi based overlay, while data persistence is managed through the DHT. Unfortunately, the use of the DHT as an approach for persistent data storage is problematic as peers come and go throughout the simulation. When a peer is not connected to the overlay, data persisted at that peer becomes unavailable. It is also relevant to note this is only a proposed approach, nothing has yet been implemented and no results given.

QuON [5] is a Quad-Tree-based overlay for organizing peers into an overlay. This work, presented in 2010, builds upon the authors’ previous experience in showing the benefits of mutual notification schemes and working with Voronoi-based NVE architectures [4, 41]. It shares much in common with the Voronoi-based approach originated by Hu et al. In their research, Backhaus and Krause describe using a Quad-Tree to ensure network connectivity among peers. Their simulation results show that QuON ensures better overall topology consistency and bandwidth utilization. In particular, the results show improvement in bandwidth utilization as the size of a local group increases, however their results are only for a fixed size AOI, which is not appropriate for the vast majority of NVE designs.

Carlini et al. propose the integration of P2P and cloud computing to realize massively multiplayer virtual environments [9]. In this architecture, they propose to construct a set
of Virtual Nodes (VN) which are composed of the participating peer computing resources. The computational workload of the simulation is shared among the VNs. Because of the untrusted and transient nature of peer resources, this architecture integrates the use of trusted and reliable cloud computing resources as backup VNs. While this proposed architecture offers the chance at greater scalability versus client-server designs, due to the requirement of backup VNs on cloud computing resources, the high cost of deployment remains.

The best closing to the general topic of the potential for P2P based MMOs was written in late 2010 by Miller and Crowcroft [44], of Microsoft Research. In this work, the authors briefly review the past ten years of work in the field of P2P MMOs. In general, this paper is pessimistic regarding the potential, at least in the near term feasibility of a P2P system, primarily due to bandwidth limitations. In their research they captured actual network traces from World of Warcraft, then repurposed those data in a publish-subscribe P2P simulation. Based upon their findings, they report that a client-server topology easily satisfied residential broadband limits, whereas their P2P simulation was often saturated. The model, implementation, and results presented in this dissertation stand in opposition to their P2P-based approach, and therefore, their findings.

2.3 Voronoi Models

This section reviews work related to the use of the Voronoi diagram to organize peers into an NVE overlay. To date, the Voronoi overlay has shown the most promise and has received the most attention, relatively speaking. At the same time, there are only a handful of publications, further illustrating the relative immaturity, and opportunity, of the research in this area.

Hu et al. have the most significant results to show of any P2P research to date through the VAST research project [18, 32, 36, 50]. VAST is the name of the framework library they are developing, but it is also the umbrella under which all their research is taking place. The research from this group is the first to demonstrate the use of a spatial data structure, the Voronoi diagram, to form a P2P overlay. Their work introduces several important procedures for such overlays, including the join, move, and leave procedures,
among others. These procedures define the interactions among peers to maintain overlay consistency. Their simulation results have shown the feasibility, in terms of bandwidth consumption and overlay consistency, for such P2P systems to be successful.

The VAST group first proposed the concept of a Voronoi diagram in 2004 [32], but it was not until 2006 they published work based upon simulation results [50]. Later published work is related to topics that improve various aspects of the overlay, including reducing bandwidth through optimizing movement updates [18] and suggesting alternative update approaches such as spatial publish-subscribe [33]. However, spatial publish-subscribe moves away from a P2P system by introducing super peers that become responsible for a VE region, which add an additional hierarchical layer to the overlay topology.

Cavagna et al. propose the use of a Voronoi diagram to support a distributed and streaming 3D world [12, 13]. They refer to the work of the VAST group as the basis for selecting the Voronoi model. Their use of the Voronoi diagram is to decompose and distribute a very large 3D spatial database among a set of peers, enabling efficient data distribution and visualization as a client navigates the scenery. Their first publication is only a proposal, the second shows initial results from their work, with these results showing the P2P system performs more efficiently than a centralized data distribution method. Unfortunately, while their results demonstrate an improvement, they do not demonstrate true scalability. The data transfer is more efficient, but it grows at the same rate as centralized data distribution with an increasing number of clients/peers. Fundamentally, this is a content distribution network, having only a passing similarity to an interactive P2P VE.

VoroGame is a design proposal for a Voronoi-based model that also utilizes a DHT for live object persistence [8]. Each peer in the model is assigned a spatial region of responsibility based upon its neighborhood, organized through a Voronoi diagram. The DHT is used as a means for the overlay to compute the network location of objects within the VE to retrieve/update their state. This proposal comes from the same research group as Cavagna, mentioned in the previous paragraph. This is only a design proposal, and at the
time of this writing, no publications demonstrating this design either through simulation or a real-world implementation are available.

In 2008 Genovali and Ricci published an extended abstract outlining their intention to explore the use of the Voronoi diagram to create distributed virtual environments [27]. The unique component they are pursuing is weighting the diagram tessellation as part of modeling hierarchical P2P networks. The weighting of each node is proportional to the computational resources available at each peer, creating a larger area of responsibility for peers with greater resources than those of its neighbors. In [2] the authors show simulation results in support of their proposal that a weighted diagram effectively performs load balancing operations.

2.4 Security

Due to the ad hoc, dynamic topology and loose authority structure, the issue of security for P2P systems is distinct from the standard client-server model. In a client-server model, the server is a validated or trusted computing platform under the control of an interested party. Each node within a P2P system acts as both a server and a client and, by definition, is not a physically controlled device. Therefore, the need to validate each peer in such a network is of utmost importance. As noted in Chapter 1, the work presented in this dissertation is a hybrid P2P system, not a pure P2P system. A motivating reason for this choice is to eliminate many of the security concerns. Because of this design, not all P2P security issues are a concern for this research.

A general theme throughout the literature is that pure P2P networks, by their nature, have participating malicious nodes. Most of the approaches presented in the literature discuss security in terms of probabilities. For example, the probability a message is delivered, the probability of detecting a malicious node and so forth. Generally, current techniques do not provide the ability to guarantee the absence of malicious nodes, therefore, P2P networks must have systems in place to mitigate or manage the influence of malicious nodes. Due to the absence of hybrid P2P systems, there is no discussion to what degree, if any, such systems suffer from these issues.
P2P systems face a variety of vulnerabilities from malicious or faulty nodes within the network. Some of these include denial of service, failure to deliver or mis-route messages, providing corrupt content, refusal to fulfill requests, fairness in resource consumption, and others. The remainder of this section provides a review of these issues and various proposed solutions.

2.4.1 Denial of Service

Denial of service (DoS) should be considered an entire vulnerability category rather than a specific attack. Most security issues in P2P systems can be classified as a form of DoS. One DoS might take the form of malicious nodes providing corrupt content in response to requests, another might be malicious nodes overloading the system with frivolous content requests. While distinct in their method, they are both effectively DoS attacks. Many of the security issues outlined in the remainder of this chapter fall within the category of DoS.

An example of a standard denial of service attack comes from the recording industry practice of placing false content within music sharing systems [19, 42]. Christian and et. al. show the effectiveness in denying content availability within music file sharing systems by placing a large number of decoys in the network, or by making available low quality, unwanted, versions of the originals, techniques they call item poisoning and pollution. Decoys contain the same meta-data as the original content, such as the musician name, date and track title, with the content being essentially a random set of bits, or some other unusable data set. These techniques increase the number of items returned when a query is performed against the network, resulting in both a network latency effect and requiring the end-user to select from a large set of results. In order to ensure an original music file is downloaded, the end-user is forced to download a large number of files, which is effectively a DoS attack. This class of DoS has become a common practice by the music industry, with some companies, such as MediaDefender and MediaSentry, offering this kind of service on a commercial basis.
2.4.2 Secure Routing

The decentralized nature of a P2P network results in the fact that each node within a P2P system is aware of an extreme minority of other nodes, therefore, in order to transmit a message to a non-neighbor node, these systems rely upon some routing protocol to forward messages among nodes. In an unsecured network, malicious nodes can choose to ignore routing requests, delay requests to increase latency, route to incorrect nodes (delaying the message delivery), or route messages to other malicious nodes.

One solution to the issue of secure routing is proposed by Wallach [54], whereby a message is sent over a variety of routes. Assume that some fraction of neighbor nodes are malicious, and that incorrectly route a message. A message is initially sent to all neighbors from the originating node all subsequent nodes forward only to their neighbor closest to the destination. In doing so, the message is sent to the destination over a variety of routes. This is due to the fact that each neighbor is a different distance to the destination and therefore, their closest neighbors (with respect to the destination) are relatively diverse. For a DHT-based system, Wallach’s simulation results show the message forwarding arrives at the destination 99.9% of the time as long as the fraction of malicious nodes is less than 30% of the total P2P network.

2.4.3 Fair Resource Usage & Utilization

Nodes within a P2P network are expected to consume and share resources in a fair manner. For example, a node may make a service request from a P2P network. At the same time this node is expected to provide similar services; that can be accessed by other peers within the network. It is the aggregation of the peer computing resources that make the network desirable. Unfair resource utilization by peers within the network can prevent the overall network from being desirable to join. Unfair resource utilization is not limited to individual malicious nodes; nodes might collude to create an unfair utilization among each other within the context of the overall network [45].

Ngan et al. present a research study that attempts to solve the issue of fair resource sharing among nodes [45]. Their model requires nodes to publish auditable records of their
usage. Given these records, other nodes can inspect the fairness of peer activity. These records, as a result, provide a kind of economic incentive for nodes to actively participate in the network. The general model is that each node publishes a record of network utilization, including both consumption and sharing. A peer desiring to validate the truthfulness of the published record can follow an auditing trail by checking the record of use from one node to the next. If the published logs of nodes show an inconsistency, the cheating node can be detected and rejected from the network. Through simulations, the authors show, the auditing overhead is very low and scales well, suggesting that auditing not only ensures fairness, but also provides an incentive for active participation in providing resources.

2.4.4 Trusted Computing

A problem faced by nodes within a P2P system is knowing which nodes to trust and not trust. One approach to the issue of node trust within P2P networks is reputation [30]. A reputation system aggregates feedback from peers’ past behavior to construct a characterization of reputation. Based upon this characterization, a node can choose to trust or not trust the services provided by its peers within the network. Nodes who trust each other can share their level of trust, creating a kind of interconnected reputation network.

One proposed solution to securing P2P networks is suggested by Sandhu and Zhang [49] through the use of a trusted computing architecture. Software techniques have not provided a guarantee of eliminating malicious nodes from a P2P network. On the other hand, the utilization of a hardware-trusted computing platform has the potential to ensure a network composed only of validated nodes.

Generally speaking, the Trusted Computing Group [28] describes trust as an expectation that a computing device behaves in an expected manner for a designed purpose. With respect to P2P networks, a device is a consumer level computing device, such as a desktop or laptop PC. The essential element for a trusted computing device is the inclusion of a hardware component called the trusted platform module (TPM). The TPM provides on-chip security services which include:
1. Public-key cryptographic services, which includes key pair generation, random number generation, signature verification, etc.

2. Integrity measurement functions, which protects data from access by malicious code

3. Attestation services, which provide cryptographic proof to a third-party that software has not been compromised.

Sandhu and Zhang [49] show that given a set of devices that all incorporate a TPM, a secure P2P architecture can be constructed. This architecture allows nodes within the system to validate the identity of any other nodes, provide role-based enforcement, identify roles, allow the migration of user credentials from one system to another and provide secure storage and distribution of confidential content throughout the network.

A similar approach to providing trusted identity is presented by Rowstron and Druschel [21]. They discuss the PAST storage management system which utilizes smart-cards. Each user of the PAST system is issued a smart-card that is associated with a unique private/public key pair. The purpose of the smart-card is to provide integrity to the assignment of a node ID, which is an essential component of P2P network security, to the computing device when accessing PAST. If one can trust the identity of a node, all other operations within the network are trusted.

The only currently suggested software only based solution is to use a centrally controlled authority that is responsible for the assignment of peer credentials [54]. The managed authority is contacted by each node before it enters the network, is assigned a credential that is signed by the authority and recognized by other peers in the network. This provides the motivation for the hybrid P2P design presented in this work.

2.4.5 Security Summary

Some research suggests various protocols that can minimize the activity of malicious node participation, or provide some confidence in message delivery, etc., but to date, no entirely software-based approach has demonstrated the ability to completely secure a pure P2P network. The general theme of the current research indicates trusted hardware and
central identity authentication are the only promising approaches to reliably securing such networks. For example, a commercial P2P deployment would make use of smart cards as a means to authenticate the identity of nodes. Alternatively, a P2P vendor can utilize a managed certificate issuing authority for node identification, thereby creating a hybrid system. In spite of the difficulty in securing these networks, their use continues to grow, with corresponding research at various levels and use of a variety of techniques to bring reliability and security.
CHAPTER 3
THE AUDREY MODEL

3.1 Introduction

Audrey is a hybrid P2P-based framework for massive networked virtual environments. Four key components create the model: Voronoi-based overlay, lightweight server, heavy-weight peers, and Kerberos style security scheme. The design of Audrey is neither pure client-server nor pure P2P. Instead, it is a hybrid P2P system, borrowing from both designs but weighted heavily towards P2P systems. The model is relatively straightforward and easy to understand, the novelty is in the design and use of its components.

The server is primarily responsible for framework security and long-term persistence, but also provides a number of other services. It is very much a server with ultimate VE responsibility, albeit significantly lighter weight than the server in a typical client-server model. Peers have the crucial responsibility to maintain the framework connectivity and the VE simulation. They are organized into a Voronoi-based overlay network, inspired by, but significantly expanding upon the procedures introduced by the voronoi overlay network (VON) [32]. The Voronoi overlay is used to determine peer connectivity for communication and also for spatially decomposing the VE to distribute the simulation workload among its peers. The security mechanism follows the Kerberos model [51]. It is the combination of the server, peer, Voronoi overlay, and security model that form the essential core of the Audrey model. Section 3.2 begins the model presentation by describing the use of the Voronoi diagram to form the overlay topology, Section 3.3 details the server, Section 3.4 describes the peer, finally, Section 3.5 discusses the security scheme.
3.2 Vorooni Connectivity

Given a Euclidean space $S$ and a set of points $P$ contained within that space, there is a region surrounding each point that is closer to that point than any other point. The diagram that describes these regions is known as a Voronoi diagram [3]. The Voronoi diagram has a computational complexity of $O(N\log N)$, where $N$ is the number of points contained within the space. Peers in the Audrey model are the points in a Voronoi diagram. Therefore, as the number of peers increases, the computational complexity grows at $O(N\log N)$, ensuring framework scalability. It is worth noting, no single peer in the model ever computes a Voronoi diagram based upon every active peer in the overlay. Peers have a limited view of their local environment, having knowledge of an extreme minority of active peers, typically on the order of less than 20, more or less depending upon the VE design.

Figure 3.1 shows an example Voronoi diagram. The points in the figure represent the location of peers in the VE space. The lines form regions that describe the space that is closer to the point contained within the region that is closer to it than any other point. Furthermore, the lines represent edges that connect points to each other as neighbors.
Once computed, the data structure representing the Voronoi diagram can be queried for the neighbors of any particular point or region of interest. Regions that share edges are known as enclosing, or direct, neighbors. Points that fall within a specific radius of a point of interest are known as area of interest (AOI) neighbors. Figure 3.2 shows a Voronoi diagram with both kinds of neighbors highlighted. The point of interest is indicated with its region edges in bold. The enclosing neighbors of this point are lightly shaded. The AOI neighbors include both the light and dark shaded regions; AOI neighbors generally include all enclosing neighbors.

Each peer maintains a Voronoi diagram based upon all peers with which it is interested, its neighbors. The peer utilizes the data structure to know which other peers should receive updates on its movement. Additionally, it is used to know when to inform a neighboring peer of the arrival of a potential new neighbor. Section 3.4 describes these procedures in detail.
3.3 Server

The server’s role is to provide a managed resource for hosting a secure NVE. The server manages all data requiring long-term persistence, such as account information, player characters, statistics, and persistent environment objects. The server shares some services common to standard client-server NVE designs, deviating significantly by having the network communication, connectivity, and VE simulation coordinated among the active peers, rather than computed and coordinated by the server. This is the key distinction, it does not participate in the VE simulation; it is not a computational component.

The server provides the following services:

- Framework Security
- Account Maintenance
- Overlay Bootstrapping
- Long-Term Persistence

As discussed in Chapter 2 there is no known security solution for a pure P2P system; some form of managed security is required. The key role for the server in this model is that of provider of framework security. The foundation of the security model is through the use of public key encryption within a Kerberos scheme. Section 3.5 describes the security scheme.

Strongly related to security is account maintenance. To ensure participation of only validated users, and to ensure identity of participants, the model requires each participant to have a registered account. Accounts are created at the server and validated through a username/password combination during login. The server securely supports these services.

Overlay bootstrapping is the process through which peers become connected to the network overlay. During an active session, a peer is assigned an initial VE position, either by the server or through some other design mechanism. Peers initially login with the server, and upon successful validation, receive session credentials. Following login, the peer makes a request to join the peer overlay, known as a forwarding request. The server is the initial
point of contact for forwarding. Upon receipt of a forwarding request, the server responds by sending the peer the contact information of an active peer it believes is closest to the indicated starting location. The forwarding peer then contacts the active peer to continue the forwarding process; this repeats until the peer closest to the starting position is found. Upon completion, the requesting peer initiates neighborhood discovery to reveal its initial peer neighbor environment and begins active participation in the overlay. Following initial forwarding, any time a new VE starting position is necessary, an active peer already known to the forwarding peer can be used as the initial point of contact, bypassing the server entirely. Appendix A is a publication (PDPTA 2011 conference proceedings) resulting from this research, which describes in detail a P2P messaging benchmark applied to several different forwarding techniques.

The server is the endpoint for any data requiring long-term persistence. As already noted, all data related to user accounts is securely persisted by the server. Other participant session generated data, such as player statistics, or VE objects, are stored by the server. A server is necessary because it is the only physically managed component of the framework with guaranteed availability. Unmanaged peer systems have no guarantee of availability, reliability, and crucially, secure storage; therefore, they cannot be used for long-term object persistence.

3.4 Peer

All peers are homogeneous; there is no differentiation in function or responsibility among any of the peers. Peers coordinate among each other to maintain the VE simulation, interacting with the server only for account creation, login, initial forwarding, and logout. The peer process is heavyweight in that it has a wide range of computational responsibilities, including networking demands, simulation workload, and rendering of the VE from the player’s perspective.

Peer communication is always direct, with each peer sending and receiving network packets to and from any peer of interest; messages between peers have no need for routing through the overlay. Throughout a peer’s active participation, every peer encountered is
recorded and kept in memory. The record of each peer encounter includes the network endpoint (an IP:Port address combination) for communication. Any time communication between peers is necessitated, a direct connection is made. While not specifically indicated by the model, it is generally expected network communication between peers is performed using a low-level stateless communication protocol (e.g. UDP communication) for performance reasons. Any communication requiring state is layered (by a developer) on top of the low-level networking protocol.

Peers maintain their set of neighbors from two categories, enclosing and AOI neighbors. The number of enclosing neighbors is defined by the Voronoi diagram, typically in the range of 6 to 12. The number of AOI neighbors is defined by the radius of the AOI and the number of other peers contained within that region. Given that AOI neighbors include enclosing neighbors, the number of neighbors could possibly include other peers if the AOI region is larger than the dynamic region defined by the enclosing neighbors.

A peer may be configured to maintain a minimum (if available) or maximum number of neighboring peers to prevent problems associated with an AOI that contains a large number of peers (i.e., crowding). In combination with a fixed AOI and a maximum number of possible neighbors, the actual number of neighbors fluctuates throughout the lifetime of a peer. The reason for choosing among these possibilities is specified by the needs of the VE interaction design and constrained by the networking demands. As the number of neighboring peers increases, so does the networking bandwidth requirement.

The next several sub-sections detail specific model behaviors and procedures of a peer.

### 3.4.1 Position Update

As peers move through the VE, they inform their neighbors through a position update protocol; Algorithm 1 shows this protocol. Each time the peer moves, or at a specified time interval (e.g., every 250 milliseconds), a peer sends its current VE position to all of its known neighbors. Additionally, it sends the distance from itself to its most distant neighbor.

A receiving peer records the new peer position, and temporarily remembers the previous position. Each time a position update is received, a receiving peer examines each of its
Algorithm 1: Position update.

Input: Peer position \( p \)
Input: Set of Enclosing and AOI Neighbors \( C \)

begin
\[ d_{\text{max}} \leftarrow \text{Max distance to neighbor in } C \]
for \( c \in C \) do
 Transmit \( p \) and \( d_{\text{max}} \) to \( c \)
end
end

neighbors to determine if any are now within the max neighbor distance of the moving peer. If any neighbors newly fall within that distance, the peer sends a message to these two peers to inform them they are potential new neighbors. This technique ensures peers continually learn of potential new neighbors. Algorithm 2 outlines this procedure.

Algorithm 2: Potential neighbors.

Input: Moving neighbor \( n_{\text{mov}} \)
Input: Moving neighbor’s previous position \( p_{\text{prev}} \)
Input: Moving neighbor’s new position \( p_{\text{new}} \)
Input: Moving neighbor’s max peer distance \( d_{\text{max}} \)
Input: Set of Enclosing and AOI Neighbors \( N \)

begin
for \( n \in N \) do
\[ d_{\text{prev}} \leftarrow \text{distance from } p_{\text{prev}} \text{ to position of } n \]
\[ d_{\text{new}} \leftarrow \text{distance from } p_{\text{new}} \text{ to position of } n \]
if \( d_{\text{new}} \leq d_{\text{max}} \text{ and } d_{\text{prev}} > d_{\text{max}} \) then
 Introduce \( n_{\text{mov}} \) to \( n \)
end
end
end

3.4.2 Neighborhood Update

A peer updates its local view of the VE through a neighborhood update procedure; Algorithm 3 shows this procedure. The data from all accumulated position updates, all accumulated potential new neighbors, and failed neighbors (Section 3.4.3) are utilized to determine a peer’s new enclosing and AOI neighbors.
Algorithm 3: Neighborhood update.

**Input:** Set of current Enclosing and AOI Neighbors $N_{cur}$
**Input:** Set of potential neighbors $N_{potential}$
**Input:** Set of failed neighbors $N_{failed}$
**Output:** Set of new Enclosing and AOI Neighbors $N_{new}$

1. begin
2. $O ← \emptyset$
3. $T ← N_{cur} \cup N_{potential} \setminus N_{failed}$
4. $V ← Voronoi(T)$
5. for $v ∈ V$ do
6. if $v$ is enclosing or $v$ is AOI then
7. $O ← O + v$
8. if $v$ not in $N_{cur}$ then
9. Request neighbors from $v$
10. end
11. end
12. end
13. end

The procedure begins by combining all current enclosing neighbors, AOI neighbors, and potential new neighbors into a single set; any position updates have been committed to the peers. All failed neighbors are removed from this set. Working from this final set, a Voronoi diagram is computed. From the Voronoi diagram, all peers considered enclosing and AOI are identified as current neighbors, with any others no longer considered neighbors. Additionally, any peers newly considered neighbors are sent a request for their neighbors. Upon receipt, those neighbors are fed back into the set of potential new neighbors to be used the next time the neighborhood update procedure is executed.

3.4.3 Neighbor Failure and Recovery

For all peers considered neighbors, aliveness requests are maintained. As long as a peer responds to an aliveness request, it continues to be considered a neighbor. Upon failure of an aliveness request, the neighboring peer is no longer considered active by the requesting peer and is disregarded as a possible neighbor in the next neighborhood update. When a neighboring peer fails aliveness, all other neighbors are sent requests for all of their
Figure 3.3. Typical peer lifecycle.

currently known neighbors. The purpose in doing this is to quickly gather a snapshot of active neighbors in a wider VE area than currently known and feed those peers into the next neighborhood update procedure. This simple technique is used to detect and heal the overlay due to peers that halt unexpectedly for any reason.

3.4.4 Lifecycle

Figure 3.3 shows a diagram that describes the lifecycle of a typical peer. For a first time participation, an account is created followed by a login with the server; all subsequent participations bypass account creation and begin with login. Following login a peer requests forwarding from the server, which continues among other active peers until the final forwarding peer is discovered. Following completion of forwarding, the peer requests neighbors from the final forwarding peer in order to obtain its initial neighborhood. At this point, the peer enters active participation in the overlay, which is a continual loop of updating its neighbors and participating in the overlay. Depending upon the VE design, the active participation may include forwarding to a new VE starting position. Upon normal exit of a peer, it logs out from the server and terminates.
3.5 Security

Security is a core design element, supported primarily through the use of public key encryption (PGP), following a Kerberos style scheme. A public/private key pair is assigned to the server, with all peer clients distributed with the server’s public key. The server’s keys are used to perform standard cryptographic procedures, including signing, verification, and encryption.

When a peer logs in to the server, the server generates a unique session certificate for a peer. The certificate includes the peer’s networking endpoint, date/time the certificate was created, and the length of time for which the certificate is valid. Additionally, these items are signed by the server’s private key, with this signature also being a part of the certificate. The model does not specify the security mechanism used for login, as that is an implementation detail, not a model specification; any secure data transmission scheme is acceptable, such as SSL. However, following successful login, no further use of SSL or other encrypted data transmission scheme is required to support correct peer validation and participation.

Algorithm 4: Peer validation.

| Input: Certificate IP:Port $C_{Addr}$ |
| Input: Certificate Signature $C_{Sig}$ encoded by Server’s Private Key $K_{Spri}$ |
| Input: Packet IP:Port $P_{Addr}$ |
| Input: Server Public Key $K_{Spub}$ |
| Output: Validation of Peer |

begin
if $C_{Addr} = P_{Addr}$ and $C_{Sig}$ validated by $K_{Spub}$ then
| Peer is valid |
else
| Peer is invalid |
end

Certificates are exchanged between peers upon initial contact. The certificates contain the networking endpoint of the peers, for example, the IP:Port combination. Each peer first compares the the IP:Port from the certificate with the IP:Port from the incoming network packet containing the certificate. If they do not match, verification fails immediately. If
they match, verification continues by validating the server’s signature on the certificate through the use of the server’s public key. If the signature validates, the peer is validated. Algorithm 4 outlines the peer verification decision making process. This validation ensures not only that the certificate is valid (i.e. signed by the server), but also that the network identity of the computing device associated with the certificate is the same one validated by the server; a session certificate is tied to a single computing device through its network identity.

Each peer is able to validate the authenticity of another peer and its permission to participate in the network overlay by verifying whether the other peer’s certificate signature is from the server. This essential design element allows peers to validate each other without having to contact the server, further supporting the security and scalability of the Audrey model.
CHAPTER 4
IMPLEMENTATION

4.1 Introduction

A core outcome for this research is the development of an operational framework, one that implements the Audrey model, and one that executes in a real-world environment. Having an operational framework ensures nothing is either consciously or subconsciously overlooked. For example, network communication and protocols are real; all the detailed complexity of creating the network communication is required versus a simulation wherein memory is easily shared and algorithmic complexity is hidden. Having an operational system ensures the peer clients are truly asynchronous at runtime; versus a simulation wherein peers are simulated using a lock-step timing mechanism, which easily overlooks real-world network messaging delivery and timing issues. These are a few examples among a myriad of other details that differentiate between an operational framework from a simulation. A simulation of a model can only suggest what may or may not be possible, whereas the successful demonstration of an operational framework is the result in and of itself. This chapter presents the successful implementation of the Audrey model as an operational framework.

Following the model presented in Chapter 3, the implementation is a distributed application, composed of a lightweight server and heavyweight peers, with the peers connected through a Voronoi-based overlay. The server and peer components share significant design and source code elements. Unless noted, the sections in this chapter describe design and functionality common to both. In addition to the operational framework, a separate post-execution visualization component was developed (AudreyViz), which is used to collect, aggregate, summarize, and visualize execution logs. The implementation is only the
necessary underlying connectivity and security framework to create a VE. The implementation does not include any interactive player control or player perspective visualization capabilities. Other support exists in the form of scripts used to automate deployment and execution, in addition to the post-execution visualization application.

The remainder of this chapter details the specifics of the Audrey model implementation. Section 4.2 begins with a description of the languages and other tools used for the implementation. Section 4.4 describes the implementation of the Voronoi diagram. Section 4.5 describes the task based execution model used to provide application scalability.

4.2 Development Tools

Several languages are used for application coding, deployment, and execution automation. The server and peer components are written exclusively in C++. The purpose in choosing C++ is to ensure the best runtime performance possible in addition to ensuring wide-scale portability. Both Visual Studio and Linux makefile projects are maintained; the server and peer applications compile and execute on both Windows and Linux (in addition to runtime interoperability). The AudreyViz application is written in C# and is a WinForms application only intended for execution on a Windows platform; incidentally, it does compile and run under Linux through the use of the Mono and MonoDevelop projects. Finally, Python is used as a scripting language to automate deployment and execution of the peer executables.

In addition to the primary languages, several third-party toolkits are used to support the application development. Boost [7] is used to provide cross-platform threading, cross-platform data type definitions, and networking. The Crypto++ [20] library provides public/private key generation, signing, and verification functionality, along with other cryptographic capabilities. RapidXML [37] provides XML parsing and persistence, which is used for the persistence of an XML based user account database for the server, as well as data logging for the peers. These three toolkits are all included as C++ source components, and all three are open source (LGPL/MIT) or are public licensed.
4.3 Execution Platform

A small local area network (LAN) of 40 GuruPlug [56] computers was used to host the peer processes, in addition to a standard desktop computer used to host the server. Each GuruPlug computer is a small form factor computing device. Its physical dimensions are approximately 2 x 3 x 4 inches. Each device includes a 1.2 GHz ARM based CPU running a Debian Linux 2.6 kernel, with 512MB of RAM, 512MB of flash storage, and a wired gigabit Ethernet connection.

The GuruPlug computers run the peer processes, with the server executing on the desktop computer. Each GuruPlug device is capable of running 10 peer processes under the most demanding experimental movement and failure conditions, enabling execution scenarios of up to 400 asynchronous peer processes. This number of peer processes is a large enough number to reasonably demonstrate the large scale participation and scalability.

4.4 Voronoi Overlay

The Audrey model specifies the use of a Voronoi diagram to form the peer overlay. The operational framework uses a custom C++ implementation of the Voronoi diagram. Specifically, the implementation uses the sweep-line method described in Chapter 7 of, *Computational Geometry - Algorithms and Applications, 3rd edition* [6].

Prior to the development of the operational framework, a simulation of the Audrey model was developed as a tool to explore and initially validate the model. The Voronoi algorithm was originally written as part of the simulation development. The simulation code is written in C#; therefore, the original Voronoi code is also written in C#.

Given an existing implementation, and considering the complexity of the algorithm and code, a conversion from C# to C++ was performed. The C# language has automatic garbage collection as a language feature, while C++ does not. The key focus for the conversion was ensuring no memory leaks, in other words, ensuring that any memory allocated from the heap is also deallocated. The technique used to ensure this worked was to place a pointer into a queue every time memory was allocated. Using this queue, the Voronoi class destructor works through the queue and frees the memory for every pointer in the queue.
The valgrind [53] dynamic analysis tool was used to validate that all memory was correctly freed by the C++ implementation.

Within the server and peer application code, the design is such that the nature of the spatial diagram is unknown. The overlay behavior is defined in an abstract base class named \textit{P2POverlay} that details what behaviors a spatial diagram must support in order to work as an overlay. These behaviors include the ability to construct a diagram from a set of points and then perform different neighbor queries when given a specific point. The custom Voronoi implementation derives from this abstract base class and provides concrete implementations for these behaviors. In the future, if it is desired for any reason to use a different spatial overlay structure, there is a relatively straightforward process for doing so. The new data structure simply derives from the abstract P2POverlay class, and the application instantiates the new data structure instead of the Voronoi diagram; the rest of the application code remains untouched.

### 4.5 Processing Architecture

The underlying processing design of the peer and server is that of a data parallel, fan out, task scheduling processing core. In this design, all computational pieces are subdivided into atomic tasks that can be executed in parallel. The implementation includes a common thread pool and a shared (synchronized) work queue. Any task that requires execution is placed on the work queue, and the next available worker thread removes it from the queue and executes the task. As noted in Section 4.2, in order to ensure cross platform functionality, all threading and synchronization primitives are provided by the boost library.

At startup, a process creates a thread pool with an initial worker thread count matching the number of available CPU cores. These worker threads all listen to the shared work queue. Upon receipt of a new work item, the queue signals the thread pool a new task is available for processing. The next available worker grabs the next item in the queue and executes that task. If no tasks are available for processing, threads enter an efficient wait state, waiting to be signaled to grab a new task. This design and implementation enables a highly scalable computational core for both the server and peer processes.
Figure 4.1 shows a diagram with the shared work queue, including three active worker threads from a thread pool. Thread 1 is shown receiving Item 1, Thread 2 is shown receiving Item 2, and Thread 3 is shown receiving Item 3. Assuming a multi-core, or multi-CPU architecture, each of these threads executes in parallel.

To ensure incoming network packets are captured and delegated as work items as quickly as possible, one application thread (in addition to the threads in the thread pool) is dedicated to receiving incoming packets. Upon receipt of a packet, a work item is created and placed in the work queue for processing. The network thread then returns to a waiting state, listening for a new network packet. The work item for a network packet is responsible for decoding the message and taking appropriate action based upon the contents of the message.

Similarly, another application thread is dedicated to sending outgoing packets. A (synchronized) Singleton send queue is available to all code throughout the process. Any time a packet needs to be sent, the work item creates a message packet and places it in the send queue. The thread dedicated to sending packets is signaled and immediately sends the packet.
4.6 Networking

All network communication is UDP. Every communication that occurs between the server and a peer, or between peers, is performed within the context of a *protocol*. Each protocol describes two state machines, one for the initiator and one for the receiver. The state machines describe the messages that are sent and received, ordering, along with the state transitions that occur upon the receipt of a message. The protocol also describes timeouts and retries for each state in support of messaging failures and/or delays.

The framework includes a base protocol class that is the underlying implementation of the state machine from which all protocols are derived. The base implementation contains the functionality for state processing, transitions, timeout periods, number of retries for each state, and a callback mechanism for reporting the completion of the protocol. The protocols are designed to fit within the processing architecture described in Section 4.5; therefore, the execution of a protocol is within the context of a work item.

4.6.1 Protocols

The model described in Chapter 3 describes the nature of the Audrey model, however, there are many different possibilities for implementations. The model does not prescribe the use of *protocols* as the communication implementation technique, others can be applied. This section identifies the different protocols developed as part of this implementation. These protocols form the communication scheme through which peers communicate with the server and each other. Again, while not specifically prescribed by the model, these protocols strongly indicate the structure an alternative implementation may take.

**Create Account** This protocol allows a client to create an account. During this protocol, details such as a username, password, and contact information are established with the server. The purpose of this protocol is to support demonstration of the security scheme as described by the model. In order for only valid peers to securely participate, an account is necessary for login validation.
Done Forwarding This protocol is used by a peer to inform the server it has successfully finished forwarding. The server uses this knowledge to update its forwarding working set.

Forwarding This protocol is used by a peer to find the active peer closest to its starting position. All peers, and the server, maintain a small working set of known active peers. Upon receipt of a forwarding request, a peer examines its working set, and its neighbors, for the peer closest to the requesting starting position. The contact information for the closest peer is sent to the requesting peer. The protocol ends when the peer receiving the request recognizes itself as the peer closest to the destination.

Login This protocol is used by a peer to securely log in and receive credentials from the server that allows it to participate in the overlay. The credentials come in the form of a certificate signed by the server’s private key. The certificate includes the peer’s unique account id, player name, time of login, networking endpoint, and the server’s signature of these items.

Logout This protocol allows a peer to gracefully leave the overlay. All known neighbors and the server are sent messages indicating the peer is leaving and should be considered inactive.

Neighbor Notify This protocol is used to introduce two peers to each other. When a peer notices one of its neighbors has moved within AOI distance of another of its neighbors, this protocol is initiated with those two peers as a means to inform them of each other, that they are now potentially neighboring peers. This is the core mechanism through which peers become aware of each other, other than through forwarding.

Peer Introduce This protocol is used by a peer to inform another peer that it now considers that peer a neighbor. Following a neighbor notify, a peer performs an update to determine if the peer about which it was notified has become a new neighbor. If it is now recognized as a neighbor, this protocol is used to introduce itself.
**Peer Move** This protocol is used by a peer to notify all of its neighbors it has moved within the VE. This is the only protocol without any timeouts or retries, it is a simple fire and forget. The reason for this is that movement updates happen regularly, and if movement packets are dropped, a new movement update quickly happens again, correcting for any dropped packets.

**Request Aliveness** This protocol is used by peers and the server to determine whether or not a peer is truly active and accessible over the network. Both the server and peers use it to track the active state of peers in their forwarding working sets. Both processes maintain a queue of peers they have seen, when a peer is lost due to aliveness, a replacement is selected from this queue and placed into the active set (along with a new aliveness protocol started for this peer). Peers also use this protocol as an aid to track possible halt failures among neighboring peers.

**Request Neighbors** This protocol is used by a peer to request from another peer all of its neighbors. Following forwarding, the forwarding peer initiates this protocol with the final peer in the forwarding sequence. Upon receipt of those neighbors, the protocol is initiated with those peers. This is the scheme used by a peer to initially discover peers in its neighborhood for consideration as neighbors. The protocol is also used by a peer when it loses a neighbor due to failure of an aliveness protocol. Upon loss of a neighbor, this protocol is initiated with all of its current neighbors, in an effort to thoroughly examine is greater neighborhood for peers it can consider as new neighbors to fill the hole created by the failed peer.

**Request Session Certificate** This protocol is used by a peer to obtain the session credentials of another peer. At initial contact, peers initiate this protocol in order to receive the other peer’s certificate. Before continuing any communication, each peer validates the other peer’s certificate.

**Request Start Position** This protocol is used by a peer to request a VE starting position from the server. This is a simple random location in a 2D Euclidean space.
4.6.2 Protocol State

To provide state to protocols when using a stateless communication scheme (UDP) and a task-based parallel processing model, a master table of protocols is maintained. The master table is a simple key-value associative container. Specifically, it is a synchronized hash table. The initiator of a protocol assigns a unique identifier (GUID) to the protocol at creation. This identifier is used as the key into the master table, with a pointer to the protocol class (containing the state) stored as the value. Every network packet includes the GUID of the protocol to which it belongs. Upon processing of a network message, the protocol GUID is decoded and used as the lookup key in the master table, from which the protocol state is referenced.

Figure 4.2 is a representation of the protocol container. The Keys are GUIDs and the values are pointers to instances of unique protocol class instances. This scheme is what enables a process to maintain and easily lookup state among all the many active protocols. The use of a hash table also aids scalability as its lookup complexity is $O(1)$.
4.7 Dropped Packets

Because the physical network the framework runs on a best-case scenario (a local area network without external traffic; refer to Section 4.3). A small concession is made to simulate real-world Internet conditions of packet loss. The send queue is parameterized with the ability to drop packets before they are sent. The parameter is a probability specified through a run-time execution scenario configuration. Before sending a packet, the send queue generates a new random number. If it is under the indicated probability, the packet is never sent, thus simulating a message delivery failure. This mechanism is used to execute the framework under different levels of packet loss.

4.8 Security

As described in Chapter 3, the security model relies upon public/private key pairs, along with server signed certificates. The server is assigned a unique public/private key, with its public key distributed to all peers, directly compiled into the executable. This section highlights the use of key pairs and certificates throughout the framework.

An account certificate is created when a peer requests a new account. This certificate is composed of a unique account ID, date the account was created, username, password (SHA-256 hash), first name, last name, and gameplay name. Figure 4.3 shows an example of the data recorded for each user account. During the account creation protocol, the peer computes an SHA-256 hash of the user’s password, then uses the server’s public key to encrypt all data before sending it to the server. The server then uses its private key to decode the data in order to create the account certificate.

The Crypto++ library is used to generate the server’s public/private key pair, perform the SHA-256 password hashing, and perform the encryption/decryption of the data using the server’s key pair. For the purposes of this research, the size of the key pair is 1024 bits; however, this is a simple parameter than can be changed to be any desired size.

During peer login, the server generates a session certificate. This certificate is composed of the peer’s unique account ID, gameplay name, public key, date/time issued, date/time of expiration, and the peer’s networking endpoint, all signed by the server’s private key.
Figure 4.3. Example user account.

Figure 4.4. Example session certificate.

Figure 4.4 shows an example session certificate. During login the peer sends its username and password (SHA-256 hash), encrypted using the server's public key. The server responds by sending the peer's session certificate back to the peer in clear text; there is no need for encryption as it is intended to be shared with other peers for validation.

The crucial elements of the session certificate are the networking endpoint and the signature. The endpoint is the one through which the peer contacted the server, recorded into the certificate, and signed by the server. This endpoint is also the same endpoint other peers see during their communication with the peer. Other peers can validate the authenticity of the certificate, and therefore, the peer, by verifying the endpoint they are receiving communication from matches the one signed by the server in the certificate, all without having to make contact with the server.
4.9 Data Collection and Visualization

During execution, each peer maintains a detailed, in-memory, log of events. This includes a record of every network message sent and received, including the type of message and its size. Additionally, once per second (a configurable parameter) the peer records its currently known neighbors. At the end of an execution scenario, each peer persists these data to a local XML formatted file, including their account ID as part of the filename in order to create unique filenames that are used for aggregating data from all peers from an execution scenario.

Figure 4.5 shows an example record for a single network message. <PID> is the account ID of the peer. <Time> is the time the message was recorded. <ProtocolID> is the GUID that identifies to which protocol the message belongs. <X> and <Y> identify the VE location at which the message was recorded. <In> indicates whether the message was sent or received. <MsgType> is an value and identifies the specific type of the message (e.g., a login request, or a logout notification). <Size> is the total size of the message, including the IP and UDP header.

Figure 4.6 shows an example record for a peer state record. <PID> is the account ID of the peer. <Time> is the time the peer state was recorded. <X> and <Y> identify the VE location of the peer at the time the state was recorded. <Neighbors> contains the set of neighbors the peer knew at the time the state was recorded. <N> indicates a neighbor of the...
peer, with <PID>, <X>, and <Y> identifying the account ID and VE location of the neighbor, respectively, at the time the snapshot was taken.

4.9.1 AudreyViz

All of the data is aggregated and initially summarized using the AudreyViz application. AudreyViz includes the capability to view aggregated data in tabular or graphical form, along with the capability to export aggregated data for additional visualization and analysis using other tools. It also features the capability to replay an execution scenario, including support for pausing or stepping through the replay. The replay offers a global reconstruction of the overlay, a perspective not possible at runtime because no single component has knowledge of every peer at runtime, including the server.
Figures 4.7 through 4.10 show some of the different visualization capabilities of AudreyViz. These screenshots show the detailed control available to the user for filtering, visualizing, and exporting data.

Figure 4.7 is a snapshot from an animated replay of an execution scenario of 400 peers. The points are the VE position of the peers and the lines are the edges of the Voronoi diagram of the peers. At any time during the replay, the user can use the mouse to select a peer and see a report of which other peers it should have known at runtime, based upon the global reconstruction, versus those peers recorded during runtime. This capability forms the basis for one of the most important evaluation techniques of the model, as detailed in Chapter 5.

Figure 4.8 is a screenshot showing the network bandwidth utilization. The X-axis is time, measured in seconds from the start of the scenario. The left Y-axis shows bits per seconds (bps), with the two noisy lines showing the average and median bandwidth utilization at each peer. The right Y-axis is the number of active peers in the scenario,
with the smooth line showing this measure starting with 0 and quickly peaking at 300. The user can filter the bandwidth by any subset of message types, including a single message type. This allows one to examine in detail the contribution each message type, or group of message types, has on the overall bandwidth utilization among the peers.

Figure 4.9 is a screenshot showing a measurement known as consistency; Chapter 6 details this measure. Again, the X-axis is time, measured in seconds from the start of the scenario. The left Y-axis is the consistency measure, and again, the right Y-Axis the number of active peers.

Figure 4.10 shows a screenshot demonstrating the heatmap visualization capability. The visualization region is the same Euclidean space of the VE. All data is logged with the VE position it generated or received. AudreyViz can take these data and represent them as a heatmap over the VE space. The screenshot in 4.10 shows the distribution of forwarding requests during the execution scenario. The user can select any subset of message types, or individual message types, for visualization.
Figure 4.9. AudreyViz consistency visualization.

Figure 4.10. AudreyViz heatmap visualization.
5.1 Introduction

This chapter presents the experimental setup used to characterize the Audrey model and implementation, as described in Chapters 3 and 4. The experimental setup is composed of two primary components. The first is the VE, and the second is the set of parameters used to describe each execution scenario; these parameters control the behavior of the server and peers. The overarching goal driving the set of parameters used for the experimental setup was to provide an overall characterization of the Audrey model, specifically, a characterization of the implementation of the model as described in Chapter 4. This chapter also presents the measures of performance used throughout the results and analysis discussion in Chapter 6.

The chapter starts with a discussion of the performance measures in Section 5.2, as they provide the background the remainder of this dissertation. The primary focus of the chapter begins with Section 5.3, wherein the VE and runtime scenario parameters are presented and discussed. Finally, Sections 5.4 and 5.5 detail the parameter settings used for the scenario executions.

5.2 Performance Measures

The two measures of performance are overlay consistency (consistency) and network bandwidth utilization (bandwidth). These represent the key performance characteristics within the context of a suite of runtime parameters. Section 5.2.1 details how consistency is measured along with insights into its interpretation. Section 5.2.2 describes the measurement of network bandwidth utilization.
5.2.1 Overlay Consistency

Overlay consistency is a comparison of the set of neighbors a peer correctly knew at runtime versus the ideal set of neighbors it should have known. Consistency is measured throughout the lifetime of the scenario execution; it is a time series. The rate at which consistency is measured matches the rate at which peers record their known neighbors.

In order to compute consistency, two components are necessary: data capture of each peer’s neighbors at runtime and a methodology for computing the ideal neighbors. Runtime data capture is straightforward. At runtime, each peer periodically records the identity of its neighbors to a logging file; for this research, this rate is once per second.

At each time step in the series, the consistency is a value in the range of [0.0, 1.0], with 1.0 indicating a peer recording the same set of peers as the ideal set. Equation 5.1 shows the simple formula for this computation.

\[
\text{consistency} = \frac{\text{runtime}}{\text{ideal}}
\]  

(5.1)

The nature of the P2P-based framework leads to the issue that no single runtime component knows of the aliveness or VE location of every peer in the overlay, including the server. Peers in the overlay do not know, or even estimate, the neighbors of its neighbors. Each peer tracks, and records, the position of its neighbors, but even those data may be instantaneously incorrect due to network transport delay in position updates, and computational frequency. In fact, each peer only authoritatively knows the peers it currently maintains as neighbors and its own VE position. Therefore, a post execution tool is required in order to compute the theoretical neighbors, based upon the logging data provided by all peers during execution.

Chapter 4 introduced AudreyViz, a tool used in support of results visualization and analysis. AudreyViz includes a capability to construct a global Voronoi diagram of the framework overlay based upon the logging data provided by all peers. The self-reported authoritative positions from all peers are taken. From these a global Voronoi diagram is constructed for each time step in the scenario replay. At each time step, for every peer, the
application uses the diagram to determine the theoretical set of neighbors a peer should have known. Equation 5.1 is computed for every peer within a single time step. It is permissible for a peer to know about more neighbors than the global reconstruction indicates, due to scenario configuration settings. In the case this happens, the consistency measure is capped at 1.0. Combining the consistency measure for all peers, the average and user specified percentile (e.g., 50th percentile) are computed and recorded as the measure for that time step. This computation is performed for every time step in the scenario, creating a consistency time series.

In addition to the measurement of consistency, for each time step, AudreyViz reports the number of active peers, the number of peers that perfectly knew their neighbors, and the number of peers missing 1, 2, 3 or more neighbors. These data provide additional insight into understanding a consistency value less than 1.0.

5.2.2 Network Bandwidth Utilization

Network bandwidth utilization is measured in bits per second (bps), kilo bits per second (Kbps), or even mega bits per second (Mbps). For reference, a dialup Internet connection is typically in the range of 14 to 42 Kbps, with broadband cable and DSL ranging anywhere from 256 Kbps to 8 Mbps, or more. Transfer rates are generally asymmetric, with download speeds much greater than upload. The nature of the Audrey model results in symmetric bandwidth utilization; all values of bandwidth used throughout this dissertation are reported as total bandwidth, the summation of both incoming and outgoing data. Therefore, if a value of 100 Kbps is reported, 50 Kbps is incoming data and 50 Kbps is outbound data.

Chapter 4 describes the technique used to collect the raw network data. Following a scenario execution, these data are ingested, processed, and exported using the AudreyViz tool, writing the data in a format useable by applications such as MS Excel for further visualization and analysis.
5.3 Primary Experiments

Four primary sets of experiments were performed:

**Experiment 1** The scenarios in this experiment have an increasing number of peers, but are distributed over an increasingly large VE, in order to maintain a constant spatial density. The purpose of this experiment is to show that with an increasing number of moving peers, at constant density, bandwidth is scalable.

**Experiment 2** The scenarios in this experiment have an increasing number of peers, but are distributed within the same sized VE, resulting in an increasing density of peers. The purpose of this experiment is to show that with an increasing number of stationary peers, at variable density, bandwidth is scalable.

**Experiment 3** This experiment provides a broad characterization of the model under a variety of runtime scenarios. These scenarios varied peer movement, message delivery failure, and peer halt failures, with respect to each other.

**Experiment 4** The two scenarios in this experiment were to show the model under one condition more representative of potentially expected conditions and another of dramatic, high peer failure.

For experiments 1 and 2, *scalable* means the bandwidth either does not increase, or increases at a rate that stays well within desired bandwidth limits (e.g., residential broadband). These two experiments are similar in their goals; however, they demonstrate scalability from two different perspectives.

5.3.1 Virtual Environment

In general, the VE is a simple, unitless, two-dimensional Euclidean space. Peers freely move about in the space without obstruction. If a peer attempts to move beyond the borders of the VE, its movement vector is reversed in order to keep it within the pre-defined space. The size of the VE has no particular meaning; its choice is driven primarily to aid in easing human understanding of the values when reviewing raw data.
For Experiment 1, the VE increased in size proportional to the number of peers in order to maintain a constant density. The density remained constant at 1 peer per 6250 square (unit-less) space of the VE. Experiments 2, 3 and 4 utilized a 10,000 by 10,000 two-dimensional VE.

5.3.2 Scenario Parameters

There are a large number of parameters used throughout the model and the implementation. These include networking timeouts and retries, minimum and maximum number of neighbors a peer should keep, frequency at which to internally update the set of known neighbors, movement rates, and many others. The number of parameters is large enough that a full parametric analysis would result in tens or even hundreds of thousands of executions, which is impossible for this, or possibly any, work. It was decided to focus on those experiments and parameters that help characterize the model under normal operating conditions and those that characterize the model under increasingly stressful conditions. Three key parameters are used to specify these conditions: peer movement, message failure, and peer failure. A supplemental listing of system parameters not identified in this chapter is available in Appendix B.

Peer movement describes the rate of movement for a peer. With different VE designs having different movement demands, it is useful to characterize the model under a range of these conditions. At one end of the spectrum, a design might not require any movement, or extremely infrequent movement. At the other end, another design might demand high rates of movement. The Audrey model is sensitive to the rate of movement, as peers are in constant coordination with each other to stay abreast of which other peers should be considered neighbors.

Message failure provides a means by which stressful Internet conditions can be simulated. Under ideal conditions, messages are always delivered, and in the wilds of the Internet, this never happens. The execution scenarios are performed on a LAN; therefore, this parameter is necessary to help characterize an important aspect of the real-world Internet.
Peer failure, or halt failure, in a real-world environment, occurs for a variety of reasons. Firstly, a computing device may unexpectedly fail, for any number of reasons, failing the peer process with it. Secondly, in any massive, and widely geographically distributed computing environment, power failure, or Internet service provider failure, is expected. Thirdly, users do not always exit gracefully. They may power down their computer without notice, or kill the application without going through a graceful logout procedure. Finally, and not least, applications have bugs that result in unexpected halt failures. All of these conditions result in a peer that simply stops communication without notice. This parameter is necessary in order to characterize the system with respect to these real-world conditions.

### 5.3.3 Scenario Configuration

In order to easily parameterize peers for an execution scenario, a scenario configuration file is used. At startup, the server reads the scenario configuration, which contains the scenario parameters, and transmits these to each peer during the login protocol. The configuration file is an XML formatted file, for ease of human readability and modification. Figure 5.1 shows an example scenario configuration.

The `<Move>` section describes whether or not a peer should move and if so, at what rate. The movement of a peer is characterized by it first randomly selecting a direction vector, a number of steps to move along that vector, and a rate of movement. After the specified number of steps are completed, a new direction vector, number of steps, and rate of movement are randomly selected, and continue until the scenario is complete. The `<DeltaVMean>`, `<DeltaVStdDev>`, `<StepsMean>`, and `<StepsStdDev>` parameters are used by a Gaussian number generator for this process. Finally, if movement is enabled, the `<Probability>` parameter indicates the probability per 100 milliseconds the peer will move.

The `<MessageFailure>` section describes whether or not message failure is to be simulated, and if so, at what rate. In the case of message failure, the `<Probability>` parameter indicates the probability, per message, that a message is dropped.

The `<HaltFailure>` section describes whether or not halt (peer) failure is to be simulated, and if so, at what rate. In the case of halt failure, the `<Probability>` parameter
<Scenario>
  <Move>
    <Use>true</Use>
    <Probability>0.1</Probability> <!-- per 100 milliseconds -->
    <DeltaVMean>10</DeltaVMean>
    <DeltaVStdDev>5</DeltaVStdDev>
    <StepsMean>100</StepsMean>
    <StepsStdDev>25</StepsStdDev>
  </Move>
  <MessageFailure>
    <Use>false</Use>
    <Probability>0.01</Probability> <!-- per message -->
  </MessageFailure>
  <HaltFailure>
    <Use>false</Use>
    <Probability>0.0005</Probability> <!-- per second -->
  </HaltFailure>
</Scenario>

Figure 5.1. Scenario configuration example.

indicates the probability, per second, that a peer fails. In other words, once per second, the peer generates a uniformly distributed random number, if that number is less than, or equal to, the halt failure probability, the peer immediately stops all communication and participation for the remainder of the scenario; it effectively disappears from all other peers. The event log of the peer, prior to the simulated halt failure, is still recorded and used for system evaluation.

5.4 Scenario Parameters - Fixed

This section presents and discusses the fixed scenario execution parameters used for all experiments. As noted previously, the number of parameters is too great to comprehensively vary each parameter with respect to all other parameters. Table 5.1 lists the fixed parameters and their values used for experiments 1, 2, and 3. Table 5.2 lists the fixed parameters and their values for experiment 4. These values are based upon expertise gained during the implementation and ad hoc experimentation with the model.
Table 5.1. Fixed Parameters - Experiments 1, 2, and 3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighborhood Update - MinNeighbors</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Neighborhood Update - Frequency</td>
<td>500</td>
<td>ms</td>
</tr>
<tr>
<td>Neighborhood Update - Discovery</td>
<td>3,000</td>
<td>ms</td>
</tr>
<tr>
<td>Aliveness - Frequency</td>
<td>500</td>
<td>ms</td>
</tr>
<tr>
<td>Aliveness - Lost</td>
<td>7,500</td>
<td>ms</td>
</tr>
<tr>
<td>Movement - DeltaVMean</td>
<td>50.0</td>
<td></td>
</tr>
<tr>
<td>Movement - DeltaVStdDev</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Movement - StepsMean</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Movement - StepsStdDev</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Login Rate</td>
<td>5</td>
<td>per second</td>
</tr>
<tr>
<td>Execution Length</td>
<td>360</td>
<td>seconds</td>
</tr>
</tbody>
</table>

Table 5.2. Fixed Parameters - Experiment 4.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighborhood Update - MinNeighbors</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Neighborhood Update - Frequency</td>
<td>500</td>
<td>ms</td>
</tr>
<tr>
<td>Neighborhood Update - Discovery</td>
<td>3,000</td>
<td>ms</td>
</tr>
<tr>
<td>Aliveness - Frequency</td>
<td>500</td>
<td>ms</td>
</tr>
<tr>
<td>Aliveness - Lost</td>
<td>7,500</td>
<td>ms</td>
</tr>
<tr>
<td>Movement - DeltaVMean</td>
<td>25.0</td>
<td></td>
</tr>
<tr>
<td>Movement - DeltaVStdDev</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Movement - StepsMean</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Movement - StepsStdDev</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Login Rate</td>
<td>5</td>
<td>per second</td>
</tr>
<tr>
<td>Execution Length</td>
<td>360</td>
<td>seconds</td>
</tr>
</tbody>
</table>

The MinNeighbors value of 6 indicates the minimum number of neighbors a peer should keep, if that number are available. The enclosing neighbors computed from the Voronoi diagram typically results in a peer keeping anywhere from six to the neighbors. The more neighbors a peer keeps, the more likely it can correctly recover from an increasing rate of halt failures. Therefore, setting this value to 6 is a conservative choice, helping demonstrate the model’s performance with a small number of neighbors.

The frequency parameter describes how often a peer recomputes who it considers as neighbors, based upon network updates received since the last update computation. A
value of 500 milliseconds means the longest lag time between notification and when a peer is recognized as a new neighbor is a half second. A smaller value increases the discovery rate, but at the expense of greater CPU utilization, while a larger value increases the lag time for neighbor discovery, but reduces CPU utilization. The choice of 500 milliseconds provides an interactive human scale rate of discovery.

The discovery parameter controls the rate at which a peer randomly selects one of its neighbors and asks for all of its neighbors. The purpose of this technique is to help peers auto-correct for possible oversights in neighbor notification. Such oversights are rare, therefore, this parameter is set to 3000 milliseconds.

The aliveness protocol is controlled by the frequency and lost parameters. The frequency parameter controls how soon to start a new aliveness request with a neighbor, following the completion of the last aliveness request. All neighboring peers are sent aliveness requests at this rate. The lost parameter indicates the length of time that must expire before a neighboring peer is considered to have failed the aliveness test. These parameters control how quickly and robustly a neighboring peer is recognized as inactive due to a halt failure of any kind. For this research effort, the values of 500 and 7500 milliseconds work well to demonstrate their contribution to bandwidth, while also helping demonstrate the performance under both message and peer failure conditions.

The rate at which peer processes are created, and therefore, logged in, is 5 per second. At this rate, it generally takes 100 seconds for 400 peer processes to be created and initiate login. This selection was made as a good balance to demonstrate a reasonably high rate of peer login, while also ensuring all peers log in quickly enough to allow more time for scenario execution after all peers have logged in.

The reason it takes more than the expected 80 seconds, is due to real-world computation conditions of the machine sending startup commands to the cluster computers. As the cluster computers have an increasing computational load, due to an increasing number of peers per computer, the time to execute the peer process startup command increases, which results in stretching out the actual rate of peer logins.
Table 5.3. Parameters - Experiment 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movement - Probability</td>
<td>0.20</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Message Failure - Probability</td>
<td>0.00</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Halt Failure - Probability</td>
<td>0.000000</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Number of Peers</td>
<td>40</td>
<td>520</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 5.4. Parameters - Experiment 2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movement - Probability</td>
<td>0.00</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Message Failure - Probability</td>
<td>0.00</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Halt Failure - Probability</td>
<td>0.000000</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Number of Peers</td>
<td>40</td>
<td>1000</td>
<td>40</td>
</tr>
</tbody>
</table>

The execution length of 360 seconds (6 minutes) provides the 100 seconds for all peers to log in, and an additional 260 seconds (over 4 minutes) of additional interaction. This length of time was chosen because it more than demonstrates the behavior of the system following full participation, while being short enough to keep the data collection log sizes within reason; under a terrabyte of data in total for all execution scenarios.

5.5 Scenario Parameters - Variable

This section presents and discusses the parameters that were varied for each of the experiments. Tables 5.3 and 5.4 show the varying parameter settings for Experiments 1 and 2, respectively. Experiment 1 utilized moving peers in a fixed density VE, while Experiment 2 utilized stationary peers in a variable density VE. The reason for the difference in the number of peers between the experiments is due to computational limitations of the cluster computers. Peer movement results in greater CPU utilization, due to the networking demands, than stationary peers. In the case of moving peers, a maximum of 520 peers was possible; in the case of stationary peers, 1000 peers executed comfortably.

Table 5.5 lists the varying parameters for Experiment 3. Varying these parameters with respect to each other resulted in 72 different scenarios.
Table 5.5. Parameters - Experiment 3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movement - Probability</td>
<td>0.00</td>
<td>0.10</td>
<td>0.20</td>
<td>0.30</td>
<td>0.40</td>
<td>0.50</td>
</tr>
<tr>
<td>Message Failure - Probability</td>
<td>0.00</td>
<td>0.01</td>
<td>0.05</td>
<td>0.25</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Halt Failure - Probability</td>
<td>0.000000</td>
<td>0.000417</td>
<td>0.00100</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The probability of movement is the probability per 100 ms. Using a range of [0.0, 0.5] with an increment of 0.10 gives a peer movement rate of [0,5] movements per second. The distance traveled at each time step is controlled by the Gaussian random number selection from the fixed parameters shows in Table 5.1. The maximum movement rate of five steps per second is limited by the computational demands required to run ten peers (on each GuruPlug computer), at the maximum message failure and halt failure rates. Under these conditions, the GuruPlug computers are at 100% utilization.

A probability of 0.50 indicates five movements per second, per peer. For a peer with six Voronoi neighbors, in terms of bandwidth utilization, this represents 30 outgoing and 30 incoming movement messages per second. In spite of the computational limitations, a movement rate of five per second is more than representative of real-world MMO movement rates. In fact, commercial multi-player environments do not notify individual movements, instead, movement vectors or movement commands are transmitted, with occasional absolute position updates to ensure correctness, at rates less frequent than five times per second.

The probability of message failure is the probability per message. The minimum value of 0.00 demonstrates the system under ideal conditions (remembering the executions are performed on a local area network), with a 0.25 failure probability exercising the system in likely unplayable conditions. The probabilities of 0.01 and 0.05 were chosen to represent expected Internet conditions. There is no official documentation or references for Internet-based UDP message failures, given the highly variable nature of the Internet itself. However, developers indicate 1 to 2% as expected, with 10% considered high, but potentially survivable.
The probability of halt failure is the probability per peer, per second that a peer will fail. The three probabilities of 0.000000, 0.000417, and 0.00100 were chosen to show the system under normal and fairly aggressive halt failure rates. Tests of lower failure rates were performed, but the data was so close to ideal there is nothing of particular interest to show. Therefore, higher halt failure rates were selected in order to provide a more interesting, and broader, characterization of the model. Failure rates of 0.000417 and 0.00100 result in approximately 50 and 100 peers, out of 400, fail during a six minute scenario execution, which is far beyond any reasonably expected operating condition.

Finally, Table 5.6 lists the varying parameters for Experiment 4. The first scenario utilized 400 peers, while the second utilized 600 (but reaching a maximum of 550).

The parameters for the first scenario in this experiment were set to demonstrate the system operation under conditions that are more representative of expected conditions. The movement parameters used in the other experiments are quite high, they were chosen to show the system performance under pressure. Therefore, for this experiment, the number and rate of movement was selected according to something more closely representative of participant movement rates, although still relatively high. There is no published data that establishes these rates, movement is unique to each design, and also varies over time with each participant and location within a VE. These values are selected based upon expertise from observation. With respect to message delivery failure over the Internet, there is no typical value, however, various reports place it in the range of 1 to 2%.

The purpose of the second scenario is to show the model under an example of dramatic failure of a large number of peers. The runtime parameters are less important, it is the runtime operation that is the key factor. The scenario began with two phases of 200 peers.

<table>
<thead>
<tr>
<th>Name</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Movement - Probability</td>
<td>0.30</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Message Failure - Probability</td>
<td>0.02</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Halt Failure - Probability</td>
<td>0.000000</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Number of Peers</td>
<td>400</td>
<td>600</td>
<td>na/</td>
</tr>
</tbody>
</table>
logging in. This was followed by near-simultaneous halt failure of 50 peers, randomly selected, throughout the VE. This was followed by another 200 peers logging in. The scenario was terminated in the normal fashion after 360 seconds of time had elapsed.

With as much fun as one can have in research, this scenario was selected to have some fun with seeing just what would happen under and extreme conditions. It is expected that extreme conditions, such as regional power failures, cut utility lines, etc. will impact a massive VE at times. These are out of the normal operation, and exceptional responses can be devised in these events. However, it is still informative to evaluate a system under such circumstances.

It is beyond the scope of this research; only usability testing for a specific VE design can reveal the set of failure conditions that represent playable or unplayable conditions.
CHAPTER 6

RESULTS

6.1 Introduction

This chapter presents the results from the Audrey model implementation, as described in Chapter 4, using the experimental setup detailed in Chapter 5. Selected results are presented and discussed, and the comprehensive set of results is available in Appendix C. Section 6.2 begins with a short review of the data collection and aggregation methods. The core of the chapter is the presentation of the results from the four experiments in Sections 6.3, 6.4, 6.5, and 6.6. The chapter concludes in Section 6.7 with an analysis of the results.

There are far too many possible visualizations, graphical and tabular, of all the raw and aggregated results to present in this dissertation. Those that primarily support the characterization of the system with respect to the research goals are utilized. The essential demonstration of the Audrey model is that the real-world implementation fits within consumer level broadband connections and the overlay remains consistent under expected failure conditions.

It is worth reminding the reader, the results presented in this dissertation are from a real-world, distributed, peer-to-peer system, not a simulation. All scenario executions were performed in real time; one second of execution is one second in the VE, there is no distinction. The effect of having more computers available for execution serves to increase the number of concurrent peers possible in the VE. It has no effect on how quickly an execution scenario can complete.
6.2 Data Collection and Aggregation

For an execution scenario, multiple peer processes are started on each GuruPlug device. For most scenarios the number is 10. At completion, each peer persists its event log to a file on the local computing device. Following execution, a Python script is used to retrieve the log files from all devices used in the scenario, to the main development computer. A typical scenario lasts six minutes, and results in an XML formatted file ranging in size from 5 to 20 MB in size for a single peer; the size is relative to the networking demands of a scenario. With 400 peers used for most scenarios, this generates anywhere from 2 to 12 GB of data per scenario.

As noted in Chapter 4, all data is aggregated and initially summarized using the AudreyViz application. The application provides a batch mode facility that processes all scenarios, computing the consistency and bandwidth measures and writing those results to comma separated files (csv), which are then manually imported into Microsoft Excel for final preparation and presentation.

6.3 Experiment 1 - Bandwidth Scalability for Constant Density

The purpose of Experiment 1 was to evaluate bandwidth scalability over an increasing number of peers while maintaining a constant density of the peers within the VE. Figure 6.1 shows the bandwidth versus an increasing number of peers, for all peers averaged. As noted in Chapter 5, these scenarios had no message failures, no halt failures, and a movement probability of 0.20 per 100 milliseconds. Thirteen different scenarios were executed, beginning with 40 peers, ending with 520, using an increment of 40. The reasoning for beginning with 40, and incrementing in steps of 40 was driven by the 40 GuruPlug computing devices used as the execution platform; at each increment, one additional peer process is executed on each device. The maximum of 520 peers is driven by the computational limitations of the computing devices, being able to execute 13 peers per device under these experimental settings. Following execution, the scenario data were aggregated and a moving average of
bandwidth computed, using a window size of 10, with a bandwidth sample taken at 250 seconds into the simulation. For 520 peers, it takes just over 100 seconds for all peers to complete login and forwarding. Therefore, by 250 seconds, all scenarios have reached the same steady state, which is used for the data sample.

The data in Figure 6.1 show a slight decrease in bandwidth utilization from 40 to 160 peers, thereafter remaining reasonably constant, with some sampling noise. While not proven, a likely reason for the higher initial bandwidth is due to the nature of peer movement and its effect on local density. When a peer hits the boundary of the VE, it reverses direction, in effect, increasing the local density along the bordering region in an area proportional to the peer movement. As the area increases, even as the number of peers increases, the relative effect is that the locally increased density is diluted, resulting in a decrease in bandwidth utilization.
6.4 Experiment 2 - Bandwidth Scalability for Increasing Density

The purpose of Experiment 2 was to evaluate bandwidth scalability over an increasing number of peers while increasing the density of the peers within the VE. Figure 6.2 shows the bandwidth versus an increasing number of peers, for all peers averaged. Twenty-five different scenarios were executed, beginning with 40 peers, ending with 1000, using an increment of 40. As noted in Chapter 5, these scenarios had no message failures, no halt failures, and no peer movement. The reason for choosing no peer movement was to isolate only the bandwidth necessary to form and maintain the overlay, without concern for any particular VE design; peer movement is different for different designs. The maximum of 1000 peers is primarily driven by the data, and secondarily by the limitations of the computing devices. The data show no increase in bandwidth utilization well before 1000 peers, and the devices reach their computational limit somewhere in the range of 1500 peers with these experimental settings.

Following execution, the scenario data were aggregated and a moving average of bandwidth computed, using a window size of 10, with a bandwidth sample taken at 250 seconds into the simulation. For 1000 peers, it takes just over 200 seconds for all peers to complete login and forwarding. Therefore, by 250 seconds, all scenarios have reached the same steady state, which is used for the data sample.

These data show a slight increase in bandwidth utilization from 40 to about 240 peers, thereafter no additional increase in bandwidth is apparent, even as the number of peers increases. These data show very little variation in bandwidth, across all scenarios, as the number of peers increases. This is the expected result, as the number of peer to peer connections is driven by the number of Voronoi neighbors, a value that is invariant with respect to the total number of peers in the system.

Figure 6.2 shows aggregated data; therefore, it is not useful in characterizing the bandwidth cycle for an individual peer throughout its lifetime. Figure 6.3 shows a time series plot for a single peer from the execution scenario with 520 peers. The data shown for this peer are representative of all other peers, in all other scenarios. The shape of the time
Figure 6.2. Bandwidth scalability for increasing density.

series for this peer is characteristic of all peers, and the overlay in aggregate. The time series begins with a short period of high bandwidth demand, continues with a moderately variable demand, followed by a long steady state period.

A peer’s lifecycle begins with server login and authentication, followed by overlay forwarding, continuing with initial neighborhood discovery, then enters a long state of overlay maintenance before final termination. The highest bandwidth utilization is in the neighborhood discovery phase. During this phase, new overlay neighbors are discovered, in which neighbor lists are requested and security certificates are exchanged and validated. Neighbor lists include multiple peer identifiers and network contact details. The certificates include several dozen bytes of identifying information along with an 8K bit signature; a relatively large network packet. The combination of the volume and size of these network messages results in an initial high bandwidth demand during the initial neighborhood discovery phase for an individual peer, and moderate bandwidth demand for neighboring peers currently active and involved in the new peer’s neighborhood discovery phase.
6.5 Experiment 3 - Model Characterization

The purpose of Experiment 3 was to provide a broad characterization of the model and implementation over a wide range of VE design and networking conditions. The majority of the experimental parameters exercised the system beyond expected conditions, but are useful in understanding its operation in stressful conditions.

6.5.1 Ideal Conditions

This section introduces the model under ideal operating conditions, those of no message failures and no halt failures. This serves as a means to introduce the data visualization techniques, along with an introduction to the nature of the results.

Figure 6.4 shows peer bandwidth for peers under conditions of no movement, no message failures, and no halt failures. The chart title shows the key experimental settings for the scenario. The left Y-axis shows bandwidth in bits per second (bps), the right Y-axis shows the number of active peers, and the X-axis is time, in seconds, during the scenario.
execution. Peers attempt to login at the rate of 5 per second; for some results this is attenuated due to computing device computational limitations for some scenario settings.

The shape of this plot is characteristic of all scenario executions; an initial period of high bandwidth utilization, followed by a generally steady-state period of activity. As described in Section 6.4, peers have an initial period of high bandwidth utilization during neighborhood discovery. As a result, when there are a fewer number of active peers, this bandwidth is averaged over a small number of peers. As more peers become active, even as the rate of peer login remains constant at 5 per second, the global effect, due to averaging, becomes less prominent, and this is what is being seen in the first part of all similar plots until peer login is complete. Following the peer login phase, all active peers are either moving or not moving, and the data shows a generally steady-state period.

Figure 6.5 shows consistency for peers under conditions of no movement, no message failures, and no halt failures. The left Y-axis shows the consistency measure, with the chart title, right Y-axis, and X-axis the same as that of the bandwidth plot.
This shape of this plot is not characteristic of most scenario executions. The ideal conditions for this scenario result in perfect consistency once all peers are active, a result that is only characteristic of no motion, which is not typical of most scenarios. The initial scenario phase, peer login, is representative of most scenarios, peer consistency is not perfect. During this phase, some peers are active, but have not yet completed their neighborhood discovery; therefore, they do not report the same peers as the ideal reconstruction suggests.

The next set of bandwidth and consistency results in Figures 6.6 and 6.7 show the system under ideal operating conditions, with a low frequency of peer movement.

The bandwidth plot is substantially similar to the that of no movement, with the notable exception of the increase in bandwidth requirements due to movement messages, which also cascade into messages involving overlay support to maintain the topology due to new neighbor discovery and validation. In other words, the bandwidth increase is due to movement messages and other messages for other protocols that result from peer movement.
Figure 6.6. Peer bandwidth - Slow movement.

Figure 6.7. Peer consistency - Slow movement.
The consistency plot, on the other hand, takes on a new, and more typical, appearance. In this case, the average peer continues to maintain perfect consistency; however, at higher percentiles perfect consistency is not maintained. This is an expected result. Given that peers are in motion and the lag time involved in notification and response, it is impossible, regardless of ideal operating conditions, to maintain perfect consistency. For the scenario executions used for this research, a peer updates its neighbors once every 500 milliseconds (twice per second). During this update, the peer may recognize the need to request neighboring peers to fill in a possible gap in its knowledge. There is some network and computational latency involved in sending and receiving that request, along with another possible 500 milliseconds before the set of neighbors is updated again. Therefore, it can easily take over 1000 milliseconds (one second) to return to perfect consistency, as measured by an ideal global reconstruction. Furthermore, during those 1000 milliseconds, based upon peer movement and additional notifications, the peer may still not be perfect with respect to the ideal global reconstruction. The ideal global reconstruction is unforgiving because it presents an impossible view of no network latency and no computational latency. Regardless, it suffices as an effective benchmark.

Figure 6.8 provides further insight into the consistency measure. The left Y-axis is the percentage of peers, the right Y-axis the number of active peers, and the X-axis time. Each shaded area shows the cumulative percentage of peers for each consistency class as indicated by the legend. This plot shows the percentage of peers that are perfect, and are incorrect by 1, 2, 3, or more peers. In the case of this scenario, more than 50% of the peers throughout the scenario are perfect, with the majority of the peers differing from the ideal by 1, and around 10% differing by 2 or more peers.

Finally, Figures 6.9 and 6.10 show combined results for all motion probabilities (0.0, 0.1, 0.2, 0.3, 0.4, and 0.5), no message failures, and no halt failures, for bandwidth utilization and consistency. The results in these plots show expected results of increased bandwidth demands and decreased consistency as the probability of movement increases.
Figure 6.8. Missing peers - Slow movement.

Figure 6.9. Bandwidth - No message or halt failures.
6.5.2 Failure Conditions

This section presents the implementation of the Audrey model under various conditions of failure. The two parameters of failure used in this experiment are message failure and halt failure. As noted in Chapter 5, most of these results are beyond expected normal operating conditions.

Figure 6.11 shows bandwidth utilization for the scenario with no movement, message failure probability of 0.01, and halt failure probability of 0.00417. Figure 6.12 shows consistency for the same scenario. As compared to the scenario with no message or halt failures, there are two differences. The first is a slight increase in bandwidth due to messaging timeout and retries, but the most interesting difference is the spikes in bandwidth due to halt failures. Figure 6.13 shows the details of which peers remained perfect or were missing peers, and how many, during periods of halt failure.

When a peer fails due to halt failure, it stops accepting or sending all network messages. When this happens, protocols for neighboring peers begin to experience message failures.
Figure 6.11. Bandwidth - Low message and halt failures.

Figure 6.12. Consistency - Low message and halt failures.
and eventually failures of the protocol as a whole. One of these protocols is the aliveness protocol, the means through which peers determine halt failures of neighbors. When this protocol fails, a peer re-enters a neighborhood discovery phase, requesting neighbors from all of its neighbors; this also occurs simultaneously for all neighbors of the failed peer. The bandwidth spikes that coincide with failed peers show a large localized bandwidth demand averaged into all peers. A closer inspection of the plot shows the average, 50th, and 75th percentiles display a small global effect, whereas the 95th percentile is more effective in showing the localized bandwidth effect. The consistency results demonstrate a short term drop in consistency upon peer failure, with a quick return to full consistency resulting from the failure recovery.

Figures 6.14, 6.15, and 6.16 provide an overview of a scenario that involved moderate movement, message failures, and halt failures. As expected, these results show an increase in bandwidth utilization and decrease in consistency, due to the combination of movement and failure conditions.
Figure 6.14. Bandwidth - Movement, message, and halt failures.

Figure 6.15. Consistency - Movement, message, and halt failures.
Figures 6.17 and 6.18 compare the results for all movement probabilities for the scenario conditions that include a message failure probability of 0.05 and halt failure probability of 0.000417. These data show the expected pattern of increased bandwidth utilization and decreased consistency.

Finally, Figures 6.19 and 6.20 show combined results for scenarios involving extreme probabilities of failure and halt conditions, conditions well beyond any expectation for any VE. As usual, these data show an increase in bandwidth utilization and decreased consistency.

In spite of the extreme operating conditions, there is good news to be gleaned. The first is that in all conditions, bandwidth utilization continues to lie well within residential broadband limits, far below video streaming, which is measured in Mbps, instead of these data, in Kbps. Secondly, for VEs involving static peer positions, or peers with low movement rates, consistency remains high. Additionally, at no point does the model show a complete breakdown; the overlay topology continues to be intact. Therefore, even given...
Figure 6.17. Bandwidth - Combined message and halt failures.

Figure 6.18. Consistency - Combined message and halt failures.
highly stressful real-world operating conditions, the Audrey model can maintain its topology and ability to recover to an acceptable state following any external circumstances that may lead to similar high failure conditions.

6.6 Experiment 4 - Selected Scenarios

The purpose of the scenarios in this experiment was to show the model under one condition more representative of potentially expected conditions and another of dramatic, high peer failure. Neither of the scenarios fit nicely into the other three experiments; therefore they are grouped as a separate experiment for presentation.

6.6.1 Expected Conditions

This scenario shows the model operation under conditions of expected message failure and representative movement rates. Figures 6.21 and 6.22 show bandwidth utilization and consistency, respectively.
Figure 6.20. Consistency - Combined message and halt failures.

Figure 6.21. Bandwidth - Scenario 1.
The average bandwidth utilization, once all peers are active, remains below 100 Kbps, with even the 75th percentile running close to the 100 Kbps line. The high end bandwidth, the 95th percentile, shows consistent utilization well below 200 Kbps. All of these are values that easily fall within current broadband limits.

Consistency shows similarly positive results, with the average closely following 0.9, and the 50th percentile maintaining a fairly consistent 1.0. A look at Figure 6.23 provides additional insight into the system behavior, with respect to how well peers maintain consistency. The data show that 50% of the peers maintain perfect consistency, with the next 35% to 40% inconsistency by only a single peer, with the remaining minority inconsistent by 2 or more peers.

6.6.2 High Failure Condition

This scenario shows the model operation under a condition of sudden, widespread, peer failure. Consistency is the main concern in this scenario, therefore that is the focus of
the discussion. Figures 6.24 and 6.25 show consistency and the number of missing peers, respectively.

The data in Figure 6.24 starts off the same as that shown for Scenario 1 in this experiment, with consistency at, or above 0.9. This continues until about 140 seconds into the scenario, when 50 peers (out of 400), where near-simultaneously terminated. At this time, consistency drops by about the same percentage as the percentage of peers that terminated. Recovery takes only a short time, around 15 seconds, with consistency returning to near pre-failure conditions. Beginning at 190 seconds into the scenario, an additional 200 peers join the overlay.

A similar pattern of consistency behavior is seen in the missing peers data, Figure 6.25. However, the divergence in consistency is more pronounced, with a greater number of post-failure peers showing inconsistencies versus those of the pre-failure conditions.

The likely cause for the difference in the pre- and post-failure consistency results is due isolation of peers. When the sudden failure event was induced, a small number of peers
Figure 6.24. Consistency - Scenario 2.

Figure 6.25. Missing Peers - Scenario 2.
lost all of their neighbors; leaving them without means to recover. Then, when new peers
logged on following the failure event, this served to only make the situation appear worse,
due to the global reconstruction not being aware of the isolated peers.

There is an easy design solution that allows for recovery, but not it is implemented as
part of this research project. If a peer becomes isolated by losing contact with all of its
Voronoi neighbors, it should contact the server and ask to be forwarded back to its current
VE location. In doing this, the peer returns to its last location, while also discovering its
new neighborhood following a failure event.

6.7 Analysis

Two key measures were put forth as the benchmark for evaluating the Audrey model,
bandwidth utilization and overlay consistency. Bandwidth is a critical measurement because
of clear limitations for residential broadband. Consistency indicates whether or not the VE
topology is correctly maintained both globally and locally, while also suggesting the kind
of designs appropriate for a hybrid P2P model.

With respect to bandwidth utilization, the results from the model implementation show
that under all scenarios presented, bandwidth remains under residential broadband limits.
Therefore, the model is highly successful in that regard. According to expectations, the
model shows an increase in bandwidth utilization as peer movement within the VE increases.
Similarly, an increase in bandwidth utilization is seen with an increase in message failure
rates. Halt failure also shows a local neighborhood increase in bandwidth utilization.

The bandwidth utilization results clearly show the tradeoff being made from the elim-
ination of a server that coordinates all participant interaction, to that of the participants
collaborating to maintain the VE; an increase in bandwidth demand pays for the elimination
of server coordination.

It is more difficult to evaluate success or failure of overlay consistency. There are
no existing benchmarks against which consistency can be compared, in order to obtain
external validation. Until a real-world, interactive, P2P-based system is deployed, how
to comprehensively interpret consistency results with respect to user acceptance of a VE
will remain an open question. In spite of this current shortcoming, there is still useful information that can be gleaned.

The consistency measure, combined with the missing peers data provide insight into the nature of the consistency numbers. In the vast majority of scenarios, greater than 50% of the peers have no missing peers, with the next 30% to 40% off by a single peer, the remaining minority inconsistent by more than 1. As noted earlier, there is no expectation for all peers to ever report 100% consistency due to the nature of the global reconstruction, movement, network latency, and computational latency in the system. Given this, the results from these scenarios are encouraging in suggesting a robust system.

With respect to the results from Experiment 3, Section 6.5, a baseline expectation for consistency is set for peers under constant motion. Whether or not this is acceptable for gameplay is a question that can only be answered by a particular VE design; a VE can be designed according to a chosen level of consistency. However, it does provide a relative basis for considering consistency results under failure conditions, and a basis for future algorithmic improvements.

Consider Experiment 4, the condition wherein the system faced a dramatic halt failure of peers; greater than 10% of the peers simultaneously failed. The consistency results demonstrate two insights. The first is that the system did not return to pre-failure consistency, demonstrating that some peers became isolated. Secondly, this same result demonstrate the overlay almost returned to pre-failure consistency, differing by a few percent, hence demonstrating a fairly robust system. Also noted above, a simple change in peer behavior upon detection of the loss of all neighbors would result in a system that returns to pre-failure conditions.
CHAPTER 7

CONCLUSIONS

7.1 Introduction

The research and work presented in this dissertation provides the basis for continuing research; it is a beginning, not an end. The primary contribution of this research is the introduction of a new hybrid P2P model, with a specific focus on a real-world implementation. The widespread use of P2P techniques in massive VEs is still many years away, but the results from this research suggest a concrete pathway to follow.

7.2 Contributions

The model presented in this dissertation is an important step forward towards the development of P2P-based MMO frameworks. A hybrid model has been defined and validated through a real-world implementation that supports the scalable construction of a secure VE involving massive numbers of participants. Specifically, the contributions of this research include:

- Hybrid P2P-based massive networked virtual environment model.
- Demonstration of a real-world implementation of the model.
- Demonstrated bandwidth scalability versus an increasing number of participants.
- Demonstrated overlay topology maintenance under expected and stressful operating conditions.
- PGP-based security scheme that allows participants to authenticate without requiring contact with the server.
• Messaging benchmark that can be used to evaluate the effect of network messaging schemes on individual peers and the global overlay.

7.3 Future Work

While the Audrey model provides a P2P-based MMO framework, it is only the first step, much work remains. The current model enables massively multiplayer VEs, but does not yet enable the kinds of VEs most popular in today’s landscape, that of massively multiplayer role playing games (MMORPG). MMORPGs have highly sophisticated VE designs, supporting a high level of participant to participant interaction, group interactions, participant to environment interactions, complex environments, and long term persistence. Before a hybrid P2P system can match the VEs offered by current client-server designs, further research is necessary. This section offers an overview of research necessary to move hybrid P2P systems in that direction.

7.3.1 Expanded Analysis and System Visualization

This research focuses on the validation of the Audrey model and implementation by characterization through consistency and bandwidth. There are many other perspectives to further characterize and analyze the system, without concern for additional features. For example, consistency is reported with the number of peers that reported the same set as the ideal, along with the number of peers that differ in their reporting by 1, 2, 3, or more peers. An expanded analysis should consider the length of time a peer remains perfect as compared to the ideal, along with the length of time it differs by 1, 2, 3, or more peers. Such an analysis may identify a weakness requiring changes to the model, along with providing a VE designer an acceptable bound within which a design must fit.

A relative minority of system parameters were varied for this research effort. An expanded analysis should evaluate the sensitivity of the system to other parameters. The minimum number of neighbors kept, how often aliveness checks are performed, the number of retires on timeouts, the period for timeouts, etc. The setting of these parameters for this work was based upon researcher expertise, but would benefit from a thorough analysis to
determine their best settings, under different environmental conditions.

Along with any expanded analysis, more sophisticated data visualization is important. Additional spatial visualizations of the data are necessary. An animated heatmap, using a moving window, of messages would be a useful analysis tool to understand the flow of data throughout the system. Such a tool will allow a researcher or designer to more thoroughly analyze system behavior both globally and locally. For example, upon halt failure of a peer, the local neighborhood surrounding the peer failure could be visualized and analyzed. Such an analysis might offer insight into algorithmic improvements to reduce detection lag, bandwidth utilization, and increase consistency during the failure.

7.3.2 Object Persistence

The next step for this research work is to define and validate an object persistence model. VEs are composed of not only individual participant controlled characters, but also objects that have a VE presence, state, and lifetime that is independent of the participants’ virtual presence. For example, a VE can have buildings. The position of the building is not player controlled, and its position is independent of any active participant. Additionally, the entrances to the building have a state that can be changed through participant interaction. These states must be maintained regardless of the locality or lifetime of any particular participant.

Object persistence can be loosely categorized into three levels: short, medium, and long term. Objects having short term persistence are typically associated with interactions between VE participants. Medium term persistence is generally associated with objects under a single participant’s control, items defined by the VE design and used for VE interaction or interaction with other participants. The lifetime of these objects, within the VE, is similar to the lifetime of a participant. Finally, many objects have a lifetime completely independent of any participant, such as the building example from above. The persistence
strategy for each of these levels of persistence is likely to differ, owing to different levels of reliability, activity, security, and interaction concerns. Continued research in this area should begin with objects having short term persistence, then increasing the sophistication by continuing through those requiring long term persistence.

7.3.3 Non-Player Characters

Closely related to the topic of object persistence is non-player controlled characters (NPC). An NPC refers to a VE participant that is not under the control of any player. An NPC’s actions are programmed, either as scripted events in response to environmental triggers, or through more sophisticated artificial intelligence techniques. As the Audrey model requires, the simulation of the VE must not take place at the server. Therefore, the simulation and object ownership of an NPC must be coordinated among the active peers. Universally, role playing style MMOs involve large numbers of NPCs, easily equaling or exceeding the number of player controlled characters.

In order to appeal to MMORPGs developers, it is essential a Hybrid P2P system allow for NPCs. The research into such techniques must follow that of object persistence, with at least a medium term object persistence model necessary to support a scheme for NPCs.

7.3.4 Messaging

An important aspect of any VE is the ability for participants to interact through messaging, either on a one-to-one basis, or group messaging (e.g. one-to-many messaging within a group). Other kinds of messaging are necessary, in the form of system broadcasts, or region casts. Following on the theme of scalability, these messaging systems must not rely upon the server for message dissemination to all other peers. Instead, the messaging must be coordinated among the active peers.

P2P messaging between two participants is relatively straightforward, as the current model already provides a direct P2P communication scheme once peers have made initial contact. Messaging within a small group, such as a part, or team, is similarly straightforward
in the form of a one-to-many broadcast; as long as the group size is relatively small (less than a few dozen), bandwidth utilization is not a serious concern.

A system wide broadcast is more complex, as neither the server or any peer has global knowledge of all active peers. For example, if the server needs to send a message to all active peers informing participants of an event, or status update, the server itself cannot reasonably send the message to every peer itself, primarily out of scalability concerns. Therefore, a broadcast message must have its origin in a single peer, or a small subset of all peers, with that message propagated throughout the system through P2P coordination.

7.3.5 Crowding

While not specifically addressed in this dissertation, crowding is a problem with the Voronoi-based overlay approach. Crowding is when a large number of participants gather closely in some VE spatial region. For example, many commercial VEs have cities that act as natural gathering points for large numbers of participants to meet and interact. The current Audrey model cannot support such a design due to bandwidth demands. The model allows a large crowd to be spatially close, but each participant only sees a few-dozen other participants at best, rather than 50 or 100 or more.

A significant improvement is to develop an extension to the overlay structure that enables the visibility and interaction with a greater number of other VE participants. This may come in the form of an improved communication scheme, or a replacement of the overlay organization structure itself, perhaps using a different data structure than the Voronoi diagram.

7.3.6 Complex Virtual Environments

Another area of research that can offer significant benefits in terms of bandwidth utilization improvement and VE design is to incorporate the use of VE geometry. The current Audrey model says nothing about VE geometry, neither precluding its use, nor taking advantage of it in any manner. Two peers close to each other in the VE but occluded by some VE geometry may not necessarily need to be considered as Voronoi enclosing, or as AOI
neighbors. A model extension that does, may allow for a larger number of peers in a VE region by reducing, or eliminating, unnecessary communication among peers.
REFERENCES


[52] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM ’01:


APPENDICES
Appendix A

Messaging Benchmark

A.1 Introduction

Communication between peers is a key concern in P2P-based Networked Virtual Environments (NVE). In such systems no peer knows about all other peers; therefore, a message forwarding scheme that overcomes this challenge is necessary. There are many different P2P NVE designs, with each impacting the performance of a messaging scheme differently. To date, the only reported evaluation technique of these schemes is the number of hops a message takes to arrive at its destination – an insufficient measure.

Client-server designs have a relatively simple communication scheme. A client sends a message, destined for another client, to the server. Because the server has a direct connection to every client, it sends the message directly to the destination client. All client-server systems share this same basic design, resulting in no differentiation in communication performance.

P2P systems differ significantly from client-server systems in the formation of their network overlay, resulting in differing messaging performance. In a client-server system, the number of connected clients has no impact on the number of hops between any client. P2P network overlays, on the other hand, change with every peer, which connects or disconnects. Additionally, some P2P network overlays change structure as peers change position within the virtual environment (VE). P2P systems are far more complex in their communication structure than client-server systems, and therefore, demand a more sophisticated evaluation basis.

To further illustrate the issue, consider a P2P design and messaging scheme that results in a peer, or peers, being overwhelmed with message forwarding requests. For a content
distribution network, this is a relatively minor inconvenience for the users. On the other hand, for an interactive Massively Multiplayer Online (MMO) system, overwhelming the bandwidth of a peer negatively impacts a user’s experience. This may result in that user disconnecting from the network, with the problem moving to another peer and cascading as the problem persists. Using the number of hops as the only evaluation criteria, the problem remains hidden until too late.

Section A.2 provides an overview of techniques others have used to evaluate message forwarding. The performance metric is detailed in Section A.3. Section A.4 describes the context in which the performance metric was originally developed. The experimental setup is presented in Section A.5. The results from the simulation experiments are discussed in Section A.6 and closing remarks in Section A.7.

A.2 Related Work

In this section we review the message forwarding choices of representative P2P systems, along with the reported performance basis used in their evaluation. Put simply, performance evaluation of message forwarding schemes has not been properly addressed; therefore, little work exists.

Two projects under the name of Solipsis have been published [24, 39]. Both solutions rely upon a greedy message forwarding scheme, with neither paper presenting a basis for evaluation. Similarly, the VON framework [32] utilizes a greedy forwarding scheme. The authors do not individually evaluate this scheme; instead, any performance impact is aggregated into overall communication bandwidth performance.

The most common P2P messaging scheme employed by massive P2P NVE systems is Pastry [48]. Upon joining a network, Pastry assigns a randomly selected 128-bit identifier to each peer. Based upon this identifier, other peers are able to use a distributed hash table (DHT) algorithm that allows peers to send messages between each other within $O(\log_2 N)$ hops, where $b$ is a configurable parameter, typically 4. Rowstron et al. evaluate the performance of Pastry exclusively through the use of the number of hops as compared to the number of nodes.
The Peer Clustering prototype [17] uses a Pastry-based message forwarding scheme, with the authors reporting performance in terms of number of hops. Knutsson et al. also used a Pastry-based scheme [40], again reporting performance in terms of the number of hops between network peers. Another scheme proposing to use Pastry is Mediator [23]. Because the paper is a proposal, there is no presentation of messaging performance.

Dickey et al. present an event ordering technique using N-Trees [26]. Event ordering relies upon messaging between peers in order to resolve the ordering. The performance measure used to evaluate the cost of messaging in this scheme was number of peers in the network versus number of messages required.

### A.3 Performance Metric

The performance of a messaging scheme is evaluated through the aggregation and summarization of data from messaging throughout the network, rather than for any single message. In other words, a messaging scheme is evaluated by sending many (thousands) messages throughout a network, with the results of those messages summarized into several performance measures. The metric is composed of the following measures:

1. Number of Hops Average/Median
2. Number of Hops Variance
3. Local Bandwidth Max
4. Local Bandwidth Average/Median
5. Local Bandwidth Variance
6. Global Bandwidth
7. Spatial Bandwidth Max
8. Spatial Bandwidth Variance
The number of hops a message takes is important because it is a proxy for how long a message takes to arrive at its destination. The average number of hops indicates the expected time to send a message, within the measured variance.

Local bandwidth indicates the bandwidth expectation at a peer. The Max value is the highest bandwidth usage by a single peer. The average, median, and variance are computed across all peers.

The peer with the maximum bandwidth demand is necessary in order to recognize the potential for demanding higher bandwidth at a peer than its expected available resources, potentially creating a highly negative user experience. The average, median, and variance values indicate whether or not the messaging scheme is appropriate for the expected bandwidth resources available at a peer. The variance additionally indicates the fairness of the scheme. A scheme with a lower variance indicates the scheme requires similar resources from all peers. A higher variance indicates the scheme favors some peers over others, creating the potential for some peers to have an advantage because their networking demands are lower than others. The median is important because the data from messaging schemes is not guaranteed to have a normal distribution. In these cases, the median bandwidth might be a better indicator of expected bandwidth demands.

Global bandwidth is the total number of hops taken for all messages recorded during the evaluation period.

Spatial Max and Variance are computed by subdividing the VE region into smaller square regions and aggregating results within each of these smaller regions. As a message is processed, the spatial location at which the processing occurred is recorded and added to the results for the subdivided region. The variance is computed over all the subdivided regions. For example, divide VE region into a grid of 100 x 100 smaller regions, creating 10,000 spatial regions in which data is collected.

The purpose of the Spatial Max and Variance is to reveal a spatial bias of the messaging scheme, if any. Whereas one scheme might not have a spatial bias, another may. A high spatial variance indicates the scheme exhibits a spatial bias. These values are computed
by recording and binning messages based upon the VE location of the peer at the time they were logged. It is not possible to specify the number of bins for any arbitrary VE; the expertise of the developer is still required to make a proper choice. While not specified in the metric, we additionally use a heatmap, which enables us to visually identify the nature of the spatial bias, if any. A messaging scheme that exhibits a spatial bias may lead to unintended social behaviors within the VE. As participants notice greater resource demands due to spatial locality, they will tend to avoid those locations, perhaps introducing additional performance problems with the scheme.

The number of messages, assuming messages are similarly sized, is a valid substitute for bandwidth.

A.4 Login Forwarding

The context of the performance metric presented in this paper is the evaluation of login forwarding techniques for our hybrid P2P NVE design, Audrey. Audrey is a Voronoi-based NVE, designed to host Massively Multiplayer virtual environments [3]. The framework includes a managed server, which is used for peer login and validation. As a peer joins the network overlay, it goes through several states before becoming an active participant. One of these states is known as login forwarding.

The login forwarding state involves a protocol through which a peer is forwarded to the correct overlay neighborhood, based upon its starting position in the VE. Login forwarding enables the joining peer to discover those neighbors with which it should be initially connected. The protocol begins with the joining peer contacting the managed server for forwarding. The server responds by sending contact details of an active peer to contact for further forwarding. The joining peer contacts this active peer to continue forwarding. This process repeats until the active peer closest to the joining peer’s destination is discovered. Fundamentally, login forwarding is a messaging scheme.

**Greedy Forwarding:** A joining peer sends a forwarding request to another peer, the receiving peer. Upon receipt of a forwarding request, the receiving peer examines all its known neighbors, both Area of Interest (AOI) and enclosing, to find the one closest to the
join destination, including the receiving peer itself. If the receiving peer is closest to the join destination, the joining peer is notified and the forwarding is complete. Otherwise, the contact information for the neighboring peer closest to the requested destination is sent to the joining peer. The joining peer continues the forwarding process by contacting the newly identified peer closest to its join destination.

Figure A.1 illustrates a greedy forwarding sequence originating at the server peer and ending with a peer in the upper left corner of the virtual environment. The sequence begins with the server peer’s Voronoi region highlighted, indicating it is the next receiving peer. The next step shows the enclosing (light grey) and AOI (dark grey) neighbors. From these neighbors, the one closest to the join location is selected as the next receiving peer; its Voronoi region is highlighted in the third step. The remaining steps illustrate the rest of the greedy forwarding sequence.

Because the purpose of Audrey is to enable massive peer participation, an efficient login forwarding protocol is needed. The naive approach to login forwarding is to use pure greedy forwarding, beginning at the server. As will be shown through our performance metric, this is also a poor choice. We identified several candidate techniques to improve upon pure greedy forwarding: three working set techniques and a grid based technique. Additionally, we included two techniques, FIFO and Random Selection, to help validate the effectiveness of the performance metric. The following list identifies these techniques, with the sub-sections that follow detailing each.

1. Best Case
2. Pure Greedy
3. First In, First Out (FIFO)
4. Random Selection
5. Working Set Random Replacement
6. Working Set Recent Replacement
For all techniques, once the initial peer is identified, greedy forwarding is employed to complete the join operation. The differentiating feature between each is the identification of the first peer to which the joining peer is handed off to begin greedy forwarding. Because the framework design is a P2P network, no single peer, including the server itself, has global knowledge of all active peer current locations. Therefore, the key to the best performance is to start the greedy forwarding process with the active peer as close to the destination as possible.
**Best Case**

Forwarding begins with the peer whose current location is closest to the destination of the joining peer. This provides the best possible selection. This is impossible in a real-world hybrid P2P deployment, because the server does not know the current location of all actively participating peers. However, under simulation conditions, it is possible to have global knowledge of the P2P overlay.

The concept is to provide a basis for evaluating how well any other variations approach the best case.

**Pure Greedy**

The design of Audrey specifies a bootstrapping peer, the server peer, located at the center of the virtual environment. This peer has no virtual environment presence; its purpose is to provide the startup/fallback peer for the construction and maintenance of the P2P overlay. This variation specifies the server peer is selected, every time, as the node from which the greedy forwarding process begins; in other words, pure greedy forwarding.

The concept is that of a naive approach to handle login forwarding, without regard for efficiency or fairness.

**First In, First Out**

A first in, first out queue of active peers is maintained at the server. As a peer makes a forwarding request with the server, the peer at the front of the queue is selected as the starting peer for the greedy forwarding process. Once a peer completes forwarding, it is added to the end of the queue.

The concept is that of fairness of resource usage. Each peer must provide the same service it consumed for the next peer that joins the overlay. Fairness is emphasized over efficiency.
Random Selection

Forwarding starts by selecting a random peer from the set of known active peers.

The concept is that of fairness of resource usage, randomly distributing forwarding requests throughout all active peers. Fairness is emphasized over efficiency.

Working Set Random Replacement

The server maintains a fixed size set of active peers, the working set. The number of peers in the working set is relatively small, proportional to the total number of active peers. As a joining peer requests forwarding, the peer in the working set with its last known position closest to the joining peer’s destination is chosen as the starting peer. The peer then selected to start the forwarding is removed from the working set and replaced by random selection from all known active peers. The number of peers in the working set is fixed throughout the lifetime of the server.

The concept is that of efficiency, with a secondary consideration with respect to fairness. Computational efficiency is considered by keeping a working set that is fixed in size and relatively smaller than all known active peers. Instead of testing every peer, a small number of peers are evaluated, ensuring a small, constant response time, even as the number of active peers increases. Efficiency with respect to global bandwidth is considered by choosing the peer with the last known position closest to the forwarding destination, the intention being to reduce the number of greedy forwarding requests required to join.

Working Set Recent Replacement

The technique has the same working set concept as described in Working Set Random Replacement, with a differing peer replacement scheme. The replacement peer is selected by choosing the peer that has most recently become active. The number of peers in the working set is fixed throughout the lifetime of the server.

The concept in choosing the most recent active peer is that it is most likely closer to its starting location than any peer selected at random from all active peers. By choosing the most recent active peer, the replacement is in a similar location to the one replaced; this
peer will also have the best, last known active location among all peers in the overlay. As this strategy is employed, the working set will contain peers with the most recent known active locations, distributed throughout the overlay.

**Working Set Proportional**

This is a variation on the Working Set Recent Replacement, differing in how the size of the working set is determined. The fixed sized working set is replaced by two parameters that control the size of a dynamically sized working set: 1) A minimum number of peers in the working set and 2) A maximum number of peers proportional to the number of active peers. The minimum specifies the smallest size the working set can ever be (given that number of active peers), while the maximum size changes in proportion to the number of active peers.

The concept is to grow and shrink the working set proportionally with the number of active peers, thereby dynamically changing the scope of the peers chosen from which to begin the greedy forwarding process.

**Grid Recent Replacement**

The server subdivides the virtual environment into a uniform grid of cells, identifying one peer for each of the cells from which the greedy login forwarding process begins. As a joining peer requests forwarding, the cell corresponding to the destination is computed and the peer within that cell is chosen as the starting peer. When a peer notifies the login server it has become active, the cell into which it belongs is computed and it becomes the forwarding peer for that cell until it is eventually replaced. Initially, the grid is populated with the server peer as the forwarding peer for each of the grid cells. As new peers become active, they replace the previous peer for their cell location. This creates turnover in the cells, helping to ensure the peer with the best last known position is represented within the grid. Therefore, the peer at all cell locations is the peer with the best last known position of any peer within that cell area. Two parameters control this variation: 1) The starting
size of the grid and 2) The threshold that causes the grid to increase in size. The grid is initialized to some size, for example, a 2x2 grid. As the number of active peers increases, the size of the grid also increases, thereby spreading out the distribution of peers from which forwarding can begin. For example, when the size of the grid is increased from 2x2 to 4x4, the peer at cell [0,0] from the 2x2 grid is replicated into cells [0,0], [1,0], [0,1], [1,1] in the 4x4 grid. Over time, as new peers become active, they replace and create unique peers in the new grid.

The concept is to ensure a uniform distribution throughout the virtual environment of peers from which the greedy forwarding begins.

A.5 Experimental Setup

Simulations were performed to collect data in order to compare each of the techniques, using both fixed and dynamic AOI. Table A.1 identifies the simulation parameters. The choice of 5,000 time steps was guided by previous work by Hu et al. [32], where 3,000 time steps were used. In evaluating the usefulness of longer simulations for these techniques, some simulations were run with much longer time steps (over 30,000), which produced no difference in the results. Therefore, 5,000 was selected as having a proper balance of a long enough simulation to collect valid results, while providing short enough computation time to repeatedly run simulations. The choice to use 1,000 VE units and 10 max peers for the fixed and dynamic AOIs was guided by identifying parameters that show the differentiation between fixed and dynamic AOI. Finally, the choice to have 1 peer arrive every 5 time steps was made to ensure enough peers (1,000) joined the simulation to simulate a large number of active peers in the NVE.

At each time step, the simulation counts the number of active login forwarding messages contained within each peer’s message queue; these data are used to compute the number of hops, local bandwidth, and global bandwidth measures. The VE is divided into a 100 x 100 grid of bins. At each time step, the number of login forwarding messages for all peers within that bin is recorded. These data are used to compute the spatial performance measures. The number of neighbors tracked by each peer is a parameter that significantly
affects the performance of login forwarding. This number is controlled by the AOI, of which two approaches are utilized, fixed and dynamic. Using fixed AOI, a peer tracks all neighbors within a fixed, circular region. Using dynamic AOI, a peer tracks a fixed number of peers regardless of their distance, with the circular AOI region defined by the distance to the furthest neighbor. For both approaches, all enclosing neighbors are tracked.

The simulated virtual environment is a 10,000 by 10,000 unitless rectangular region. As a peer joins the simulation, its joining location within the VE is determined by the server using uniform random selection. The movement of each peer is also randomly determined. Initially, a random direction vector is selected, along with a randomly selected speed, and a randomly selected length of time for which the peer will move at that speed along the vector. Once the time period for that movement is complete, a new direction, speed, and length of time is selected; this is repeated for the lifetime of a peer.

Fixed AOI is not a scalable solution in a P2P environment due to the non-scalable number of messages required to maintain a P2P overlay. The number of messages required to support the overlay grows combinatorially with the number of neighbors within a peer’s AOI. When using a fixed AOI, this number easily becomes a problem for network bandwidth utilization. The reason for showing fixed AOI results is to help understand the performance of dynamic AOI in comparison.

Table A.1. Simulation Parameters.

<table>
<thead>
<tr>
<th>AOI</th>
<th>AOI Range</th>
<th>Time Steps</th>
<th>New Peer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>1,000 units</td>
<td>5,000</td>
<td>1 @ 5 steps</td>
</tr>
<tr>
<td>Dynamic</td>
<td>10 peers</td>
<td>5,000</td>
<td>1 @ 5 steps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Working Set Parameters</th>
<th>Technique</th>
<th>Working Set</th>
<th>Min Size</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>20</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Recent</td>
<td>20</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Proportional</td>
<td>n/a</td>
<td>10</td>
<td>0.05</td>
<td></td>
</tr>
</tbody>
</table>
Table A.2. Fixed AOI - Number of Hops.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average</th>
<th>Median</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case</td>
<td>8.59</td>
<td>8.00</td>
<td>2.27</td>
</tr>
<tr>
<td>Pure Greedy</td>
<td>24.24</td>
<td>25.00</td>
<td>8.81</td>
</tr>
<tr>
<td>FIFO</td>
<td>30.54</td>
<td>29.00</td>
<td>13.84</td>
</tr>
<tr>
<td>Random</td>
<td>28.91</td>
<td>29.00</td>
<td>13.93</td>
</tr>
<tr>
<td>WS Random</td>
<td>16.22</td>
<td>17.00</td>
<td>7.22</td>
</tr>
<tr>
<td>WS Recent</td>
<td>13.57</td>
<td>13.00</td>
<td>5.18</td>
</tr>
<tr>
<td>WS Proportional</td>
<td>13.29</td>
<td>13.00</td>
<td>5.14</td>
</tr>
<tr>
<td>Grid Recent</td>
<td>14.87</td>
<td>13.00</td>
<td>6.55</td>
</tr>
</tbody>
</table>

Table A.3. Dynamic AOI - Number of Hops.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Average</th>
<th>Median</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case</td>
<td>8.66</td>
<td>8.00</td>
<td>2.52</td>
</tr>
<tr>
<td>Pure Greedy</td>
<td>29.05</td>
<td>29.00</td>
<td>12.84</td>
</tr>
<tr>
<td>FIFO</td>
<td>36.23</td>
<td>22.00</td>
<td>18.56</td>
</tr>
<tr>
<td>Random</td>
<td>36.29</td>
<td>33.00</td>
<td>19.53</td>
</tr>
<tr>
<td>WS Random</td>
<td>18.39</td>
<td>17.00</td>
<td>9.59</td>
</tr>
<tr>
<td>WS Recent</td>
<td>16.58</td>
<td>13.00</td>
<td>8.25</td>
</tr>
<tr>
<td>WS Proportional</td>
<td>14.85</td>
<td>13.00</td>
<td>6.08</td>
</tr>
<tr>
<td>Grid Recent</td>
<td>18.36</td>
<td>17.00</td>
<td>10.68</td>
</tr>
</tbody>
</table>

A.6 Results

Tables A.2 through A.6 report the performance metric measures from the simulations. Tables A.2 and A.3 report number of hops for the messages. Tables A.4 and A.5 report local and global utilization. Finally, Table A.6 reports the spatial measures.

In addition to the simple measures of spatial maximum and variance, we have created heatmaps based upon the VE location of peers at the time the messages were processed. These data are visualized in Figures A.2 and A.3. The shape of the heat map corresponds to the rectangular region of the simulated virtual environment. Each data point represents the location of a peer at the time a message was processed. As the number of data points within an area accumulates, it is further darkened. Lighter regions represent areas of relatively few login forwarding messages, while darker regions represent areas with higher frequencies of messages. A visual inspection of the message distribution and density in the heat maps
Table A.4. Peer Messages - Fixed AOI.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Total</th>
<th>Average</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case</td>
<td>2,320</td>
<td>2.32</td>
<td>2</td>
<td>3.04</td>
<td>16</td>
</tr>
<tr>
<td>Pure Greedy</td>
<td>9,237</td>
<td>9.23</td>
<td>4</td>
<td>13.85</td>
<td>108</td>
</tr>
<tr>
<td>FIFO</td>
<td>13,144</td>
<td>13.13</td>
<td>8</td>
<td>14.17</td>
<td>79</td>
</tr>
<tr>
<td>Random</td>
<td>12,958</td>
<td>12.95</td>
<td>8</td>
<td>14.97</td>
<td>90</td>
</tr>
<tr>
<td>WS Random</td>
<td>6,815</td>
<td>6.81</td>
<td>5</td>
<td>7.19</td>
<td>45</td>
</tr>
<tr>
<td>WS Recent</td>
<td>5,506</td>
<td>5.5</td>
<td>4</td>
<td>5.15</td>
<td>32</td>
</tr>
<tr>
<td>WS Proportional</td>
<td>5,244</td>
<td>5.24</td>
<td>4</td>
<td>5.34</td>
<td>41</td>
</tr>
<tr>
<td>Grid Recent</td>
<td>6,055</td>
<td>6.05</td>
<td>4</td>
<td>6.8</td>
<td>48</td>
</tr>
</tbody>
</table>

Table A.5. Peer Messages - Dynamic AOI.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Total</th>
<th>Average</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Case</td>
<td>2,274</td>
<td>2.27</td>
<td>2</td>
<td>2.97</td>
<td>18</td>
</tr>
<tr>
<td>Pure Greedy</td>
<td>12,268</td>
<td>12.26</td>
<td>6</td>
<td>16.58</td>
<td>140</td>
</tr>
<tr>
<td>FIFO</td>
<td>17,174</td>
<td>17.16</td>
<td>11</td>
<td>16.57</td>
<td>92</td>
</tr>
<tr>
<td>Random</td>
<td>16,765</td>
<td>16.75</td>
<td>12</td>
<td>16.5</td>
<td>110</td>
</tr>
<tr>
<td>WS Random</td>
<td>8,271</td>
<td>8.26</td>
<td>6</td>
<td>7.65</td>
<td>47</td>
</tr>
<tr>
<td>WS Recent</td>
<td>6,368</td>
<td>6.36</td>
<td>5</td>
<td>5.52</td>
<td>36</td>
</tr>
<tr>
<td>WS Proportional</td>
<td>6,259</td>
<td>6.25</td>
<td>5</td>
<td>5.8</td>
<td>37</td>
</tr>
<tr>
<td>Grid Recent</td>
<td>7,325</td>
<td>7.32</td>
<td>6</td>
<td>7.46</td>
<td>65</td>
</tr>
</tbody>
</table>

Table A.6. Spatial Messages.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Fixed AOI</th>
<th>Dynamic AOI</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Std. Dev.</td>
<td>Max</td>
<td>Std. Dev.</td>
<td>Max</td>
</tr>
<tr>
<td>Base Case</td>
<td>0.69</td>
<td>8</td>
<td>0.68</td>
<td>6</td>
</tr>
<tr>
<td>Pure Greedy</td>
<td>10.13</td>
<td>24</td>
<td>2.13</td>
<td>20</td>
</tr>
<tr>
<td>FIFO</td>
<td>2.02</td>
<td>94</td>
<td>2.21</td>
<td>66</td>
</tr>
<tr>
<td>Random</td>
<td>1.82</td>
<td>61</td>
<td>2.18</td>
<td>80</td>
</tr>
<tr>
<td>WS Random</td>
<td>1.15</td>
<td>8</td>
<td>1.23</td>
<td>17</td>
</tr>
<tr>
<td>WS Recent</td>
<td>1.00</td>
<td>13</td>
<td>1.13</td>
<td>16</td>
</tr>
<tr>
<td>WS Proportional</td>
<td>0.96</td>
<td>8</td>
<td>1.09</td>
<td>21</td>
</tr>
<tr>
<td>Grid Recent</td>
<td>1.12</td>
<td>27</td>
<td>1.23</td>
<td>28</td>
</tr>
</tbody>
</table>
Figure A.2. Fixed AOI messages. (a) Best case (b) Pure greedy (c) FIFO (d) Random (e) Working set random (f) Working set recent (g) Working set proportional (h) Grid recent

is confirmed by the data presented in the tables.

Results Discussion

Confidence in the validity of the metric is provided by the results of the Best Case technique in comparison to all others. In every performance measure, the Best Case is always better. The results of the performance measures, along with the heat maps, show differentiation among the techniques, illustrating the ability of the metric to differentiate performance among schemes.

Before running the simulations, we expected both FIFO and Random to be similar in performance, along with being the worst case scenarios; the metric validated this expectation. Peer starting positions are randomly selected throughout the VE; therefore, FIFO is essentially a random selection technique. The performance metric correctly captured this result. Among the working set techniques, we expected the random replacement to
Figure A.3. Dynamic AOI messages. (a) Best case (b) Pure greedy (c) FIFO, (d) Random (e) Working set random (f) Working set recent (g) Working set proportional (h) Grid recent

be the least effective of the three; this was shown to be true. However, we expected proportional replacement to perform much better than recent replacement; the data do not show a statistically significant difference. An important differentiation feature of the proportional replacement technique is its computational complexity. As the number of active peers grows, the computational complexity grows linearly. Given that the fixed size recent replacement technique performs nearly as well with constant computational complexity, the choice between the two becomes obvious. The metric does not offer this insight; the expertise and knowledge of the developer is still required.

Before the simulations were performed, there was some debate regarding the performance of the grid replacement technique relative to the working set techniques. The metric shows its performance is better than the working set random replacement, but worse than the recent and proportional replacement techniques. Note the high spatial variance under
fixed AOI for the pure greedy technique and the circular pattern seen in Figure A.2b. The concentric circle radii approximate multiples of the AOI range from the server. The first circle lies along the outer boundary of the server’s AOI, the second circle is two times that distance, and so on. This clearly illustrates the algorithm choosing the known neighbor closest to the destination of the joining peer, exhibiting a spatial bias the performance measure numbers don’t readily demonstrate. In comparing Figures A.2b and A.3b, the behavior difference between the fixed and dynamic AOI is seen. For the dynamic AOI simulations, the number of neighbors was kept relatively low (10), which results primarily in choosing enclosing neighbors for handing off the login forwarding to the next peer. As all peers are constantly in motion, no particular fixed distance from one peer to its furthest neighbor exists, unlike fixed AOI where it is likely for one to have AOI neighbors near the fixed AOI distance.

A.7 Closing Remarks

This metric should be applied to individual message sub-systems and to all message sub-systems combined. The combined effects of messaging systems describes performance in general, but it is important to decompose the results by individual messaging schemes. Login forwarding, movement updates, broadcasts, and other systems must be individually characterized in order to correctly identify which systems are contributing to various performance measures.

The primary contribution of this paper is a performance metric that represents an important step forward in the characterization of P2P-based virtual environment messaging schemes. The performance measures offer a developer deeper insight into the side effects a scheme has locally among the peers, along with global effects such as spatial bias. Expertise on the part of the developer is still required to interpret the results, along with knowing what additional measures may be required.

The secondary contribution of this paper is the characterization, through the application of this performance metric, of several candidate login forwarding techniques. The results suggest the working set approach is the most promising direction in which to con-
tinue additional research. We anticipate further application of this metric as development continues on our hybrid P2P framework.
Appendix B

Detailed Parameters

B.1 Introduction

This appendix provides a supplemental listing of the Audrey implementation parameters not detailed in Chapter 5, along with a brief description of each. Most parameters are specified through an application configuration file that allows the parameters to be easily changed and evaluated through different runtime scenarios. A very small minority of these parameters are encoded into the application itself.

B.2 Neighborhood

The parameters detailed in this section control the rate at which a peer actively updates its knowledge of neighboring peers. The effect of these parameters is to increase consistency with a potential cost of higher bandwidth utilization. These also have an effect on the rate at which a peer discovers new neighbors, or supports recovery in the event of a neighboring peer halt failure.

**Neighborhood Update - MinNeighbors** This describes the minimum number of neighbors a peer must keep when performing a neighborhood update. For example, the result from the Voronoi diagram might indicate six enclosing neighbors. If this parameter is eight, then the next two closest neighbors are kept beyond the six indicated by the Voronoi diagram.

**Neighborhood Update - Frequency** This parameter describes how often (in milliseconds) a peer recomputes who it considers as neighbors, based upon network updates received since the last update computation.
**Neighborhood Update - Discovery**  
This parameter controls the rate (in milliseconds) at which a peer randomly selects one of its neighbors and requests all of its neighbors. The purpose of this technique is to help the peers auto-correct for possible oversights in neighbor notification.

**B.3 Timeout and Retries**

The network protocols described in Chapter 4 involve states that order the sending, receipt, and response of messages. Each of these states are parameterized by a pair of values, a *timeout* and a *retry*. Timeout defines the length of time (in milliseconds) a protocol waits before performing a retry on the state. Fundamentally, a retry of a protocol state results in a message being resent. Retry defines the number of times the protocol retries a state. Once the number of retries decrements to 0, the protocol is considered to have failed and the code initiating the protocol is notified of the failure.

The purpose of these settings is to mitigate the noisy environment of network communications, especially over the Internet. Generally speaking, a shorter timeout may result in more unnecessary retries attempted. A longer timeout may result in a less responsive system in the case of a higher rate of message failure. There is no one set of correct settings for these parameters. The design goals for a virtual environment help drive these settings, and they can only be finalized following rigorous real-world testing.

The following is a complete listing of the protocol states that are a part of the implementation. Each parameter has two values, one for the timeout and another for the retry. For the scenario executions performed as part of the research presented in this dissertation, all of these parameters were set to a timeout of 500 milliseconds and ten retries.

- **CreateAccountRQ**
- **CreateAccountRQWait**
- **CreateAccountRQRESP**
- **CreateAccountDetailsWait**
• LoginRQ
• LoginSessionCertificateRESP
• LoginConfigWait
• LoginConfigRESP
• LogoutInit
• StartPositionRQ
• StartPositionRESPWait
• ForwardingRQ
• ForwardingPeerInfoRESP
• NeighborhoodRQ
• NeighborhoodCertificateRQ
• NeighborhoodNeighborsWait
• RQSessionCertificateRQ
• RQSessionCertificateRESPWait
• PeerIntroduceInit
• AliveRQ
• DoneForwardingInit
• NeighborNotifyInit
• NeighborNotifyContactInfoWait
• NeighborNotifyContactInfoRESP
B.4 Misc.

The parameters in this section do not fall under any other major categorization.

**Session Expiration** This is the length of time (in minutes) for which a session certificate is valid. When peers exchange session certificates, the current time is compared against the certificate expiration time as part of the validation procedure.

**Working Set Size** This is the number of peers maintained in each peer’s forwarding working set. Each peer maintains an aliveness request with the peers in the working set. Therefore, the size of the working set has a direct effect on bandwidth.

**Aliveness - Frequency** This parameter controls how soon (in milliseconds) to start a new aliveness request with a neighboring peer, following completion of the last aliveness request.

**Aliveness - Lost** This parameter is the length of time (in milliseconds) that must expire before a neighboring peer is considered to have failed an aliveness test.
Appendix C

Comprehensive Results
Figure C.1. Bandwidth - Move = 0.00, Message = 0.00, Halt = 0.00000.

Figure C.2. Bandwidth - Move = 0.10, Message = 0.00, Halt = 0.00000.
Figure C.3. Bandwidth - Move = 0.20, Message = 0.00, Halt = 0.00000.

Figure C.4. Bandwidth - Move = 0.30, Message = 0.00, Halt = 0.00000.
Figure C.5. Bandwidth - Move = 0.40, Message = 0.00, Halt = 0.00000.

Figure C.6. Bandwidth - Move = 0.50, Message = 0.00, Halt = 0.00000.
Figure C.7. Bandwidth - Move = 0.00, Message = 0.01, Halt = 0.00000.

Figure C.8. Bandwidth - Move = 0.10, Message = 0.01, Halt = 0.00000.
Figure C.9. Bandwidth - Move = 0.20, Message = 0.01, Halt = 0.00000.

Figure C.10. Bandwidth - Move = 0.30, Message = 0.01, Halt = 0.00000.
Figure C.11. Bandwidth - Move = 0.40, Message = 0.01, Halt = 0.00000.

Figure C.12. Bandwidth - Move = 0.50, Message = 0.01, Halt = 0.00000.
Figure C.13. Bandwidth - Move = 0.00, Message = 0.05, Halt = 0.00000.

Figure C.14. Bandwidth - Move = 0.10, Message = 0.05, Halt = 0.00000.
Figure C.15. Bandwidth - Move = 0.20, Message = 0.05, Halt = 0.00000.

Figure C.16. Bandwidth - Move = 0.30, Message = 0.05, Halt = 0.00000.
Figure C.17. Bandwidth - Move = 0.40, Message = 0.05, Halt = 0.00000.

Figure C.18. Bandwidth - Move = 0.50, Message = 0.05, Halt = 0.00000.
Figure C.19. Bandwidth - Move = 0.00, Message = 0.25, Halt = 0.00000.

Figure C.20. Bandwidth - Move = 0.10, Message = 0.25, Halt = 0.00000.
Figure C.21. Bandwidth - Move = 0.20, Message = 0.25, Halt = 0.00000.

Figure C.22. Bandwidth - Move = 0.30, Message = 0.25, Halt = 0.00000.
Figure C.23. Bandwidth - Move = 0.40, Message = 0.25, Halt = 0.00000.

Figure C.24. Bandwidth - Move = 0.50, Message = 0.25, Halt = 0.00000.
Figure C.25. Bandwidth - Move = 0.00, Message = 0.00, Halt = 0.000417.

Figure C.26. Bandwidth - Move = 0.10, Message = 0.00, Halt = 0.000417.
Figure C.27. Bandwidth - Move = 0.20, Message = 0.00, Halt = 0.000417.

Figure C.28. Bandwidth - Move = 0.30, Message = 0.00, Halt = 0.000417.
Figure C.29. Bandwidth - Move = 0.40, Message = 0.00, Halt = 0.000417.

Figure C.30. Bandwidth - Move = 0.50, Message = 0.00, Halt = 0.000417.
Figure C.31. Bandwidth - Move = 0.00, Message = 0.01, Halt = 0.000417.

Figure C.32. Bandwidth - Move = 0.10, Message = 0.01, Halt = 0.000417.
Figure C.33. Bandwidth - Move = 0.20, Message = 0.01, Halt = 0.000417.

Figure C.34. Bandwidth - Move = 0.30, Message = 0.01, Halt = 0.000417.
Figure C.35. Bandwidth - Move = 0.40, Message = 0.01, Halt = 0.000417.

Figure C.36. Bandwidth - Move = 0.50, Message = 0.01, Halt = 0.000417.
Figure C.37. Bandwidth - Move = 0.00, Message = 0.05, Halt = 0.000417.

Figure C.38. Bandwidth - Move = 0.10, Message = 0.05, Halt = 0.000417.
Figure C.39. Bandwidth - Move = 0.20, Message = 0.05, Halt = 0.000417.

Figure C.40. Bandwidth - Move = 0.30, Message = 0.05, Halt = 0.000417.
Figure C.41. Bandwidth - Move = 0.40, Message = 0.05, Halt = 0.000417.

Figure C.42. Bandwidth - Move = 0.50, Message = 0.05, Halt = 0.000417.
Figure C.43. Bandwidth - Move = 0.00, Message = 0.25, Halt = 0.000417.

Figure C.44. Bandwidth - Move = 0.10, Message = 0.25, Halt = 0.000417.
Figure C.45. Bandwidth - Move = 0.20, Message = 0.25, Halt = 0.000417.

Figure C.46. Bandwidth - Move = 0.30, Message = 0.25, Halt = 0.000417.
<table>
<thead>
<tr>
<th>Movement (p=0.40) Message Failure (p=0.25) Halt Failure (p=0.000417)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Figure C.47. Bandwidth - Move = 0.40, Message = 0.25, Halt = 0.000417.

<table>
<thead>
<tr>
<th>Movement (p=0.50) Message Failure (p=0.25) Halt Failure (p=0.000417)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Figure C.48. Bandwidth - Move = 0.50, Message = 0.25, Halt = 0.000417.
Figure C.49. Bandwidth - Move = 0.00, Message = 0.00, Halt = 0.00100.

Figure C.50. Bandwidth - Move = 0.10, Message = 0.00, Halt = 0.00100.
Figure C.51. Bandwidth - Move = 0.20, Message = 0.00, Halt = 0.00100.

Figure C.52. Bandwidth - Move = 0.30, Message = 0.00, Halt = 0.00100.
Figure C.53. Bandwidth - Move = 0.40, Message = 0.00, Halt = 0.00100.

Figure C.54. Bandwidth - Move = 0.50, Message = 0.00, Halt = 0.00100.
Figure C.55. Bandwidth - Move = 0.00, Message = 0.01, Halt = 0.00100.

Figure C.56. Bandwidth - Move = 0.10, Message = 0.01, Halt = 0.00100.
Figure C.57. Bandwidth - Move = 0.20, Message = 0.01, Halt = 0.00100.

Figure C.58. Bandwidth - Move = 0.30, Message = 0.01, Halt = 0.00100.
Figure C.59. Bandwidth - Move = 0.40, Message = 0.01, Halt = 0.00100.

Figure C.60. Bandwidth - Move = 0.50, Message = 0.01, Halt = 0.00100.
Figure C.61. Bandwidth - Move = 0.00, Message = 0.05, Halt = 0.00100.

Figure C.62. Bandwidth - Move = 0.10, Message = 0.05, Halt = 0.00100.
Figure C.63. Bandwidth - Move = 0.20, Message = 0.05, Halt = 0.00100.

Figure C.64. Bandwidth - Move = 0.30, Message = 0.05, Halt = 0.00100.
Figure C.65. Bandwidth - Move = 0.40, Message = 0.05, Halt = 0.00100.

Figure C.66. Bandwidth - Move = 0.50, Message = 0.05, Halt = 0.00100.
Figure C.67. Bandwidth - Move = 0.00, Message = 0.25, Halt = 0.00100.

Figure C.68. Bandwidth - Move = 0.10, Message = 0.25, Halt = 0.00100.
Figure C.69. Bandwidth - Move = 0.20, Message = 0.25, Halt = 0.00100.

Figure C.70. Bandwidth - Move = 0.30, Message = 0.25, Halt = 0.00100.
Figure C.71. Bandwidth - Move = 0.40, Message = 0.25, Halt = 0.00100.

Figure C.72. Bandwidth - Move = 0.50, Message = 0.25, Halt = 0.00100.
Figure C.73. Consistency - Move = 0.00, Message = 0.00, Halt = 0.00000.

Figure C.74. Consistency - Move = 0.10, Message = 0.00, Halt = 0.00000.
Figure C.75. Consistency - Move = 0.20, Message = 0.00, Halt = 0.00000.

Figure C.76. Consistency - Move = 0.30, Message = 0.00, Halt = 0.00000.
Figure C.77. Consistency - Move = 0.40, Message = 0.00, Halt = 0.00000.

Figure C.78. Consistency - Move = 0.50, Message = 0.00, Halt = 0.00000.
Figure C.79. Consistency - Move = 0.00, Message = 0.01, Halt = 0.00000.

Figure C.80. Consistency - Move = 0.10, Message = 0.01, Halt = 0.00000.
Figure C.81. Consistency - Move = 0.20, Message = 0.01, Halt = 0.00000.

Figure C.82. Consistency - Move = 0.30, Message = 0.01, Halt = 0.00000.
Figure C.83. Consistency - Move = 0.40, Message = 0.01, Halt = 0.00000.

Figure C.84. Consistency - Move = 0.50, Message = 0.01, Halt = 0.00000.
Figure C.85. Consistency - Move = 0.00, Message = 0.05, Halt = 0.00000.

Figure C.86. Consistency - Move = 0.10, Message = 0.05, Halt = 0.00000.
Figure C.87. Consistency - Move = 0.20, Message = 0.05, Halt = 0.00000.

Figure C.88. Consistency - Move = 0.30, Message = 0.05, Halt = 0.00000.
Figure C.89. Consistency - Move = 0.40, Message = 0.05, Halt = 0.00000.

Figure C.90. Consistency - Move = 0.50, Message = 0.05, Halt = 0.00000.
Figure C.91. Consistency - Move = 0.00, Message = 0.25, Halt = 0.00000.

Figure C.92. Consistency - Move = 0.10, Message = 0.25, Halt = 0.00000.
Figure C.93. Consistency - Move = 0.20, Message = 0.25, Halt = 0.00000.

Figure C.94. Consistency - Move = 0.30, Message = 0.25, Halt = 0.00000.
Figure C.95. Consistency - Move = 0.40, Message = 0.25, Halt = 0.00000.

Figure C.96. Consistency - Move = 0.50, Message = 0.25, Halt = 0.00000.
Figure C.97. Consistency - Move = 0.00, Message = 0.00, Halt = 0.000417.

Figure C.98. Consistency - Move = 0.10, Message = 0.00, Halt = 0.000417.
Figure C.99. Consistency - Move = 0.20, Message = 0.00, Halt = 0.000417.

Figure C.100. Consistency - Move = 0.30, Message = 0.00, Halt = 0.000417.
Figure C.101. Consistency - Move = 0.40, Message = 0.00, Halt = 0.000417.

Figure C.102. Consistency - Move = 0.50, Message = 0.00, Halt = 0.000417.
Figure C.103. Consistency - Move = 0.00, Message = 0.01, Halt = 0.000417.

Figure C.104. Consistency - Move = 0.10, Message = 0.01, Halt = 0.000417.
Figure C.105. Consistency - Move = 0.20, Message = 0.01, Halt = 0.000417.

Figure C.106. Consistency - Move = 0.30, Message = 0.01, Halt = 0.000417.
Figure C.107. Consistency - Move = 0.40, Message = 0.01, Halt = 0.000417.

Figure C.108. Consistency - Move = 0.50, Message = 0.01, Halt = 0.000417.
Figure C.109. Consistency - Move = 0.00, Message = 0.05, Halt = 0.000417.

Figure C.110. Consistency - Move = 0.10, Message = 0.05, Halt = 0.000417.
Figure C.111. Consistency - Move = 0.20, Message = 0.05, Halt = 0.000417.

Figure C.112. Consistency - Move = 0.30, Message = 0.05, Halt = 0.000417.
Figure C.113. Consistency - Move = 0.40, Message = 0.05, Halt = 0.000417.

Figure C.114. Consistency - Move = 0.50, Message = 0.05, Halt = 0.000417.
Figure C.115. Consistency - Move = 0.00, Message = 0.25, Halt = 0.000417.

Figure C.116. Consistency - Move = 0.10, Message = 0.25, Halt = 0.000417.
Figure C.117. Consistency - Move = 0.20, Message = 0.25, Halt = 0.000417.

Figure C.118. Consistency - Move = 0.30, Message = 0.25, Halt = 0.000417.
Figure C.119. Consistency - Move = 0.40, Message = 0.25, Halt = 0.000417.

Figure C.120. Consistency - Move = 0.50, Message = 0.25, Halt = 0.000417.
Figure C.121. Consistency - Move = 0.00, Message = 0.00, Halt = 0.00100.

Figure C.122. Consistency - Move = 0.10, Message = 0.00, Halt = 0.00100.
Figure C.123. Consistency - Move = 0.20, Message = 0.00, Halt = 0.00100.

Figure C.124. Consistency - Move = 0.30, Message = 0.00, Halt = 0.00100.
Figure C.125. Consistency - Move = 0.40, Message = 0.00, Halt = 0.00100.

Figure C.126. Consistency - Move = 0.50, Message = 0.00, Halt = 0.00100.
Figure C.127. Consistency - Move = 0.00, Message = 0.01, Halt = 0.00100.

Figure C.128. Consistency - Move = 0.10, Message = 0.01, Halt = 0.00100.
Figure C.129. Consistency - Move = 0.20, Message = 0.01, Halt = 0.00100.

Figure C.130. Consistency - Move = 0.30, Message = 0.01, Halt = 0.00100.
Figure C.131. Consistency - Move = 0.40, Message = 0.01, Halt = 0.00100.

Figure C.132. Consistency - Move = 0.50, Message = 0.01, Halt = 0.00100.
Figure C.133. Consistency - Move = 0.00, Message = 0.05, Halt = 0.00100.

Figure C.134. Consistency - Move = 0.10, Message = 0.05, Halt = 0.00100.
Figure C.135. Consistency - Move = 0.20, Message = 0.05, Halt = 0.00100.

Figure C.136. Consistency - Move = 0.30, Message = 0.05, Halt = 0.00100.
Figure C.137. Consistency - Move = 0.40, Message = 0.05, Halt = 0.00100.

Figure C.138. Consistency - Move = 0.50, Message = 0.05, Halt = 0.00100.
Figure C.139. Consistency - Move = 0.00, Message = 0.25, Halt = 0.00100.

Figure C.140. Consistency - Move = 0.10, Message = 0.25, Halt = 0.00100.
Figure C.141. Consistency - Move = 0.20, Message = 0.25, Halt = 0.00100.

Figure C.142. Consistency - Move = 0.30, Message = 0.25, Halt = 0.00100.
Figure C.143. Consistency - Move = 0.40, Message = 0.25, Halt = 0.00100.

Figure C.144. Consistency - Move = 0.50, Message = 0.25, Halt = 0.00100.
Appendix D

Reprint Permission
Dr. Hamid Arabnia  
The University of Georgia  
Department of Computer Science  
415 Boyd Graduate Studies Research Center  
Athens, Georgia 30602-7404

Dr. Arabnia,

I am preparing my Dissertation in the Department of Computer Science at Utah State University. I am anticipating completion of my degree in May of 2012.

An article, A Performance Metric for Message Forwarding Schemes of Massively Multiplayer Peer-to-Peer Based Networked Virtual Environments, of which I am the first author, and which appears in the proceedings of the PDPTA 2011 conference, reports an essential part of my Dissertation research. I would like to reprint it as an Appendix in my Dissertation (reprinting the paper may necessitate some revision.) Please note that USU sends Dissertations to Bell & Howell Dissertation Services to be made available for reproduction.

Please indicate your approval of this request by signing in the space provided, and attach any other form necessary to confirm permission. If you charge a reprint fee for use of an article by the author, please indicate that as well.

Best Regards,

[Signature]

Dean Mathias

I hereby give permission to James Dean Mathias to reprint the requested article in his Dissertation.

Signed: [Signature]  
Date: 12/12/2011  
Fee: No Fee.
CURRICULUM VITAE
James Dean Mathias

EDUCATION

Ph.D., Computer Science. Utah State University, Logan, UT. 2012.


RESEARCH INTERESTS

Distributed and parallel systems, massively multiplayer frameworks, computer graphics, evolutionary algorithms, and concurrency

CONFERENCE PUBLICATIONS


GOVERNMENT PUBLICATIONS


SOFTWARE MANUALS


INDUSTRY EXPERIENCE


Lecturer. Utah State University. 2001 to Present.


*Ecological Database (EcoDB)*. An enterprise database product for storage of ecological data collections. Desktop and browser client UIs developed with Visual Studio.NET using C# and ASP.NET. Middle tier components using COM, developed with Visual Studio.NET, coded in C++, using ATL. Backend database using MS SQL Server or MySQL.

*Recreational, Invertebrate, and Fisheries Database*. Three-tier software systems developed with Borland C++ Builder for the UI, Visual C++ for the COM middle-tier and MS SQL Server for the back end database. Designed and developed for Idaho Power.

*Reservoir Release Forecast Model (RRFM)*. Rule based decision tree and Monte Carlo uncertainty simulation package for evaluating the Risk of emergency flood control operations for the Folsom dam in Sacramento, California.

*Aquatics Database* Rescued a failed project and provided further phases of development for Idaho Power. UI development in Delphi and back end database in SQL Server.
Phabsim. A Windows based fisheries physical habitat model based upon a legacy DOS system. MS Access used to store data, project written in Visual C++ and uses ADO for DB interaction.

Windows NT 4.0 Simulator A comprehensive Windows NT 4.0 simulator used by Microsoft for their certification exams on the Sylvan testing platform. All code developed with Borland Delphi.

Driven. A program used to help design pile foundations for bridge structures (Federal Highway Administration). Developed for Windows 3.1 using Borland C++.

TEACHING EXPERIENCE

CS 1400 - CS 1 (C++). Utah State University. 2002 to 2010.


CS 5410 - Technical Game Development. Utah State University. 2008 to Present.

CS 5890 - Software Optimization. Utah State University. 2009.

SERVICE


Reviewer, Frontiers In Education. 2009.

Invited presentation on “Technical Job Interviews” presented to the Utah State University ACM Womens club. Multiple occasions.

Invited presentation on “Technical Job Interviews” presented to the Utah State University CS 3100 undergraduate seminar. Multiple occasions.

Invited presentation on “Genetic Programming” presented to the Computer Science faculty of Utah State University. October, 2005.