Network aspects of playability and fairness: Improving user experience in online games

Jeremy Pascal Brun
University of Wollongong
UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
Network Aspects of Playability and Fairness: Improving User Experience in Online Games

A thesis submitted in fulfilment of the requirements for the award of the degree

Doctor of Philosophy

from

The University of Wollongong

by

Jeremy Pascal Brun

School of Electrical, Computer and Telecommunications Engineering

2012
Abstract

Born only a few decades ago, the video game industry has now surpassed the revenue of the movie business and shows no sign of decline in its expansion rate. In addition to the large number of exclusively online titles operating today, most regular recent game also provide a networked multiplayer mode. Despite their success, the online experience of online games is extremely sensitive to network conditions.

Network games are realtime and highly interactive. For these reasons they are more affected by the imperfections of the telecommunication network than most other Internet applications. In particular, geographical distances introduce unavoidable delays which degrades the experience of players. This objective of this thesis is to models the impact of network imperfections on game playability and propose novel solutions to improve the quality of experience of participants in online games.

There is no consistent analysis across different game research groups on how network delays affect game users. In order to understand the impacts of network delays on game quality, we introduce a generic framework which can analyse the propagation of network disturbances in a game as a three steps process. First, measurable inconsistencies are derived from the game’s network topology and synchronisation scheme. Next, we determine the violations of the ideal laws of the virtual environment generated by these inconsistencies. Finally, the perceptual impact of these violations on players is estimated using fuzzy logic utility functions.
Abstract

Game quality metrics proposed in the literature either suit a particular setup or do not capture every aspects of the players’ experience. Using our framework, we introduce a measurable and objective definition of network playability as the collection of all inconsistencies endured by a player. As fairness is a function of the relative playability amongst game users, we define the network unfairness of a game as the variation coefficient of the participants’ playability and validate this definition in an experimental setup.

The thesis also investigates the potential to trade-off different aspects of the playability of participants through the alteration of (the) game synchronisation. We demonstrate that absolutely conservative and fully optimistic synchronisation schemes are the extremes of a continuum of possible strategies. Using a specialised game simulator, we search and find the optimum synchronisation parameters within this continuum of trade-offs. Instead of using the same synchronisation for all actions, a second improvement consists of tuning the synchronisation parameters used for different actions independently and according to the specific requirements of each action type.

After studying and proposing enhancements to the synchronisation scheme of online games, this thesis formulates an integer programming problem aiming at optimising the selection of game servers in a cloud of potential sites. We introduce the notion of critical inconsistency and demonstrate that optimising this criteria tends to improve both playability and fairness. Finally, the resolution of small instances of this problem lead to the development of a recursive heuristic capable of converging towards a near-optimum set of servers providing best network conditions for the players of an online game. Simulations shows the gap between our heuristic solutions and a calculable lower bound average 5.19% (2.95 StdDev).
Statement of Originality

This is to certify that the work described in this thesis is entirely my own, except where due reference is made in the text.

No work in this thesis has been submitted for a degree to any other university or institution.

Signed

Jeremy Pascal Brun
5 August, 2012
Acknowledgments

My PhD adventure did not match my initial expectations. The road has been much longer, harder and rewarding than I anticipated. I would not have been able to write and complete this thesis without the help of numerous people who have supported me in many different ways over the past 10 (no, this is not a typo) years.

First, I would like to thank both my supervisors Professor Farzad Safaei and Dr. Paul Boustead which beside their obvious technical contributions to my work have also been most patient and never gave up supporting me in my research and more. Farzad, many of the discussions we had in your office have influenced me not only on a technical level but also philosophically, making me a different and I believe a better person. Paul, your limitless enthusiasm is only one of the many reasons I (still) love working with you.

I also want to thank my fellow PhD students from the University of Wollongong: Daniel Franklin, Justin Lipman, Mehran Abdolassan, Stephen Davis, Vinh Nguyen, Eva Cheng, Daniel Smith, Xinbo Jiang and Irwan. Thank you all for the chats, the laughs, the coffees, the squash games and the late Quake nights.

Thank you to my parents Michel and Antoinette, my big little brother Adam and my funny little sister Charlotte who also supported me despite not understanding what was taking me so long.

Mitra, thank you for everything and for the food, but you always make too
much! Daniel, thanks for answering my impossible Perl and Linux questions. Amina, thanks for the awfully direct criticisms and all the good times we have.

Special thanks to: Lisa Barlow for the very special letter which helped me bouncing from the bottom when everything was falling apart. Isabelle Choque for the weekly remote continuous support from France. Sheik and Sobhana Huque for providing me with the quiet place I needed to finish the writing.

I am grateful to the Smart Internet CRC for the money and the University of Wollongong for the PhD opportunity and the occasional free food (especially when the money ran out).

Thank you to the reviewers Dr. Stefano Ferretti and Dr.-Ing. Carsten Griwodz which comments have helped me improving this work.

Negative acknowledgements are also deserved to particular people I will not mention by name. Fair credit to the Unibar for providing some of the worst food quality to price ratio I know. Not forgetting the regular networking screw-up of ITS which have cost me quite a bit of time and energy because, yes, a working computer network is useful for research in telecommunications. Also, a special mention for the particular rock on Bondi cliffs which refused to support my weight with dramatic gravity induced and vertebra crushing consequences.

Last but not least, thank you to my wife to be, Tajeen Huque, for the enormous amount of moral encouragement and support which allowed me to get through the writing of this thesis.
Contents

1 Introduction ........................................... 1
   1.1 Background ....................................... 2
   1.2 Thesis objective, overview and contributions ...... 4
       1.2.1 Overview .................................... 5
       1.2.2 Contributions ................................. 7
       1.2.3 Peer reviewed publications .................... 9

2 Literature and industry state of the art review .... 11
   2.1 Introduction ....................................... 12
   2.2 Origin of networked game research ................ 13
       2.2.1 Military Simulations ............................ 13
       2.2.2 Networked Virtual Environments ............... 14
   2.3 The Game Industry .................................. 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1 Network Architecture in the Game Industry</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2 Distributed architectures in the game industry</td>
<td>19</td>
</tr>
<tr>
<td>2.3.3 Hiding and compensating latency</td>
<td>20</td>
</tr>
<tr>
<td>2.3.4 Directions of the game industry</td>
<td>21</td>
</tr>
<tr>
<td>2.3.5 Conclusion on the state of the industry</td>
<td>23</td>
</tr>
<tr>
<td>2.4 Impact of network imperfections</td>
<td>24</td>
</tr>
<tr>
<td>2.4.1 Classifying network games</td>
<td>24</td>
</tr>
<tr>
<td>2.4.2 First Person Shooters</td>
<td>25</td>
</tr>
<tr>
<td>2.4.3 Role Playing Games</td>
<td>26</td>
</tr>
<tr>
<td>2.4.4 Real Time Strategy</td>
<td>27</td>
</tr>
<tr>
<td>2.4.5 Sport simulations</td>
<td>28</td>
</tr>
<tr>
<td>2.4.6 Concluding remarks on empirical studies</td>
<td>28</td>
</tr>
<tr>
<td>2.5 Inconsistency models</td>
<td>29</td>
</tr>
<tr>
<td>2.6 Communication Architectures and Synchronisation</td>
<td>31</td>
</tr>
<tr>
<td>2.6.1 An introductory example to the synchronisation problem</td>
<td>33</td>
</tr>
<tr>
<td>2.6.2 Distributed computing and simulations</td>
<td>37</td>
</tr>
<tr>
<td>2.6.3 Consistency and Synchronisation in distributed games</td>
<td>38</td>
</tr>
<tr>
<td>2.6.4 Conclusion on synchronisation</td>
<td>42</td>
</tr>
</tbody>
</table>
3 Terminology, Assumptions and Introductory concepts

3.1 Introduction ................................................. 49
3.2 Terminology ................................................... 49
3.3 Assumptions .................................................... 52
  3.3.1 Simulations requirements ................................ 52
  3.3.2 Network and processing imperfections ............... 53
3.4 Game State: the logical heart of networked games ...... 55
  3.4.1 Decision points, terminals and game state .......... 55
  3.4.2 Decision: altering the game state .................... 57
  3.4.3 Game state vector and trajectory ..................... 60
  3.4.4 Game state: summary ................................... 61
3.5 Conclusion .................................................... 61
## 4 Networking aspects of user experience

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>64</td>
</tr>
<tr>
<td>4.2 Model of game disturbances</td>
<td>65</td>
</tr>
<tr>
<td>4.2.1 From network delays to user impact</td>
<td>65</td>
</tr>
<tr>
<td>4.2.2 Inconsistencies and violations</td>
<td>67</td>
</tr>
<tr>
<td>4.2.3 Summary</td>
<td>68</td>
</tr>
<tr>
<td>4.3 Inconsistencies</td>
<td>69</td>
</tr>
<tr>
<td>4.3.1 Inconsistencies in the central server architecture</td>
<td>69</td>
</tr>
<tr>
<td>4.3.2 Inconsistencies in the peer to peer model</td>
<td>72</td>
</tr>
<tr>
<td>4.3.3 Inconsistencies in the distributed server model</td>
<td>78</td>
</tr>
<tr>
<td>4.3.4 Aggregating of Update, Request ages and local lag</td>
<td>80</td>
</tr>
<tr>
<td>4.3.5 Inconsistencies: summary</td>
<td>81</td>
</tr>
<tr>
<td>4.4 Violations and user impact</td>
<td>82</td>
</tr>
<tr>
<td>4.4.1 Inconsistencies cause violations</td>
<td>82</td>
</tr>
<tr>
<td>4.4.2 Action evaluation distance</td>
<td>82</td>
</tr>
<tr>
<td>4.4.3 Response time</td>
<td>84</td>
</tr>
<tr>
<td>4.4.4 Causality related violations</td>
<td>85</td>
</tr>
<tr>
<td>4.5 Conclusion</td>
<td>87</td>
</tr>
</tbody>
</table>
5 Playability and fairness

5.1 Introduction ........................................ 89
5.2 Playability ........................................... 90
  5.2.1 Defining playability ............................ 90
  5.2.2 Network playability ............................ 90
5.3 Impact on players ................................... 92
  5.3.1 Utility functions ................................. 92
  5.3.2 Composing utilities ............................. 97
  5.3.3 Utilities: summary .............................. 99
5.4 Altering violations and inconsistencies .......... 99
  5.4.1 Trading violations .............................. 100
  5.4.2 Trading inconsistencies ....................... 103
  5.4.3 Trading inconsistencies and violations: summary ... 108
5.5 Fairness ............................................. 109
  5.5.1 Source of unfairness ........................... 109
  5.5.2 Estimation of network fairness ................. 110
  5.5.3 Experimentally measuring fairness ............. 112
  5.5.4 Fairness considerations ........................ 116
CONTENTS

6 Improving playability: distributing servers and tailoring local lag 119

6.1 Introduction ........................................ 120

6.2 NetGameSim: a network game simulator ................. 121

6.2.1 Overview ........................................... 121

6.2.2 Player and Decision-Point state machines ............ 123

6.2.3 Simulation outputs ................................. 124

6.3 Trading-off conservative and optimistic synchronisation . 125

6.3.1 Conservative vs. Optimistic ......................... 125

6.3.2 Tuning Local Lag ................................. 126

6.3.3 Simulations and results ............................. 127

6.3.4 Discussion on Local Lag tuning .................... 129

6.3.5 Conclusion ....................................... 134

6.4 Tailoring synchronisation parameters per actions ........ 136

6.4.1 Unbinding actions’ synchronisation fate ................ 136

6.4.2 Simulations and results ............................. 137

6.4.3 Discussion on tailoring Local Lag to action ............ 139

6.4.4 Conclusion on tailoring synchronisation ............... 140
7 Decision Point Topology Considerations in Online Games

7.1 Introduction .................................................. 144

7.2 Decision point placement problem formulation ....... 146

7.2.1 General Description ..................................... 146

7.2.2 Problem Formulation .................................... 147

7.2.3 Concluding remarks on the formulation ............ 150

7.3 Critical Inconsistency ...................................... 151

7.3.1 The critical path ......................................... 152

7.3.2 A lower bound to Critical Inconsistency .......... 153

7.4 Server selection in small networks ................. 156

7.4.1 Simulations setup ....................................... 156

7.4.2 Simulations results ..................................... 157

7.4.3 Discussion ................................................ 160

7.4.4 Conclusions on optimisation of small networks .. 161

7.5 Server selection in large networks ................. 162

7.5.1 Heuristic description ................................... 162

7.5.2 Performance simulations ............................. 163
8 Conclusions and future work

8.1 Overview .................................. 172

8.2 Summary of contributions and findings .................. 172

8.2.1 A framework for the analysis of network disturbances . 172

8.2.2 Improving the network infrastructure ............... 174

8.3 Future work .................................. 175

A NetGameSim tables of simulations results ............. 194

B More about inconsistencies .......................... 196

B.1 Mathematical Sequence formulation of correct synchronisation 196

C More about Utility functions ........................ 200

C.1 Mathematical consideration of game utility ............ 200

C.1.1 Definition of a utility function .................... 200

C.1.2 Composing utilities .......................... 201

C.1.3 Inconsistencies and total game utility ............ 203

D Cancelling a game’s initial conditions ................. 204
D.1 Initial game state conditions and fairness .......................... 204

E Linearisation of decision point problem .............................. 206

E.1 Introduction .............................................................. 206

E.1.1 Given Parameters ..................................................... 206

E.1.2 Decision Variables ................................................... 207

E.1.3 Dependent Variables ............................................... 208

E.1.4 Constraints ............................................................ 208

E.1.5 Objective functions and resolution ............................... 209
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Main network games architectures</td>
<td>32</td>
</tr>
<tr>
<td>2.2</td>
<td>A two players game</td>
<td>34</td>
</tr>
<tr>
<td>2.3</td>
<td>Central server time diagram</td>
<td>35</td>
</tr>
<tr>
<td>2.4</td>
<td>Distributed servers time diagram (no paradox avoidance)</td>
<td>36</td>
</tr>
<tr>
<td>3.1</td>
<td>Decision point, action and game state</td>
<td>58</td>
</tr>
<tr>
<td>3.2</td>
<td>Variables roles regarding a given action</td>
<td>59</td>
</tr>
<tr>
<td>4.1</td>
<td>From network topology to impact on players</td>
<td>66</td>
</tr>
<tr>
<td>4.2</td>
<td>Terminal inconsistencies in the central server architecture</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>Imbalanced central server game topology</td>
<td>72</td>
</tr>
<tr>
<td>4.4</td>
<td>Event age in optimistic and conservative synchronisation</td>
<td>75</td>
</tr>
<tr>
<td>4.5</td>
<td>Inconsistencies in distributed server architecture</td>
<td>79</td>
</tr>
<tr>
<td>4.6</td>
<td>Asymmetric delays to a central server</td>
<td>80</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>5.1</td>
<td>Approximation of a FPS utility as a type-1 membership function</td>
<td>94</td>
</tr>
<tr>
<td>5.2</td>
<td>Approximation of a utility function as an IT2 membership</td>
<td>95</td>
</tr>
<tr>
<td>5.3</td>
<td>Estimation of response time and co-simulation utilities</td>
<td>101</td>
</tr>
<tr>
<td>5.4</td>
<td>Total utility using classic fuzzy logic conjunction operators</td>
<td>102</td>
</tr>
<tr>
<td>5.5</td>
<td>Playability space and playable zone</td>
<td>104</td>
</tr>
<tr>
<td>5.6</td>
<td>A simple two decision points distributed architecture</td>
<td>105</td>
</tr>
<tr>
<td>5.7</td>
<td>Moving players in the Playability Space (1)</td>
<td>106</td>
</tr>
<tr>
<td>5.8</td>
<td>Moving players in the Playability Space (2)</td>
<td>107</td>
</tr>
<tr>
<td>5.9</td>
<td>Fairness measure - simulated topologies</td>
<td>114</td>
</tr>
<tr>
<td>5.10</td>
<td>Score of players vs P0's distance from the server</td>
<td>115</td>
</tr>
<tr>
<td>5.11</td>
<td>Game (un)fairness vs P0's distance from the server</td>
<td>115</td>
</tr>
<tr>
<td>6.1</td>
<td>A snapshot of a NetGameSim game session</td>
<td>124</td>
</tr>
<tr>
<td>6.2</td>
<td>Variable local in a time diagram</td>
<td>127</td>
</tr>
<tr>
<td>6.3</td>
<td>NetGameSim simulated topology</td>
<td>128</td>
</tr>
<tr>
<td>6.4</td>
<td>Tuning Local Lag: Response Time and Rollbacks</td>
<td>129</td>
</tr>
<tr>
<td>6.5</td>
<td>Tuning Local Lag in Quake3 (source: Liang)</td>
<td>130</td>
</tr>
<tr>
<td>6.6</td>
<td>Decision point inconsistency vs. Local Lag</td>
<td>133</td>
</tr>
<tr>
<td>6.7</td>
<td>Rollbacks vs. inter-DP inconsistency</td>
<td>133</td>
</tr>
</tbody>
</table>
6.8 Estimations of response time and rollback utilities ............ 135
6.9 Total utility when varying the local lag .......................... 135
6.10 Rollbacks contours versus shooting and moving local lag ... 138
6.11 Response time contours versus shooting and moving local lag 138
7.1 Critical links, critical path and critical players .................. 152
7.2 A regular hierarchical network topology .......................... 158
7.3 Optimum critical inconsistency vs number of servers .......... 159
7.4 Iterative evolution of a typical heuristic convergence .......... 164
7.5 Heuristic performance using network playability ............... 166
7.6 Average and standard deviation of inconsistencies ............. 166
7.7 Utility function of playability for Role Playing Games (RPGs) . 168
7.8 Heuristic performance using perceived playability ............. 168
List of Tables

2.1 Early NVEs and some of their key features .................. 16

A.1 Average number of Rollbacks ................................. 195

A.2 Average response time when ................................. 195
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>Action Age</td>
</tr>
<tr>
<td>AoI</td>
<td>Area of Interest</td>
</tr>
<tr>
<td>CV</td>
<td>Coefficient of Variation</td>
</tr>
<tr>
<td>CI</td>
<td>Critical Inconsistency</td>
</tr>
<tr>
<td>DES</td>
<td>Discrete Event Simulation</td>
</tr>
<tr>
<td>DiffServ</td>
<td>Differentiated Services</td>
</tr>
<tr>
<td>DIS</td>
<td>Distributed Interactive Simulation</td>
</tr>
<tr>
<td>DP</td>
<td>Decision Point</td>
</tr>
<tr>
<td>DVE</td>
<td>Distributed Virtual Environment</td>
</tr>
<tr>
<td>DR</td>
<td>Dead Reckoning</td>
</tr>
<tr>
<td>EA</td>
<td>Electronic Art</td>
</tr>
<tr>
<td>ET</td>
<td>Enemy Territory</td>
</tr>
<tr>
<td>FOU</td>
<td>Footprint of Uncertainty</td>
</tr>
<tr>
<td>FPS</td>
<td>First Person Shooter (Game)</td>
</tr>
<tr>
<td>GS</td>
<td>Game State</td>
</tr>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task-Force</td>
</tr>
<tr>
<td>IntServ</td>
<td>Integrated Services</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IRC</td>
<td>Internet Relay Chat</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LL</td>
<td>Local Lag</td>
</tr>
<tr>
<td>MMO/MMOG</td>
<td>Massively Multiplayer Online Game</td>
</tr>
<tr>
<td>MMORPG</td>
<td>Massively Multiplayer Online Role Playing Game</td>
</tr>
<tr>
<td>MPLS</td>
<td>Multi-Protocol Label Switching</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NPC</td>
<td>Non Player Character</td>
</tr>
<tr>
<td>NVE</td>
<td>Networked Virtual Environment</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer to Peer</td>
</tr>
<tr>
<td>PDES</td>
<td>Parallel Distributed Event Simulation</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RB</td>
<td>Roll Back</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>RPG</td>
<td>Role Playing Game</td>
</tr>
<tr>
<td>RT</td>
<td>Response Time</td>
</tr>
<tr>
<td>RTS</td>
<td>Real Time Strategy (Game)</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>T1 MF</td>
<td>Type-1 Membership Function</td>
</tr>
<tr>
<td>T2 MF</td>
<td>Type-2 Membership Function</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>WoW</td>
<td>World of Warcraft</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Synopsis

This first chapter aims to introduce the reader to the thesis and its subject.

Content:

- Background network games
- Thesis objective, overview and contributions
1.1 Background

Although the Internet was initially developed as a research and military the name ARPARNET in the 70’s, it only reached the general public a couple of decades later in the 90’s. At first, its popular applications, basic HTML browsing, email, newsgroups and file exchange were neither real-time nor interactive.

Before the availability of a global digital network, video games were typically running on a single console or computer. Multiplayer games were possible only with different users controlling independent input devices connected to the same machine.

Emergence of networked games

The exponential growth of processing power and telecommunication transmission capacity went along with the wider availability of electronic, computer and telecommunication hardware. These factors enabled new classes of network centric software applications to emerge. Today, the Internet carries audio and video streams and provides a variety of used-to-be desktop applications under the web2.0[108] banner, such as word processing, spreadsheet or image manipulation which can run in a web browser.

Networked interactive multimedia applications, to which online games belong, is another one of this class of new communication centric software. In general, these applications differ in fundamental ways from the original web usage which was primarily about retrieval of pre-generated data. Because online games are all about the real-time actions and interactions of multiples users, their content is highly dynamic and have to be generated and distributed on the fly. These operations are very resource hungry in terms Central Processing Unit (CPU) time and transmission capacity with requirements growing with the number of simultaneous users. The real-time requirements of online games data flows also make them very sensitive to network quality parameters like latency and packet loss.
Challenges to networked game deployment

As an entertainment application, a successful game must consistently provide acceptable levels of user enjoyment. Early network games were bound to limited number of users playing over Local Area Network (LAN). In these conditions the negative impact of the infrastructure supporting the game, such as delay or packet loss, is minimal and the user experience depends, almost exclusively, on the originality, artistic design and rendering of the game content.

However, newer online games aim to gather thousands of players located on different continents in massive virtual worlds. Scaling the number of users increases requirements both in terms of transmission capacity and data processing while scaling geographically increases the propagation delay of information. In these scenarios, and in contrast to single player and LAN based games, the network infrastructure supporting a game becomes a major factor impacting the experience of players.

Naturally, game players would not like to spend time (and money) in a game where actions are delayed beyond acceptable playability levels or where other players seem to have inexplicable and unfair advantages. Therefore the successful deployment of such systems requires solving the complex technological challenges of scaling in terms of player population and geographical span while maintaining enjoyable levels of playability and fairness.

The need for CPU-bound applications in many other domains, such as scientific simulation, has driven the active area of distributed and parallel processing also called High Performance Computing (HPC). Parallelization research not only led to hardware improvements such as multi-processor machines and multi-core processors, it also developed the higher level technologies of clusters and grid computing. Online game developers have been using these techniques to scale in terms of number of players so that the same virtual world can accommodate larger number of simultaneous avatars. In these large data centres, transmission capacity have also been provisioned to meet the requirements of these data flows.
Coping with latency

While CPU and transmission capacity limitations can technically be overcome by increasing available resources, there is no simple solution for propagation delays which are truly bounded because information can not travel faster than the speed of light.

Moreover, the Internet Protocol (IP) was designed as a best effort service with no quality assurance about the transport of data as packets may be corrupted during transmission and routers may delay or even discard them. Multiple Quality of Service (QoS) technologies have been developed by the Internet Engineering Task Force (IETF) to overcome IP limitations. These initiatives include Integrated Services (IntServ)[19], Differentiated Services (DiffServ)[15] or Multi-Protocol Label Switching (MPLS)[127]. However, these protocols would have to be adopted by all data carriers to effectively provide end to end QoS from any host to any host across the Internet.

Service distribution can also be used to cope with latency issues[130]. For example, in the area of web content distribution, the private company Akamai developed a world wide overlay network. Associated studies [125][137] show significant improvement in latency of web content retrieval using the Akamai technology.

For technical and economic reasons, game developers have not attempted to distribute game servers. Instead they came with a number of other workarounds to cope with the telecommunication network imperfections. These improvements, which are listed and analysed later in this thesis, always end up providing delayed or inconsistent perceptions of the virtual environment to players.

1.2 Thesis objective, overview and contributions

In online games, latency impacts the quality of game experience by reducing playability and possibly introducing unfairness amongst the players. The ob-
jective of this thesis is to study the mechanisms leading to loss of playability and fairness and propose novel solutions to improve the experience of players in online games.

1.2.1 Overview

Chapter 2 reviews the published material related to networked games and virtual environment development. Networked games are classified into four categories depending on their rules, level of interactivity and latency requirements. We highlight each category and present relevant studies on the effect of delay on players. Next, we review work on the various game communication architectures and available techniques to improve and optimise data exchange including multicast and interest management before introducing background on state synchronisation techniques. We identify the origins of current state synchronisation techniques from work on synchronisation of logical processes in Parallel Distributed Event Simulations (PDES) and draw the distinctions between optimistic and conservative approaches. Finally, this chapter discusses the proposed schemes found in the literature aiming at improving users’ experience in games and virtual environments.

Prior to the main contributing chapters, Chapter 3 introduces the necessary thesis terminology and framework of assumptions. We also present the actors of a networked game with their respective roles and expose the notions associated to game state such as initial conditions, game state distances and trajectories which are used in the following chapters.

Chapter 4 proposes a hierarchy of disturbances propagating from the telecommunication network to the players. In this analysis, delays, originating in the network and modulated by a synchronisation scheme, generate measurable inconsistencies. Players then perceive these inconsistencies through violations of the ideal laws of the game virtual environment. We also show how the main game architectures and synchronisation schemes fit and can be analysed with
this framework. Aspects of this Chapter were published in the November 2006 issue of “Communications of the ACM”[22].

Using the framework previously laid out, Chapter 5 explores the relationships between game violations and user experience defined as playability and fairness. In some circumstances, violations are shown to be tradable. We propose to use fuzzy logic utility functions to model the subjective perceptual impact of the violations of a game’s ideal laws. Although estimated utility functions are provided as examples, the empirical subjective studies required to generate such functions are outside the scope of this thesis. However, we expect that the adopted methodologies are quite flexible to model a large variety of practical situations. With the help of estimated utility functions, we show how violation trading enables to balance the impact of different violations in order to maximise the users’ game experience. This Chapter also introduces fairness, theoretically defined as the standard deviation of playability, and demonstrates how it affects players. To validate this definition, an experimental measure of fairness is shown to stay tightly correlated to our original definition. Parts of the contributions of this chapter were published at the international workshop on Networking Issues in Multimedia Entertainment (NIME) in the 3rd IEEE Communications and Networking Conference (CCNC) in January 2006 [21].

Chapter 6 uses findings from the two previous chapters to propose two enhancements over classic synchronisation schemes capable of improving users’ experience in online games. We first present NetGameSim, the simulator we developed in order to experiment different scenarios. Then the technique of violation trading presented in Chapter 5 is used to find the optimum synchronisation parameter for best user experience. Simulations show the optimum is found somewhere between truly conservative and totally optimistic strategies. These simulation results are shown to correlate with experimental data measured on live system by Liang in [92]. Further improvement is found by tailoring the synchronisation parameters to different game actions. Aspect of this chapter were published in the 2005 Autumn volume of the “Telecommu-
While Chapter 6 presented synchronisation (ie. software) improvements, Chapter 7 explores possible improvements of user experience via alteration of the topology of decision points. To start, the concept of Critical Inconsistency (CI) is defined and is used as the objective function for a server selection optimisation problem. We show that lower bound of the CRT can always be calculated but is not always reachable. The formulated problem is highly non linear so its optimal solution is found for three types of small networks using exhaustive search. This shows the presence of sweet spots for the position of decision points in some circumstances. The chapter then introduces a heuristic solution called "minimum critical inconsistency growth" which converges towards a set of servers with close to optimal critical inconsistency. Finally, the heuristic is shown to provide better results compared to three central server placement strategies. Results presented in this chapter were presented and published in the November 2006 “ACM Workshop on Network and system support for games” (NetGames) [23].

1.2.2 Contributions

The contributions of this thesis are:

A framework for game disturbance analysis.

The network infrastructure supporting online games is composed of the telecommunication network topology of hardware elements participating in the exchange of game state information and the software synchronisation used to maintain the game consistency. We analyse the propagation of disturbances from the network delays creating inconsistencies, which generate violations of the virtual world’s ideal laws. While this framework supports the rest of this thesis, it was built in a generic way and could be used to analyse other synchronisation protocols and network architectures.
Network playability definition and measure of its impact.

There is no clear and commonly accepted definition for playability in the literature, and this concept is often simplified to Response Time (RT) or Round Trip Time (RTT). We clearly define the notions of measurable network playability and propose to use fuzzy logic tools to assess the final impact of combined violations on player’s experience. These definitions can be used in simulators to evaluate playability and its impact.

Measure of a game network fairness.

The concept of game fairness for a game is even less discussed in the literature than playability. We propose fairness to be defined as the standard deviation of playability amongst participants. This theoretical definition is validated through a comparison to an experimental fairness measure based on game score.

Trading-off optimistic and conservative synchronisation.

Pure optimistic or conservative synchronisations are not always satisfactory in a given game. We show how the optimum playability can be found by tuning the synchronisation parameters to reach a trade-off between optimistic and conservative synchronisation.

Tailoring synchronisation parameters to the requirements of distinct action types.

Different in-game actions may represent very different concepts with different requirements in terms of synchronisation. We propose to unbind all actions from the same fate and show that tailoring the respective synchronisation parameters for different in-game actions can improve overall playability.

Formulation of a topology optimisation problem and introduction of critical inconsistency.
 Aside from software synchronisation, changing the topology of decision points also improve a game playability and fairness. We define the concept of critical inconsistency and prove that sets of decision point locations minimising this criteria tend to simultaneously provide good average playability and global game fairness. It is demonstrated that a lower bound for the critical inconsistency of a set of players is always calculable, but not necessarily reachable by a set of decision points. Subsequently, we formulate an integer programming problem aimed at optimising the location of decision points to attain the best user experience. The solution of this problem for small networks is presented.

**Heuristic solution to the decision point location problem.**

We develop a recursive heuristic solution to the previous problem for large networks which tend to minimise the critical inconsistency. The performance of the heuristic in large networks is compared to the calculable lower bound and the various central server selection strategies.

### 1.2.3 Peer reviewed publications


J. Brun, F. Safaei, and P. Boustead. “Fairness and playability in online multiplayer games” In *Proc. 2nd IEEE international workshop on Networking Issues*


Chapter 2

Literature and industry state of the art review

Synopsis

In this Chapter, we critically review the academic literature and documented industrial technologies relevant to networked games playability and fairness.

Content:

- The games industry
- Latency and user impact studies
- Communication architectures, state synchronisation and user experience improvement in the literature
2.1 Introduction

The history of computer games started with tennis for two [4] developed at Brookhaven National Laboratory in Upton, NY. Gradually, along with greater availability to the general public of computers and electronics under various forms, computer games have become the multi billion dollar fast growing industry it is today[40][41]. Following the boom of this industry, with some delay, academic game research is rapidly losing its “not serious enough” stigma and becoming a more recognised and active domain.

Networking aspect of games

Quickly after the popularisation of the Internet, network games entered the market scene. Today, most stand-alone games also offer some sort of network multiplayer mode. Game consoles from Sony, Microsoft and Nintendo have had online capabilities for two generations and now natively connect to their respective Internet game portals. Online games seem today like a very natural and popular technology to consumers of industrialised countries.

However, because games are interactive and have to run in real-time, they are highly demanding online applications when it comes to networking[134] and the original Internet design as an best effort[129] point-to-point service delivery does not natively satisfy requirements of networked games. In fact, since multiple players share the same virtual space, they need to exchange data in a multipoint-to-multipoint logical fashion. Also, the real time aspect of games make them very sensitive to latency from all sources such as:

Geographical distances create unavoidable propagation delay.

Processing speed not being infinite generates computation time when data needs to be processed.

Bandwidth [122] can become a limiting resource when network links get congested creating queueing delays in routers. This is particularly true for
user access bandwidth[82] which is typically more prone to congestion than core network transmission capacity.

**Organisation of the chapter**

First, we present the current state of the industry (Section 2.3) followed by a classification of games based on their latency requirements and research on the effect of delay on players in Section 2.4. Section 2.5 presents research done on modelling network inconsistencies in games and DVE. Section 2.6 discusses the main literature dealing with game communication architectures and state synchronisation; it includes an introduction to optimistic and conservative protocols and their origin in Parallel Distributed Event Simulations (PDES). Finally, Section 2.7 discusses work on techniques used to improve user experience in virtual environments and games.

### 2.2 Origin of networked game research

Military simulations initiated the research on network infrastructure for military simulations. Soon after, multiple industrial and academic Networked Virtual Environment (NVE) projects started to tackle different aspect of the network architecture and synchronisation issues.

#### 2.2.1 Military Simulations

The United State's Defence Advanced Research Projects Agency (DARPA) created the SIMNET [27] project in 1983 to interconnect multiple war simulators. Early SIMNET simulations could only support a limited number of users and objects and were heavily based on packet broadcasting. The need for specification for common state synchronisation between different systems in SIMNET directly lead to the development of the Distributed Interactive Simulation (DIS)[75] protocol.
In 1994, Macedonia et al. published details of NPSNET-IV[95][94], the DIS based environment built by the Naval Postgraduate School of Monterey, California. NPSNET implemented dead reckoning and Area of Interest (AoI) management to reduce the amount of transmitted data and could be used over the Internet using the MBone[86] multicast capabilities.

The Defence Modelling and Simulation Office (DMSO) later initiated the development of the High Level Architecture (HLA)[76] standard in order to improve inter-operability and code reuse of simulation components. Whereas DIS specifies a network protocol, the HLA provides a general set of rules on how federates (the HLA term for simulators) interact through a common Run-Time Infrastructure (RTI). The HLA standard has and is being used by multiple military and industrial NVEs and is still being extended at the time of writing.

2.2.2 Networked Virtual Environments

Networked Virtual Environments (NVEs) are also referred to with various names[38] in the literature such as: Distributed Virtual Environment (DVE), Collaborative Virtual Environment (CVE), Shared Virtual Environment (SVE), Multi-user Virtual Environments (MVE) or Distributed Synthetic Environment (DSE). All these labels essentially refer to a virtual environment shared across a telecommunication network by multiple users in different physical locations.

Research on NVEs started earlier than research on networked games. This delay was probably due to the "not serious enough" stigma attached to video games while communication, simulation and collaboration were perceived as promising applications for NVEs. Anyhow, online games and NVEs are tightly connected as networked games essentially are virtual environments dedicated to entertainment.

Multiple NVE projects using different network architecture have been implemented and tested, Table 2.1 list the main NVE of the 90's. The first platforms were mostly peer to peer (P2P) and implemented single ownership meaning that
each interactive entity in the environment would be owned and controlled by one single process at any given time. Early versions of interest management were used, usually tied to multicast groups to avoid sending updates to all participating machines. Many projects evolved successive versions over the years improving various aspects of their platform. One significant and interesting trend has been to move from the traditional P2P multicast to a client-server unicast communication model.

Notable advancement were found in Spline[8] which introduces “locales” as partitioning of the virtual environment. Similar concepts are currently in use in most online game today as discussed in next section. MASSIVE[66] focused on interactions and added the concepts of “auras” to entities specifying the minimum distance at which the entity could be interacted with. RING[59] used servers as update filter and relay instead of multicast groups. CALVIN[89] and its successor CAVERN[90] used a central database guaranteeing consistency and supporting a persistent environment.
<table>
<thead>
<tr>
<th>Project Name</th>
<th>Origin</th>
<th>Year</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVE[54]</td>
<td>Swedish Institute of Computer Science</td>
<td>1991</td>
<td>P2P multicast, evolved multicast groups</td>
</tr>
<tr>
<td>NPSNET[95][94]</td>
<td>Naval Postgraduate School of Monterey</td>
<td>1994</td>
<td>P2P multicast with groups</td>
</tr>
<tr>
<td>BrickNet[132]</td>
<td>National University of Singapore</td>
<td>1994</td>
<td>Client-server unicast, geometry and behaviour of entities are exchanged</td>
</tr>
<tr>
<td>MASSIVE[64]</td>
<td>University of Nottingham, UK</td>
<td>1994</td>
<td>P2P unicast, introduced area of interest refinements</td>
</tr>
<tr>
<td>RING[59]</td>
<td>AT&amp;T Bell Laboratories</td>
<td>1995</td>
<td>Servers selectively relay and filters entities updates to relevant clients</td>
</tr>
<tr>
<td>CALVIN[89]</td>
<td>University of Illinois</td>
<td>1995</td>
<td>Central database guaranteeing consistency and supporting persistent entities</td>
</tr>
<tr>
<td>Spline[8]</td>
<td>Mitsubishi Electric Research Laboratories</td>
<td>1996</td>
<td>P2P multicast with groups, introduces “locale” and relaxed consistency amongst peer</td>
</tr>
<tr>
<td>MASSIVE-2[65]</td>
<td>University of Nottingham, UK</td>
<td>1996</td>
<td>evolved to P2P multicast</td>
</tr>
<tr>
<td>Spline 3[143]</td>
<td>Mitsubishi Electric Research Laboratories</td>
<td>1997</td>
<td>evolved to hybrid P2P multicast with servers managing shared resources</td>
</tr>
<tr>
<td>MASSIVE-3[66]</td>
<td>University of Nottingham, UK</td>
<td>2000</td>
<td>evolved to client-Servers unicast, improve entities updating and ownership transfer</td>
</tr>
</tbody>
</table>
As non-entertainment NVEs did not reach the threshold of becoming useful enough to be marketable collaboration and communication tools, research and development efforts have slowed down in the area since the beginning of the millennium. From the mid 90s, networked games, over LAN and later over the Internet, have become increasingly popular. Since then, the online gaming industry has been growing steadily to become a profitable US$11.8 Billion worldwide market[39] in 2009. Advancements in online game technologies, which are the focus of the next section, are now opening the way to not-only-for-entertainment applications such as Second-Life[93].

2.3 The Game Industry

This section presents an overview of the background industrial technologies relevant to networked games playability and fairness. Although academic work is usually published, the proprietary nature of commercial products makes technical information on the implementation of commercial games difficult to find. Therefore, industrial examples and references are provided whenever possible; unfortunately the review in this area cannot be comprehensive because of the lack of published documentation from the game developing companies.

2.3.1 Network Architecture in the Game Industry

Although online games are related to NVEs, their key problems are different. First, games must be able to work properly over the Internet with no end to end QoS or multicast, through the limited access bandwidth available and the Network Address Translation (NAT) routers that gamers often use. Second, because most games are competitive, cheating can become an issue. On PC and Mac platforms, players install and run the game executable themselves hence the game terminal cannot be implicitly trusted. Game consoles do not suffer as much from this issue as modified consoles are typically detected and banned from the online game portal.
The central server architecture in which participants connect to a common game server immediately became the preferred solution adopted by game developers. On the game client, most if not all the possible game content is stored locally as files and can be updated out of game through patching. From a game state point of view, the game client only displays the environment and relays its player’s actions to the server. The game server is the unique authoritative decision point on the state of the entire environment, limiting synchronisation requirements and cheating opportunities. Only relevant updates are sent to the players, limiting both the required bandwidth and, again, the cheating possibilities.

For that reason, most commercial networked games have been using a central architecture with dedicated or non-dedicated servers. For example, Internet Service Providers (ISPs) often run dedicated servers for popular games. Many First Person Shooter (FPS) and Real Time Strategy (RTS) games on PC and Mac enable users to open their own server to play on a LAN or over the Internet. Also, most game sessions on Xbox Live are hosted by one of the participant’s console with or without their knowledge[144].

At the time of writing, no Massively Multiplayer Online game (MMO game or MMOG) is known for using anything else but a central server approach. Such games use Resource Driven Distribution (RDD) techniques such as distributed locale on dedicated clusters to scale the number of users.

**Distribution of Locales**

The concept of partitioning the virtual world in “locale” was introduced in NVEs with Spline[8]. Today, dividing the virtual world into distinct manageable areas is one of the main technique used to distribute the game networking and processing load across multiple servers. Locales may be static and assigned to fixed machines or may be dynamically altered on the fly using floating boundaries depending on the load [30]. For example, Second-Life from LindenLabs[93] uses static locales whereas the BigWorld distributed servers[14] manage dynamic
locales.

Another variation of locale distribution is the creation of private “instances” for a specific group of players. Each instance is an alternate version of a specific area in the virtual world which can be managed by a different machine. Like in two parallel worlds, players in two different instances of the same area cannot interact with each other, but can meet again after leaving their respective instance and joining a common area. *Guild Wars* [109] for example, is a game heavily based on instances, however, instances can be found in most recent Massively Multiplayer Online Role Playing Games (MMORPGs) today.

Currently, most subscription based commercial massively multiplayer games, such as *World of Warcraft* [18] or *Lineage II* [110], deploy independent virtual worlds on different continents to reach their clients. This solution provides better network latency to the users since they connect to geographically closer servers but it prevents any type of interaction across the independent virtual worlds.

When using locales, even if multiple machines are used to manage one game, these approaches still fundamentally stay in the realm of the central server architecture. At a given point in time and space in the virtual world, one server is the unique decision point for all actions; consequently, no game state synchronisation is ever required and paradoxes cannot occur\(^1\). For this reason, the schemes described above are considered being “central” architectures and not “distributed” by the literature.

### 2.3.2 Distributed architectures in the game industry

There are very few commercial networked game using peer to peer and none known using distributed servers. *Age of Empires I* [103] and *II* are real time strategy games created by Ensemble Studios [44] which uses a full peer to peer (P2P) architecture when played online. However, NAT were not supported and

\(^1\) cf. 2.6.1, p.37
cheating was mitigated by dropping peers showing “out of sync” game state[13]. This approach seems to work with some RTS games which only host a limited number of players and are less demanding in terms of responsiveness[33] than other type of networked games.

The game giant Electronic Arts attempted to make *The Sims Online* (the early version of the now discontinued *EA-land*[61]) a full peer to peer MMOG. The initial architecture was abandoned due to the issues involved in synchronising large number of peers[84] and the game was ultimately re-engineered and deployed in a regular client-server fashion.

### 2.3.3 Hiding and compensating latency

As latency is a particularly big problem for fast pace games, FPS game developers, in particular, use techniques we list here to hide or compensate the latency introduced by the network.

Most multiplayer networked games use some form of *dead reckoning*. This technique already used in DVEs and military simulation consist in the game client guessing the state of entities it does not control. Dead reckoning is also largely discussed in the academic literature and will be treated in Section 2.7.1, p.43.

Valve documented in [12] the implementation of what they call “client side prediction” used in their blockbuster FPS *Half-Life*[140]. Armitage, Claypool and Branch discuss the technique in [6] (Chapt 6, pp 87) under the label "player prediction". This method Client-side prediction is another common approach used in modern fast pace game to hide latency. The game client tries to predict the outcome of its player’s actions by using the same logic as the game server. The prediction is used for immediate display and hides the latency due to the delayed server response. If the prediction turned out to be wrong, the game server update corrects the game client state and its display. In this thesis we prefer to refer to this technique as "co-simulation" to reflect that the client is
simulating using the same algorithm as the server and to avoid confusion with
deal reckoning which is also referred to sometimes as a type of “prediction”.

In the same paper[12], Valve documented another scheme they called “lag
compensation”. It uses timestamps on game state updates to execute user
actions back in time in the context the player perceived the game state. This
technique is also described by Armitage et al. in [6] (Chapt 6, pp 94) as a
time manipulation. The down side is that such approach may lead to the “shot
around the corner” problem. The issue is named after the situation where a
player move behind a wall for cover and get told by the server that it got shot
only after reaching the safe area, creating the feeling of being shot around the
corner of the wall.

2.3.4 Directions of the game industry

The game industry is bound to follows the available technologies and market
opportunities to provides best value games with limited cost.

P2P, NAT and IPv6

From a networking point a view, the increasing access bandwidth available
to end users could motivate pushing towards more P2P games architectures
as a way to reduce data centre costs. However, NAT routers used by most
ISP subscribers prevent reliable P2P connections. Possibly, the situation may
change when NAT becomes deprecated by the deployment of IPv6 as a solution
to the IPv4 address shortage.

Mobile games

In the mobile space, hardware improvements led to the commercialisation
of smartphones more powerful than desktop computers were in the previous
decade at the same time as Internet access over mobile network has become
affordable. The launch of the AppleStore has provided a simple way for virtually
any programmer to develop and sell applications to any iPhone user. iPhone
gaming has since become a huge market which is expanding on phones pow­
ered by Android and Windows operating systems. Recently, the success of the
iPad by Apple is inspiring others into developing competing products. With
their large display, these tablet devices are the newest promising entertainment
mobile platforms.

Although smartphones and tablets do not lack network capabilities, most mobile
games today are stand-alone. As this market is evolving quickly, new technolo­
gies and services such as cloud gaming may enable these devices to become
major online game platforms in the future.

Cloud gaming
A handful of companies have been announcing the introduction of cloud gaming.
In the cloud gaming paradigm, the game client is executed on a specialised
server which receives the mouse and keyboard (or other device) inputs from
the player’s machine and returns the graphics and audio in a video stream.

OnLive[113] service is currently operating in the US only and supports Windows
and Mac systems as well as a MicroConsole that can be plugged into a TV.
OTOY[114], another US based company promises the similar services via a
proprietary web plug-in. At the time of writing, Gaikai[60], is in beta test
phase and has demonstrated multiple games running on standard Flash and
Java plug-in. Gaikai targets its services to game publishers for “try before you
buy” playable game demonstrations. Another company, Playcast [123], aims
at providing cloud gaming on televisions via IPTV providers and has started a
pilot test in July 2009.

The advantage of this approach is that players do not need to install the game
or even buy expensive hardware as the processing is done on the cloud gaming
servers. The reduced hardware requirements also enables lightweight devices
such as netbook, tablets or even smartphones to play the latest graphics pro-
cessing hungry games as long as they can decode the video stream.

On the other hand, the access bandwidth of the user must be good enough to allow high quality video streaming. Furthermore, the geographical distance between the player and its cloud gaming server introduces an unwelcome additional delay. OnLive [113] is, at the time of writing, the only company operating game on demand. Their service requires users to have a minimum of 5MB/s download bandwidth to accommodate the video stream and not being located further than 1600km from the OnLive servers to experience reasonable delays.

With the game clients in the cloud, the network scenario is changed significantly. As the game client cannot be tampered with by players it becomes trustable. Also, machines running the game client can have real IP address (i.e. not be behind a NAT router); and if all clients in a game have a routable IP address, proper P2P communication become possible. Because it locates the game clients in a data centre, cloud gaming relaxes the two constraints which prevented the game industry from developing and using P2P architectures. Also, one can imagine using QoS to further reduce latency between game clients as it may be available in data centres. The down side is that a game must be specifically developed for cloud gaming services to use a trusted P2P architecture.

2.3.5 Conclusion on the state of the industry

The game industry is driven by requirements for acceptable cost to benefit ratio and widely opted for the the central server architecture as it is the most practical approach. Game companies use Resource Driven Distribution (RDD), such as locale distribution and its variants, to overcome the scalability issues of the central server approach. They also developed some specialised techniques to hide or compensate network latency.

As the current boom of this industry shows, this strategy has been a success so far. On the other hand, there have been little private research and invest-
ment towards true distributed architectures. Mobile gaming is emerging as a significant market but most mobile games are not multiplayer yet. The cloud gaming paradigm has the potential to modify the game industry in many ways if the technology becomes widely adopted.

The rest of this chapter will focus on the academic literature: Section 2.7 discusses the research on hiding and compensation techniques whereas the next section classifies games by types to present published material available about latency impact on users.

### 2.4 Impact of network imperfections

Game developers are aware of the impact of telecommunication network on their products [12][13] and usually do address these influences in some ways. However, no public documentation on evaluation techniques or tangible results from industrial sources can be found on this subject. A few academic research groups have published in this area in order to understand and attempt to measure the influences of the network on game players.

First, the section introduces a classification of online games in four genres. We then summarise, for each of the four game genre, the conclusions achieved by empirical studies on the impact of network on players. The last part of this section describes the different attempts published in the literature to model how delays impact users.

#### 2.4.1 Classifying network games

The three main imperfections introduced by the telecommunication infrastructure are: delay, jitter and packet loss. Their final impact on gamers can vary widely depending on the type of game, the compensation techniques implemented and the design choices of the game developers.
Different rules and aims in games lead to different network requirements. Mark and Kajal Claypool\cite{34} classify games based on the player perception (first person, third person and omnipresent) to explain the differences in latency requirements. This categorisation roughly matches the traditional organisation of the gaming industry and gamer community which tend to classify game based on their aim and rules into three main categories which are: First Person Shooters (FPS, typically in first person view), Role Playing Games (RPG, typically in third person view) and Real Time Strategy (RTS, typically in omnipresent view). We add a fourth category for games not falling in any of the other class such as many sport simulations which often combine omnipresent and third person views.

### 2.4.2 First Person Shooters

In First Person Shooters (FPS) participants play in the first person, that is they ‘see’ the game through the eye of the avatar representing them in the virtual environment. Players typically access an arsenal of weapons they use to complete personal or team objectives. Usually limited to a few tens of players, this class of game requires hand-eye coordination and is extremely delay sensitive. \textit{Quake\textsc{III}}, \textit{Half Life2}/\textit{Counter Strike}\cite{140} and \textit{Unreal Tournament}\cite{46} are popular examples of FPS games.

Henderson\cite{71,72} found that absolute delay bounds appeared less important than relative delay between players in Half-Life, suggesting fairness in delays is important to gamers. However, the study admits that most players connected to servers providing them with delay considered as acceptable by other studies. Following work on the same game\cite{73} suggests that players select their servers based on the measured round trip time (RTT), but are willing to accept significant delay degradation once connected.

An experiment of Armitage \cite{5} with two \textit{Quake\textsc{III}} servers in different locations confirms that players preferably connect to servers less than 150 to 180\textit{ms}
away. In a comparative survey of players on Halo and Quake III[149] Zander and Armitage found that good (and presumably more experienced) players are more aware of QoS issues and cope better with degraded conditions than bad players. The paper concludes that delay has a higher impact than packet loss and that differences in QoS lead to unfairness.

Quax studied the objective and subjective effects of the network conditions on players in Unreal Tournament 2003[45] in [124] and [124]. The author concluded that delay above 60\text{ms} Round Trip Time (RTT) disturbed players. In a study of the same game[11], Beigbeder et al. find that latency has no effect on player’s movements but decreases shooting accuracy as early as 70-100ms RTT. RTT of 200\text{ms} is described as annoying.

Although numbers vary a little, academic studies confirm the hypothesis that the FPS genre is not very tolerant of network delay.

### 2.4.3 Role Playing Games

Role Playing Game (RPG) are usually played in ‘third person’ which means participants see their avatar from a third person view. Usually picturing a fantasy or science fiction world, these games often include a story telling aspect and are usually more social than other types of on-line games. Some online RPG titles such as, World of Warcraft[18] or Lineage II[110] are considered "massively multiplayer" and can accommodate many thousands of players in the same virtual world. These games are also referred to as Massively Multiplayer Online RPG or MMORPG.

In their field study[55] of Everquest II[135], Fritsch, Ritter and Schiller confirmed that latency affects unequally different types of actions in the games: typically combat situations cope less well than simple movement but the game was found to stay playable up to 1250\text{ms} of latency.

Chen et al. analysed the relationship between network QoS and game play-
ing time through real packet traces of the Taiwanese game *Shen-zhou Online*\[32\][31]. Results show a negative correlation between the playing session time and the following QoS parameters: RTT, jitter (as standard deviation of RTT) and packet loss. The authors explained that Shen-zhou players are more sensitive to packet loss than RTT because the game uses TCP which incurs large delay variations when packets get lost.

In accordance with the general beliefs in the gaming communities, academic studies agree that RPGs are moderately sensitive to network delay.

### 2.4.4 Real Time Strategy

In Real Time Strategy (RTS) games, participants control groups of avatars, often armies, to achieve specific objectives. RTS games are usually more delay tolerant than FPS or RPG especially when they are turn based. Example of RTS games include *StarCraft*[16], *Age of Empire*[104] series or even online Chess.

Claypool et al. observed the impact of network latency in *WarcraftIII*[17], *Age of Mythology* and *Command and Conquer* in two publications [131][33]. The studies differentiated three distinct interaction types, namely building, exploring and combat. For each interaction type, a map scenario was designed to test how well players would perform in the specific interaction. Scenarios were repeated under various artificially controlled network delay conditions. It was observed that, although latency became subjectively perceptible from 500ms, it had little influence on the objective outcome of any of the scenarios in any of the tested games. The authors interpret their result explaining that the strategic aspect is much more important than real time interaction between players in RTS, making them more resilient to delay than other online games.

This sole study confirm the beliefs from the gaming communities that RTS are more resilient to latency than other online game genre.
2.4.5 Sport simulations

Sport games can simulate all types of sport contest such as car racing, football, rugby, tennis, ice hockey and much more. Depending on the sport rules and the implementation of the game, hand-eye coordination may also be required as in FPS.

Nichols and Claypool [111] experimented with Madden NFL 2004 on the PlayStation2 console and deduced that it implemented some client side prediction minimising the impact on user with latency up to 500ms at which level noticeable performance loss occurred.

In a car racing game, Pantel et al. studied the influence of artificial delay on players[120]. How often players would drive off the course and lap times were recorded. Results show that both lap time and number of off-course departure strongly correlated with the introduced delay. The authors concluded delays under 50ms are insignificant, 100ms or more should be avoided, 200ms makes the simulation unrealistic and 500ms makes the game unplayable.

2.4.6 Concluding remarks on empirical studies

Mark and Kajal Claypool proposed a "precision-deadline" model to classify in-game actions. In this model, actions are assigned a precision rating representative of the accuracy required to perform the action, and a deadline rating related to the amount of time in which action must be accomplished in order to achieve its intended goal. The authors observed the sensitivity of game types to latency matched the classification of their action in the precision-deadline model: actions from FPS tend to require high precision and short deadlines while actions in RPG required less stringent constraints and finally the deadline of most RTS actions were loose.

Dick, Wellnitz and Wolf [42] measured a player sensitivity to network delay in FPS, RTS and sport genre in-line with the values obtained by the other research
groups mentioned above. In most of the experiments, the authors also correlate player's skills to game score more strongly than network latency to game score. The paper also conclude that different FPS games show different sensitivity to network conditions.

The influence of player skills is also demonstrated by Pantel et al. who showed in [120] that better drivers are less affected by latency. The variations in sensitivity to network delay between different FPS games concur with the results from Zanders et al. in [149] which clearly show that players in Halo and Quake 3 are affected differently by artificial network manipulations.

In conclusion, the amount of published material dedicated to the impact of network imperfections on game players is limited. In accordance with opinions in the gaming community, evidences support the hypothesis that FPS are very delay sensitive, RPG can tolerate more delay than FPS and that RTS are the least sensitive games.

2.5 Inconsistency models

The previous section has been presenting a few academic publication empirically studying the effects of network on players in online games. Other research groups have attempted to analyse how latency affects or DVE or networked game users. The central notions of "inconsistencies" and/or "perception" are often used, however they are usually defined to suit a particular context.

In a discussion of P2P DVEs[152], Zhou et al. distinguish time-space inconsistency and violation of causal order as the two classes of inconsistencies. The authors argue that the violation of causal order of events is not as crucial in DVE as it is in Parallel Discrete Event Simulations (PDES). The article focuses on quantifying the time-space inconsistency which is defined as the divergence between the perception of the same object by the different actors of the DVE. In a subsequent publication [151], Zhou and Shen define a consistency model
for online games where the system must provide the same perception of the game state at all sites. The authors relax this constraint by taking into account human spatial and temporal perception threshold. The paper however limit its scope of game state to the position of objects and avatars in the virtual environment.

In [139], Vaghim, Greenhalg and Benford differentiate three types of consistencies. Presentation inconsistency is defined as the matching of the virtual world perception amongst the users; physics consistency as the matching of the world behaviour to its ideal rules and finally interaction inconsistency as the matching between the users' actions and the world reactions. Following the empirical study of the effect of latency on a simple two-player game, the authors argue that the presence and effect of delays should be made aware to participants in order to devise strategies to cope with the situation.

Griwodz et al. specifically analyse inconsistencies in distributed applications in order to create a list of relevant QoS network parameters[68]. In this publication, a distributed application is defined as providing perceived consistency if the perceived output is interpreted equally by all users. In a subsequent contribution [115], Palant, Griwodz and Halvorsen favour the users' perspective and differentiate three types of inconsistencies. Physical consistency is defined as the compliance of events to the virtual world's physical model. Maintaining state consistency requires that "any change in game state that is important to more than one player has to be signalled to either every player concerned with reasonably low difference in arrival times or to none of them". Finally, to maintain reaction consistency, a player's actions should not "appear unreasonable in other players' views unless those actions are also unreasonable in the player's own view".

In a publication presenting reusable and replaceable consistency maintenance modules for client server architecture[53], Fletcher, Graham and Wolfe distinguish four components to consistency. Fidelity represents the degree of matching between the client representation of the world and the canonical game state,
feedback time is the time taken for a player to see his action realised, degree of warping is defined as the magnitude and frequency of game state update discontinuity and finally animation rate is the frequency of position update on the player’s display.

Different research groups have presented various model of classification of inconsistencies. Some classification only suit a particular synchronisation scheme and/or topology setup or consider only a subset of possible game state parameters such as the position of entities. In these cases, the consistency classification are not generalisable. Other published classifications provide loose definitions which are difficult to translate into a concrete model of inconsistencies. Chapter 4 of this thesis aims to fill the gap in the literature by providing a framework for the analysis of network disturbances in games.

2.6 Communication Architectures and Synchronisation

The published work presented in section 2.4 discussed the objective and subjective impact of the network conditions on players in specific commercial games. Although the game type largely influences the sensitivity of player to latency, studies have also shown that games of the same type can show variations in their players’ sensitivity to network conditions. Such variations can be explained by the difference in the game implementation.

Communication architectures

The major aspects of game implementation affecting the sensitivity to network conditions are the communication architecture and the synchronisation scheme in use. Figure 2.1, p.32 illustrates the three main classes of communication architecture namely central server, Peer to Peer (P2P) and hybrid.

In a central architecture (Figure 2.1a, p.32), participants connect to a server
which is the authoritative decision point for all actions in the game. The server is also responsible for the dissemination of the game state data to the terminals used for the game rendering. This architecture is technically the easiest to implement since the consistency of the game state is guaranteed by the presence of one unique decision point for all actions modifying the game state. Hence, no state synchronisation is required. Section 2.3 p.17 discussed the state of the game industry and explained that most commercial games use a central architecture.

Peer to peer (P2P) games (Figure 2.1, p.32) have an opposite approach in comparison to central server schemes. Instead of a unique decision point in charge, all players’ machines, called peers, are authoritative on the game state in pure P2P environments. Many NVE projects were based on P2P communication (cf. 2.2 p.13) while P2P based games such as Age of Empire [13] are the exception in the game industry (cf. 2.3 p.17). On the other hand, multiple P2P academic game platforms such as MiMaze [43] and SimMud [31] have been developed for research purposes. Research on synchronisation requirements and

\footnote{Apart from some rare case such as latency compensation seen in 4.3.2 p.70}
solutions for distributed systems is discussed later in this Section.

Hybrid solutions (Figure 2.1b, p.32), usually referred as distributed\textsuperscript{[88]}, mirrored\textsuperscript{[37]}, proxy\textsuperscript{[107]} servers or hybrid architectures\textsuperscript{[58]}, uses a constellation of machines to which participants connect in a regular client/server fashion.

**Synchronisation issue in distributed architectures**

Distributed architectures involve the presence of multiple machines making authoritative decisions on the same game state data. To “agree” on the state of the virtual environment, these decision points need to stay synchronised. The difficulties in implementing and deploying distributed games largely explain the non-adoption of distributed architectures in the industry. Interestingly, these same fundamental issues are perceived as challenging by academia and have been driving the areas of network games and distributed virtual environment research in the past few years.

First, this section will introduce the distribution problem through a simple example. Next, and before discussing the various synchronisation schemes in the literature, we will summarise background synchronisation concepts from the older research field of Discrete Event Simulation (DES).

### 2.6.1 An introductory example to the synchronisation problem

This example illustrates the benefits and issues in distributing game servers. It explains the notions of response time, inconsistency, paradox and local lag. The network delay between elements $a$ and $b$ is referred to as $D(a, b)$ and represents the time it takes for a packet to travel directly from $a$ to $b$. A variation of this indicator is $D_S(a, b)$ which represents the time it takes to travel from $a$ to $b$ through a server $S$.

Players $P1$ and $P2$ want to play a given network game together; let's study
**Figure 2.2:** A two players game: central server or distributed architectures

different scenario cases.

**Central Server**

In this first case, the game is available through a central server as shown in Figure 2.2a.

When P1 performs an action, an action order is emitted from his terminal toward the central server S. S makes a decision about this action and then sends an answer back to P1 and if necessary a game update to P2 as shown in Figure 2.3. Let us calculate the Round Trip Time (RTT) of a player when using a central server:

\[
RTT_{central}(P) = 2 \times D(P, S)
\]  

(2.1)

In this case, with two players, the average response time for each player is:

\[
\overline{RTT}_{central} = \frac{1}{2} \times (2 \times D(P_1, S) + 2 \times D(P_2, S))
\]

\[= 2 \times \overline{D}(P, S) = D_S(P_1, P_2)
\]  

(2.2)


\section*{Distributed Servers}

In this second case, the game runs as a distributed application on two servers \( S_1 \) and \( S_2 \). \( P_1 \) connects naturally to the closest server \( S_1 \) and \( P_2 \) to \( S_2 \) as shown in Figure 2.2b. The notation \( P.server \) refers to the server which player \( P \) is connected to. We assume \( S_1 \) to be on the direct path between \( P_1 \) and \( S \) and \( S_2 \) on the direct path between \( P_2 \) and \( S \). Therefore, \( D(P, P.server) \leq D(P, S), \forall P \). Both servers are authoritative on the virtual world and synchronise their game states exchanging their action events in synchronisation packets (Figure 2.4). Therefore the round trip time of each player is:

\[ RTT_{Distributed}(P) = 2 \times D(P, P.server) \quad (2.3) \]
Figure 2.4: Distributed servers time diagram (no paradox avoidance) The server S1 resolves the action of player P1 as soon as it has received its action order and forward the event message to S2. The response time of P1 is equal to its RTT to S1. Server S2 is inconsistent with S1 for the time it take the event message to reach it.

The average response time is:

\[
\text{RTT}_{\text{Distributed}} = 2 \cdot D(P, P, \text{server}) = \frac{1}{2} \cdot (2 \cdot D(P1, S1) + 2 \cdot D(P2, S2)) = D_S(P1, P2) - D_S(S1, S2)
\]

The average round trip time is reduced by \( D_S(S1, S2) \). However, game states of servers S1 and S2 can now become inconsistent due to the propagation delay of the synchronisation messages. A game state inconsistency is a divergence in the value of one or more game state parameters between two servers due to the propagation time of an event. The game state of the inconsistent server shall be corrected once the event synchronisation message reaches it.
We define a paradox as a decision made by an inconsistent server which is incompatible with the decision it would have made if it was consistent.

A paradoxical scenario

Let us consider a scenario of this distributed game involving the two players $P_1$ and $P_2$ and a simple game object, we call “Schrodinger’s cat”. The delay between the two servers is $D(S_1, S_2) = 200ms$. At time $t_0 = 0ms$, $P_1$ shoots $P_2$’s avatar. From time $t_0$ to $t_2 = 200ms$, while $S_1$’s event message about $P_2$’s death is still travelling on the network, $S_2$ is inconsistent about $P_2$’s life state and believes that $P_2$ is alive. During this short window, at time $t_1 = 100ms$ and in response to the action by the player $P_2$ shooting the cat, $S_2$ decides that the cat is dead and forward the event to $S_1$. Let us see what are the game states on both servers: for $t \geq 200ms$, $P_2$’s avatar is definitely dead on $S_1$ and $S_2$. However, from $S_1$’s perspective, $P_2$ did not have the time to shoot the cat and from $S_2$ perspective $P_2$ killed the cat before dying. Schrodinger’s cat is dead and alive at the same time: this is a paradox.

For the virtual world to make sense, paradoxes should be avoided as much as possible and in the worst case, they must be quickly corrected. This is where state synchronisation strategies enter into play. The origin of state synchronisation schemes in network games and virtual environments can be found in the area of Discrete Event Simulations (DES) research which is discussed next.

2.6.2 Distributed computing and simulations

Like any type of virtual world, a networked game, is a simulated environment. At the human scale it is supposed to be real time, or at least game developers attempt to make it look as real time as possible. When decisions about the online game state can be taken in two or more locations, we are in the presence of a distributed network game which becomes a specific type of Parallel Discrete Event Simulation (PDES)[138]. Therefore, in addition to the nearly real time constraints specific to interactive applications, distributed games inherit all the
synchronisation issues involved in PDES. For comparison with the research done in distributed games, it is interesting to examine the synchronisation strategies used in this much older field.

Synchronisation mechanisms in distributed simulations[57] come in two flavours[56] to ensure smooth running of distributed simulations: conservative and optimistic. The conservative approach[105] ensures that each involved Logical Process (LP) safely executes any event: no event is executed on a LP before making sure all previous events in virtual time have been already executed. Hence, causality is never violated and no impossible state, the equivalent of game paradox, can appear. Optimistic simulations[81] set recovery checkpoints during simulation and allow LPs to execute events which might violate causality. If such a violation generates an impossible state, the simulation rolls back to the last safe checkpoint state to heal.

There is no consensus about which synchronisation is better and while some have warned that using optimistic techniques may lead to totally incorrect outcomes[112], others have argued that it is possible to run distributed simulations without any checkpoints and still get useful results [126]. Therefore, the optimum synchronisation parameters actually depends on the characteristics and requirements of the simulated model.

2.6.3 Consistency and Synchronisation in distributed games

Single and shared entity ownership models can be distinguished. In single ownership systems, a game state variable can only be written by one unique process which becomes authoritative on any decision affecting this game state variable. Other decision processes may hold a shadow copy of the entity but cannot modify it directly. In shared ownership systems, game entities may be modified by multiple peers or servers.
Shared ownership synchronisation

In distributed games, the state synchronisation process must avoid or get rid of paradoxes in a similar manner to distributed simulations. However, this must be done in real time in the eyes of players. The Timewarp[20][99] technique is the optimistic approach in which the system hopes that no paradoxes will happen. If one (or more) paradox occur, it is repaired by “warping” back in time and restoring event causality. The process is similar to optimistic synchronisation in DES[81].

Cronin et. al. developed and tested in [37],[35] and [36] a distributed version of a Quake III server. The entire game state is replicated on each “mirrored” server. The associated Trailer State Synchronisation is optimistic and maintains a few trailing game states to fall back on if a paradox was to occur.

On the other hand, Local lag is an example of the conservative approach: an artificial delay (lag) is introduced on each distributed server or peer delaying the execution of actions and prevents the generation of paradoxes. The usage of a Local Lag[96][100][141] covering the maximum delay between any two servers, as recommended by Mauve et al. [99], ensures a perfect game state consistency among all the servers.

Δ-synchronisation[79][80][77][78] where events are queued in a buffer and reordered (or discarded) based on their generation time and a maximum age Δ, is related to Local Lag and can maintain causality, assuming all sources and destinations have synchronised clocks.

However, in the same way that conservative techniques hinder computing parallelization [56] in distributed simulations, local lag, lock step or any other conservative synchronisation scheme, introduces additional delay deteriorating playability. Optimistic synchronisation can also be used in which case state healing mechanism must be applied such as the Timewarp algorithm which ensures state convergence and maintain decision correctness [99].
The same choice between conservative and optimistic schemes arise in distributed games. This discussion is crucial because synchronisation parameters affect both responsiveness and inconsistencies which impacts a game’s playability and fairness. Different trade-offs in game synchronisation are discussed in Chapter 6, p.119.

**Single ownership**

Single ownership simplifies synchronisation but is restricted in the type of decision it can make without introducing possible artefacts. In general, game events entangling two or more state variables can not be allowed in single ownership systems unless all these variables are owned by the same process. For example let us consider an avatar picking up a medkit found on the ground. At the same time that its health goes up, the medkit must disappear from the map as the item is being consumed. Hence the decision must be made by a process owning both the avatar life state and the medkit entity. Furthermore, because event processing may be based on shadow copies of other game state variables, the event outcome is based on the local view of the deciding process which may be out of date. Mauve described in [97] how such scenario can lead to undetectable causality violation. In that sense, synchronisation in single ownership system diverge from DES techniques. Section 4.3.2, p.76 discuss this type of synchronisation in more details.

Diot and Gauthier presented MiMaze[43][63] a fully peer to peer game using multicast and a simple bucket synchronisation mechanism. This game tends to be conservative as it adds delay to cluster actions done at similar wall-clock time into the same buckets. However, dead reckoning is used to supply information missing from buckets which can create a drift amongst the game states. Ultimately, each peer is authoritative on the game state relevant to its own avatar since it disseminates this information itself: for example if, in a paradox, an avatar was shot on one or more peers -because of some drift but not on the original terminal, the next update would re-synchronise all peers
with a living avatar.

Muller et al. also propose a proxy game model[107] where proxies actually are distributed servers which maintain a complete version of the game state each. Consistency is maintained by authorising only one process to have ownership and modify any given game state variable. Other processes only have a replica or shadow copy of the variable they do not own. A following publication [106] presents a Qfusion[136] implementation of a similar model featuring a Local Lag and Timerwarp schemes to avoid or correct inconsistencies.

A experimental P2P game, SimMUD[85], was developed by Knutsson based on the P2P overlay Pastry[128] and its associated multicast infrastructure Scribe[28]. In this implementation, player states are managed by their own peers while other generic object entities are assigned randomly to a authoritative “coordinator”. Hampel, Bopp and Hinn proposed a similar Pastry-Scribe P2P architecture targeted at MMOG in [70] without showing any implementation.

Yamamoto et al. expose in [145] a P2P publish/subscribe event delivery system in which each peer takes charge of a small portion of the virtual environment space and related variables.

Aggarwal et al proposed in [2] a proxy based game model into which authority on entities or subset of the game state can be assigned to either the game client if there is no risk of creating inconsistencies or otherwise to a central server. Proxies maintain portions of the game state themselves in order to manage server hand-off and forward game update more efficiently.

A similar scheme has been proposed by Kawano and Yonekura [83] where non-avatar entities switch process-owner based on their position in the environment. A demonstration of the technique was presented in a peer to peer virtual air-hockey game.

In both MASSIVE[66] and Spline/ISTP[142], exclusive update rights to each
objects in the world are given to a unique designated owner process in order to maintain consistency.

### 2.6.4 Conclusion on synchronisation

There are many different ways to synchronise game states across authoritative decision points. Each technique has fundamental properties which affect the virtual environment’s behaviour. Single ownership simplifies the synchronisation issue, but assigning each game state variables to a single logical process either limits the scope of actions to the variables owned by each logical process or allows incorrect decisions (in the correctness sense of Mauve [99]) to take place. Shared ownership synchronisation relates to synchronisation in DES and can be implemented conservatively, focusing on maintaining state consistency or optimistically to provide better interactivity.

### 2.7 Improving user experience

Beside the research in communication architecture and synchronisation protocols, other research groups have been studying various aspect of improving the impact of the telecommunication network infrastructure on the user experience. Aside from the artistic design and originality of an online game, the network aspect of playability and fairness is of crucial importance to give the participants an enjoyable game experience. This section presents aspects of user experience treated in the academic literature which have not been covered earlier in the chapter:

- Dead reckoning and client-server synchronisation
- Cheating prevention and fairness enforcing protocols
- Playability and fairness through event queue management
2.7.1 Dead reckoning

One method to hide latency to users in the central server architecture is to use Dead Reckoning (also called "opponent prediction" [6]). These algorithms extrapolate the state of an entity in the virtual environment by feeding its last known state to a prediction function. Typically, the scheme is used to do short term predictions on the position of entities in a virtual world. Although there is no reference to it, the same technique could be used to extrapolate any other continuous predictable game state variable. Not only dead reckoning can reduce the perception error of users, it can also significantly reduce the amount of information to be transmitted [133][25].

Shortcomings of dead reckoning have been discussed by Mauve in [97] where the author explains why, in single ownership systems, performing predictions on a machine authoritative on the game state may lead to the creation of paradoxes such as a "flying tank". In this example, the process owning a mine uses dead reckoning on the position of a tank to determine if the mine should explode. Error in the dead reckoning prediction may prevent detecting that the tank drove on the mine, allowing the vehicle to fly above the explosive without getting destroyed.

Pantel and Wolf in [121] show how the accuracy of prediction algorithms depends on the style of players and conclude that dead reckoning prediction can be useful for fast pace games such as car simulations.

Improving dead reckoning fairness

Virtually all proposed techniques to improve fairness from a network point of view in a central architecture involves degrading the playability of better off players towards the playability level of the worse-off ones. Zanders, Leeder and Armitage[150] designed the Self-Adjusting Game Lagging Utility (SAGLU) to enforce a minimum delay to all participants equal to the highest player delay below a preset acceptable threshold. Aggarwal et al. proposed in [3] a fairness
scheme by evaluating and equalising the dead reckoning error amongst players. Even though this study was based on a distributed game, the same principle can be applied to central architectures as well.

2.7.2 Fairness enforcing protocols

Fairness in online game has no commonly accepted definition and different research groups usually discuss fairness concepts in specific ways that suit a particular situation or model. Most work in the area of cheating prevention can also be classified as network fairness enforcement if the concept of cheating used is broad enough to include network delay manipulation: if a synchronisation scheme actively enforces identical delay on all participants it also enforces game fairness in relation to network delay.

For example, Lin, Guo and Paul presented in [148] “Sync-MS”, a synchronisation scheme for central server games aiming at improving fairness with the help of proxies artificially delaying update and action packets. This work was extended in [69] with the definition of the “Fair event ordering”. The associated framework delays the events in proxies to enforce the fair ordering of action based on the measured reaction time of players.

A secured lock-step protocol was presented by Baughman and Levine[10] which introduces a delay penalty for enforcing fairness and prevents look ahead cheats. In the same publication, they introduced “asynchronous synchronisation” which enforce a lock-step style synchronisation only when interaction is required. The New Event Ordering (NEO) protocol[62], from Gauthierdickey et al., manages to maintain a cheat proof and fair game with less delay penalty than lock-step. This scheme breaks actions into rounds during which each peer sends their actions to all other peers in an encrypted form along with the key to decrypt actions from the past round. In the similar way, Chen and Maheswaran proposed in [29] the use of an independent machine called “pulser” to distribute encryption/decryption keys to all peers which ensure that all game clients can
decrypt other player’s actions at the same time.

2.7.3 Event Queueing management

Ferretti, Palazzi, Roccetti et al. have presented in a series of publications and two Ph.D theses\cite{47}\cite{116} various event management schemes which can improve interactivity and/or fairness in mirrored servers or P2P architectures where processing time is scarce. Ferretti and Roccetti studied in \cite{50} and \cite{49} some issues and concepts of game event delivery. As a result, they proposed the ideas of relaxing some event orders and reliability constraints to improve game interactivity without affecting its consistency.

Palazzi et al. presented the Interactivity Loss Avoidance (ILA)\cite{118}\cite{119} technique, inspired from a proactive network congestion avoidance algorithm called Random Early Detection (RED). In ILA, the game semantic is used to detect obsolete events which are discarded with a probability depending on the game’s perceived responsiveness.

The system was adapted in \cite{117} for fast paced games such as FPS by dropping non-obsolete events when high degree of interactivity is required. Discarding non-obsolete events could generate inconsistencies, however the authors argue these artefacts are short lived and that the requirement for interactivity surpasses the need for consistency in some games. A more optimistic version of obsolescence based delivery using Timewarp algorithm was presented in \cite{48} along with simulations comparing it to the previously published event synchronisation schemes.

A fairness improvement over ILA was introduced by Ferretti et al. in \cite{51} and \cite{52} with the Fairness and Interactivity Loss Avoidance (FILA) event delivery. In this scheme, delays of event delivery are kept below a specified interactivity-threshold by dropping obsolete events while local lag scheme delays some events up to the interactivity-threshold to improve network event fairness.
2.8 Conclusion

The game industry has historically focused on the central server architecture and avoided requirements for inter-server/peers synchronisation. All recent delay sensitive games, such as FPS, RPG and most sport simulations, use some sort of latency compensation technique. Unfortunately, little documentation is available on their implementation.

Academia has recently initiated research in the gaming area, based on work on virtual environments which started a decade earlier. Studies on the effect of latency on players classify games in three categories: FPS (and some sport games such as car racing) as highly interactive, RPG as mildly interactive and RTS as the most latency tolerant.

While the negative impact of network conditions on users has been assessed for various games in multiple studies, there is no consistent analysis, across different research groups, on how network delays affect users. The only common conclusion is that latency generates “inconsistencies” of different nature. Although some publication define an inconsistency metric, their definition is not generic but restricted to specific aspects of the game state such as the position of entities. More importantly, the central notion of user experience is often reduced to response time or “interactivity” without providing any sort of measure other than player’s Round Trip Time (RTT). Such a rough metric neither take into account latency compensation techniques nor the fact that playability is perceptual. Additionally, RTT as sole metric does not include fairness as a component of user experience.

As a result of the lack of consistent and well defined terminology in the literature, this thesis proposes a framework in which the various aspects of network related user experience in games can be studied. Chapter 4, p.63 presents an analysis of game disturbances from network delays to impact on users through the delivery architecture and synchronisation scheme in use by games. This
classification is used by Chapter 5, p.88 to clearly define and assess the notions of playability and fairness and introduce fuzzy logic tool to model the perception of players.

A game's performance heavily depends on its communication architecture which dictates the flows of information across the network elements participating in the game session. Systems relying on a central server contain a potential bottleneck and a single point of failure while distributed architectures require some sort of synchronisation to maintain consistency amongst the authoritative game state images held on decision points. As shown in this literature review, there are many different ways to synchronise game states across authoritative decision points. Each technique has fundamental properties which affect the behaviour of the virtual environment. Conservative synchronisations maintain good state consistency while optimistic techniques provide better interactivity.

Chapter 6, p.119 challenges the binary choice between synchronisation providing consistency or interactivity. Concentrating on distributed servers architectures, this chapter shows that optimistic and conservative approaches are the extreme cases in a continuum of synchronisation possibilities. As we show that optimal synchronisation behaviour can be found in the trade off between optimistic and conservative, we argue that as in Distributed Discrete Event Simulations, there is no "one solution fits all" for networked games. There is only a range of available techniques with different properties to choose from.

Finally, server virtualization and other cloud computing technologies are becoming main stream and may enable fast game server(s) migration and re-localisation. However there has been very little published material on the optimisation of game servers topology. Using the analysis framework developed in Chapters 4 and 5, Chapter 7, p.143 formulates and resolve an integer programming problem aiming to select the location of game servers in order to optimise player experience.
Chapter 3

Terminology, Assumptions and Introductory concepts

Synopsis

As a preliminary to the main contribution of this thesis, this chapter describes and clarifies notions required by the rest of the dissertation.

Content:

- Terminology
- Assumptions of this thesis
- Introduction to game state and related concepts
3.1 Introduction

Previously, the literature review showed how the industry and academia discussed various concepts related to network games and synchronisation. Before presenting the contributions of this thesis, this chapter will introduce and clarify notions required by the rest of the dissertation.

The chapter is organised as follows. Section 3.2 disambiguates selected terms. The set of assumptions used throughout this thesis are presented and justified in Section 3.3. The anatomy of a networked game architecture is described in Section 3.4. It explains the roles of the different actors of a network game and introduces important notions related to the game state of online virtual environments.

3.2 Terminology

Many key technical words can have different meanings depending on the context they are being used. This section aims at clarifying the meaning of specific terms used in this dissertation.

**Playability, fairness and user experience**

In this thesis, the “playability”, “fairness” and the expression “user experience” are always used in the restricted context of networking i.e. playability and fairness issues induced by network imperfections. This is different from the generic game playability and fairness which also depend on the game engine, user interface, game rules, input/output devices and many more parameters.

**User, participant and bot**

The words “user”, “participant” and “bot” all refer to external participating actors from the game engine point of view. A user is a physical human being, a
bot is an artificial player and a participant may be either. Bots, which are simulated or emulated players, are not to be confused with Non Player Characters (NPCs) which are part of the game environment like monsters or terrain.

**Avatar and character**

In Fist Person Shooters (FPS) and Role Playing Games (RPGs), participants interact through “avatars” or “characters” which are the unique digital representation of the players’s presence in the virtual universe. Avatars are usually, but not always, anthropogenic. For example in online car racing games, avatars are likely to be vehicles instead of human-like. Participants in Real Time Strategy (RTS) games do not have unique representation of their presence in the game as they may control several entities simultaneously.

**Player**

In the video game culture, the word “player” is often used to refer either to a user or his in-game avatar, therefore blurring the line between participants and their digital representations. Consequently, in this thesis “player” refers either to a user/bot or to his avatar depending on the context.

**Game session**

In most FPS and RTS games, a “game session” (or simply a “session”) is a contest or competitive trial of finite time. In this context a game session is comparable to a sport match.

In games with a persistent and continuously evolving universe such as Massively Multiplayer Online Role Playing Games (MMORPGs), participants connect in individual sessions which start when they join and end when they leave the persistent world.
Decision point, terminal, server and node

A ‘decision point’ (DP) is a process running on a computer which takes authoritative decisions on the evolution of the Game State (GS), to be defined in the next section. A ‘terminal’ is a process running on a participant’s machine which displays the perceived representation of the game state to the user. In the client-server paradigm, terminals are being served by one or more decision points. Even though the word ‘server’ is often nowadays understood as a physical stand-alone machine running services on a network, it is used in this thesis here as a synonym to ‘decision point’. Note that in a peer to peer (P2P) architecture, participants’ machines are running both a terminal and a decision point process where in central architectures, the decision point is located on a single logical\(^1\) server.

In this thesis we will sometimes use the term ‘node’ to refer indiscriminately to either a decision point or a terminal. By extension, the ‘nodes of a game’ is the ensemble of all decision point(s) and terminal(s) involved in that game.

Game network infrastructure

For a given game session, we define the game network infrastructure as the combination of

- the network topology of all the nodes involved in the session. And,
- the synchronisation scheme (and its relevant parameters) used by the game terminal and decision point process to synchronise their game state.

Inconsistencies and violations as network induced disturbances

In this thesis, a “network induced disturbances” (or more simply, a disturbance) refers to anything directly or indirectly caused by the game network infrastructure which has a negative contribution to the game playability. Therefore delays, inconsistencies and violations (both these terms are defined in Chapter 4) are all disturbances to a networked game.

\(^1\)One single logical server may be distributed across a cluster or a grid of physical machine
**Network delay and distance**

Unless specified otherwise the word “*distance*” is used as a abbreviation for “*network distance*” and refers to the “*propagation delay*” involved in carrying the data between two points of the telecommunication network via the shortest available route. Therefore, such a distance is measured in *millisecond* (*ms*). As a generalisation, the “*network distance*” may also include (in accordance to this thesis’s assumptions explained in the next chapter) transmission, queueing and processing delays as long as they are considered constant for a short duration on each telecommunication link.

### 3.3 Assumptions

Throughout its contribution chapters, this thesis assumes a few approximations necessary to the in-depth and complex study of user experience in network games. This section formulates these assumptions and explains the reasons for their presence.

#### 3.3.1 Simulations requirements

The next paragraphs expose the elements required by our simulations and thought experiments.

**Game session stability**

In our simulations and examples we assume that the node topology of the game does not change during the playing session. In particular, terminals do not join or leave, decision points do not appear or disappear.

**Events sequentiality**

We consider actions to be strictly sequentially ordered. In other words, two actions may never happen at the exact same time. In reality, even in the
situation when multiple actions have the same timestamp, other parameters, such as IP addresses for example, can always be used to deterministically order them in a strict fashion.

**Numerical parameters**

While the framework developed here aims to be as generic as possible to include most types of online games, the study through examples and simulations requires numerical parameters. This is the case for synchronisation and game rule parameters, network topologies and fuzzy logic membership utility functions. Facing a choice, we selected values we trust to be reasonable and most people should not find unrealistic.

### 3.3.2 Network and processing imperfections

In the context of this thesis, we assume that the processing and telecommunication network imperfections affecting online video games can be aggregated in a constant latency between any two nodes.

**Short term jitter compensation**

The Internet best effort service does not guarantee packet delivery, delay, jitter or even in-order packet arrival. In practice however, within short periods of time, successive packets from the same source to the same destination usually follow the same route and the jitter stays in the order of a fraction of the average delay. In these conditions, the maximum delay of packets in the network can be assumed to be bounded by a value $X$ in ms. Packets timestamped at the source can be stored in a buffer of adequate size once received at the destination node and be distilled to the application once their timestamps is exactly $X$ms old. This enforces proper packet ordering and delivery to the application in such a way that they appear to arrive in timely fashion. Such buffers are commonly used to compensate jitter.
Long term jitter and topology changes

As a result of network fault or changes in congestion conditions for example, the topology of the telecommunication network may be altered. Such modifications happen on relatively long time frames and a game environment may search for new adequate synchronisation parameters on a regular intervals to adapt to these new topology conditions.

Packet loss

As this thesis focuses on the effect of latency, no packet loss is assumed throughout this dissertation. It can be argued that packet loss caused by data corruption during transmission is extremely low in current technology. Furthermore, packet drops from routers are a function of the network congestion and can be improved by devoting exclusive network resources. Techniques to cope with data loss are being studied in different telecommunication research fields and are outside the scope of this work.

Processing time

We assume the time taken by an application to process a data packet to be included in the total constant packet latency we consider. This is equivalent to say that the variation in processing time is negligible. The optimisation of playability and fairness when considering variable processing time and event queues was researched by Ferretti, Palazzi, Roccetti et al. in a series of publications and two PhD theses[47][116].

Studying the effects of all the different parameters simultaneously would be intractable within the scope of a thesis. Moreover, amongst all network imperfections, only propagation delays are truly bounded as information can not travel faster than the speed of light. When considering uncongested network and light CPU load, jitter, packet loss and processing time variation may be considered negligible.
For these reasons, this thesis focuses on studying the impact of latency on users’ experience. Under the above assumptions, all delays and latencies occurring in information transmission and processing between two network nodes may be aggregated into a single constant “network” latency.

3.4 Game State: the logical heart of networked games

The virtual world shared amongst all participants is digitally described by its Game State (GS). Each actor of a networked game maintains a local game state representing its own perception of the game universe. Differences amongst these local game states, caused by the network infrastructure, leads to divergent perceptions of the game environment and are the centre of the synchronisation issues in networked games. This section introduces concepts associated with the game states which are necessary for the investigation of online game networking issues in the following chapters.

3.4.1 Decision points, terminals and game state

The Game State is a set of variables describing the virtual environment. Such variables set includes, but is not limited to, the position and state of avatars and other in-game objects in the virtual world. Each terminal renders a projection of the game environment to its player based on game state updates received from its associated decision point(s). Decision points are processing locations (either in dedicated servers or client machines) that can authoritatively determine alterations in the virtual world.

Participants react according to their perception of the virtual world which may be incomplete and/or incorrect through their terminal. Each action has to be evaluated, based on the relevant subset of the game state, by an authoritative decision point which will determine the action consequence(s) on the game
state, as shown in Figure 3.1, p.58.

As an example, here is what happens when one player tries to shoot another avatar in an on-line First Person Shooter (FPS) game. Based on relevant game state information and action parameters such as the position information on both the target and the shooter, the decision point for this particular action will have to decide if the target is being hit or not and apply the consequences.

All actors of a networked game maintain a local game state which is more or less complete and more or less accurate. However, only decision points are authoritative in changing the referential game state. Any local decisions made on a terminal, using dead reckoning [121][1][12] or co-simulation (also called "client-side prediction")[12][6] for example, will only affect its own version of the game state and representation of the virtual world.

A decision point may be a single machine or a cluster/processing farm dedicated to the task. In some architectures, a decision point can also be hosted on the same machine as a terminal in which case it is non dedicated. Decision points may also be hierarchically organised, in which case a decision may be vetoed by another, higher ranked, decision point. Example of such situation include tactical shooter games in which the user’s machine can make shooting decision which are then validated by a central server[7]. Decision points may also not be authoritative on the entire game state. For example a given decision point may be authoritative only in a specific region on the virtual environment, on a given type of actions or within the scope of a specific event.

Prior discussing the resolution of action in more details, here is a summary of the section so far. The virtual environment of a networked game is described by its game state. All terminals and decision points maintain a local version of the game state. Terminals display a projection of the virtual environment for players to interact with the game. In-game actions are resolved by decision point(s) which also disseminate the updated game state.
3.4.2 Decision: altering the game state

The Game State is being altered by the actions of entities in the virtual environment. Each action leads to one (or more) decision(s) to be made to determine the consequences in terms of game state variation(s). The game state evolves in time through the sequential succession of decisions.

Let us define the Game State as the mathematical ensemble $GS$ of all the variables describing a given virtual environment, for any action $A$ processed by a decision point. Two important sub-ensembles can be described:

- $GSr(A)$, the relevant game state, is the ensemble of all the variables required to make the decision about the action $A$

- $GSm(A)$, the modified game state, is the ensemble of all the variables modified by the outcome of the action $A$

The set of relevant variables $GSr(A)$ depends only on the action $A$ itself. The set of modified variable $GSm(A)$ depends on both action $A$ and the state of the variables used in the decision making process $GSr(A)$. For example if $A$ is the action of Alystan attempting to shoot Pazoo, $GSr(A)$ is likely to include the position of both players and the direction of the projectile. Pazoo’s health level will only be part of $GSm(A)$ if the shooting succeed.

Figure 3.1, p.58 illustrates how the decision process produces a modified game state from the relevant previous state and the action parameters.

Furthermore, as shown by Figure 3.2, p.59, for each action in the game, any variable $v$ in the game state $GS$ belong to one of these four sub-ensembles:

**Irrelevant variables** if $v \notin GSr$ and $v \notin GSm$: $v$ is totally independent from action $A$.

**“In” only (or “informative”) variables** if $v \in GSr$ and $v \notin GSm$: $v$ influences the outcome of the action but is not altered by it.
Figure 3.1: Decision point, action and game state. The decision about the outcome of action A requires the action parameter, the input and input/output game state variables. The consequences of the action is modifying the input/output and the output games state variables.

"In/Out" variables if $v \in GSm$ and $v \in GSr$: $v$ influences the outcome of the action and is altered by it.

"Out" only variables if $v \in GSm$ and $v \notin GSr$: $v$ is altered by the outcome of the action but does not participate in the decision process.

As an example of action in a FPS game, we consider Alystan attempting to shoot at Pazoo. The irrelevant variables include the colour of the sky for example. One In (informative) variable is the position of Pazoo, which is required to calculate if the shot is successful or not, but will not modified by the outcome of the action. An In/Out variable could be Pazoo’s health status which may deteriorate, possibly to the “death” of the avatar, weather the bullet hits or not. A variable storing the name of Alystan’s last victim would be an Out variable as its new value does not depends in any ways from its previous value. Finally, the current example requires the projectile trajectory an additional action parameter to resolve the shooting action.
Correlated actions

Let us define the function $\text{Exec}$ which executes any action $A$ and transforms the game state $GS$ accordingly:

$$GS[n + 1] = \text{Exec}(A, GS[n])$$

Palazzi et al. define in [119] two actions to be correlated if the final game state depends on their execution order. Hence actions $A$ and $B$ are correlated if:
The cause for two actions $A$ and $B$ to be correlated is that the results of one is used as a parameter by the other. In logical terms:

$$GSm(A) \cap GSr(B) \neq \emptyset$$

or

$$GSm(B) \cap GSr(A) \neq \emptyset$$

Out of order execution of correlated actions leads to paradoxes and is discussed in Section 4.4.4, p.85.

The resolution of actions in the decision points requires the knowledge of multiple game state variables either to be used as input, output or both. The execution of actions out of order may lead to an incorrect decision if the output variables of one action is to be used as the input of the other.

### 3.4.3 Game state vector and trajectory

In the final part of this section, we introduce the concepts of Game State vector and trajectory which are important to simulation results presented later in this dissertation.

The game state is the collection of all the variables describing the game’s virtual world. Let us imagine a vector space with one dimension per game state parameter which we call $GS_{space}$. The game state at any given time can be represented as a vector in that $GS_{space}$.

During the unfolding of a game session, the game state evolves and therefore travels in the $GS_{space}$. Its successive positions in this space from the beginning to the end of a given played session is the *game state trajectory* of that session.
The game state trajectory of a session depends on:

- network infrastructure of the game (including synchronisation scheme and topology)
- Initial game state conditions
- the actions of participants
- rules of the game (which can include some randomness)

Subsets of the game state can also be observed. In particular, it is often interesting to look at the state parameters related to one particular avatar. The variations in a game session of the state parameters associated with a given player is referred to as the player state trajectory.

### 3.4.4 Game state: summary

A game virtual environment is described by its game state and can be represented as a vector in the game state space. As entities in the game take actions, decisions about the result of these actions modify the game state vector in time producing a trajectory. This trajectory depends on the network infrastructure of the game, initial conditions, actions of players and game mechanics.

### 3.5 Conclusion

This preliminary chapter laid down foundations required by the following contributing chapters. We formulated and justified the approximations underlying this thesis and clarified some important terminology. Finally, notions related to game state were introduced.

Because the thesis focuses on studying the impact of latency, its main assumption is that the processing and telecommunication network imperfections
affecting online video games can be aggregated in a constant delay between any two nodes.

The next chapter is dedicated to show how this delay is the source of a cascading chain of disturbances which ultimately impacts the experience of users.
Chapter 4

Networking aspects of user experience

Synopsis

This chapter introduces the framework used to analyse the network infrastructures supporting online games. We show how propagation delays modulated by the synchronisation scheme generate inconsistencies which impact the game through violations of the virtual environment ideal laws.

Content:

- Inconsistencies created by the network infrastructure
- Violation of the virtual world’s laws impact user experience


4.1 Introduction

Previously, the literature review showed how the industry and academia discussed various concepts related to network games and synchronisation. However, some terms such as inconsistencies, playability or fairness have no precise and commonly accepted definition. Consequently, different research groups usually discuss these terms in specific ways that suit a particular situation or model. This chapter aims at refocusing some of these concepts and sets up tools to study the influence of network infrastructure on players.

To this aim, we propose a hierarchy of game disturbances to help analysing the impact of network architectures and classes of synchronisation scheme on the players. In this hierarchy, the propagation delays in the network, modulated by a synchronisation scheme, create measurable inconsistencies. In turn these inconsistencies generate violations of the ideal laws ruling the game virtual environment. These violations are more difficult to quantify and lead to degradation of the game experience of participants.

Within this framework, which is used throughout this dissertation, analysing a networked game is a three steps process. First, measurable inconsistencies are derived from the network infrastructure and the synchronisation scheme used by the game. Secondly, with the knowledge of the game rules and laws, the possible violations generated by the established inconsistencies are determined. While these first two steps are covered in the next sections of the current chapter, the last step which consists in studying the impact of the identified violations on the game experience of users, is a larger body of work which relate to the notions of playability and fairness. Therefore, this last step is treated in its own dedicated Chapter 5 “Playability and fairness”.

This chapter is organised as follows. Section 4.2 presents our hierarchy of disturbances. It introduces the core notions of inconsistencies and violations within our framework. Section 4.3 is dedicated to the first step of our analysis
process and will translate the classic network game infrastructures into their inconsistencies. Central server, peer to peer with different type of synchronisations, and distributed servers topologies will be considered. Next, Section 4.4, p.82 discusses the notion of violations and shows how inconsistencies generate them. Finally, Section 5 will conclude this chapter.

4.2 Model of game disturbances

As seen in Chapter 2, p.11, some research groups have categorised disturbances [139][152][68][115][53]. However, these classifications were developed to match specific setups and have not been shown to be generalisable. In order to understand how the network game infrastructure impacts the experience of users, this section introduces a generic model of game disturbances.

4.2.1 From network delays to user impact

In an ideal network game with all participants in the same physical location, ie. with no appreciable network propagation delays and infinite processing power/transmission capacity, the virtual environment can perfectly obey the ideal laws of the virtual world (assuming the software is bug free). In particular, all players perceive the same game state at the same time, actions are executed in time and order and the physics of the virtual world are respected.

Geographical distances amongst terminals and decision point(s) introduce unavoidable propagation delays in the telecommunication network which prevent decision points and terminals from receiving simultaneous and accurate perceptions of the game state. The end result for the players is a perceptual loss in playability and/or fairness, disturbing the game play.

Strictly speaking, because the network topology between sessions of the same game may vary significantly, one can only analyse the playability and fairness of a given game session (as opposed to a game in general). In this chapter, the
network topology is assumed to be a given parameter.

The hierarchical organisation of disturbances we use to explain how the topology finally impacts the users is illustrated by Figure 4.1 p. In this model, inconsistencies are the results of network delays after modulation by the synchronisation scheme used by the game. In turn, these inconsistencies generate violations of the ideal laws of the virtual environment. It is the perception of these violations by the players which reduces the playability of a game.

Inconsistencies and violations are the key concepts of our hierarchical organisation. These two notions enable the full deconstruction of the mechanisms leading from the network infrastructure of a game to the final impacts on its users.
4.2.2 Inconsistencies and violations

The synchronisation scheme of a game defines the information flows and processes updating the game states on the decision points and terminals. For any information flow, the amount of inconsistency generated by the sender on the receiver of the flow is determined by the delay in the network path between them. Depending on the architecture, inconsistencies can be present on decision points and/or terminals.

In our model, the network topology and the synchronisation scheme only influence a network game through inconsistencies. Consequently, the collection of all inconsistencies captures all the possible negative influences a game network infrastructure can have.

Within our framework, inconsistencies have the following properties:

- an inconsistency is measurable in unit of time. This dissertation will use milliseconds.
- the collection of all inconsistencies on all nodes captures all disturbances influencing users’ experiences introduced by the network infrastructure (as the combination of the network topology and the synchronisation scheme).

Ultimately, inconsistencies lead to variations, temporary or not, between the perceived game state of different game nodes. We will refer to such variations as divergences. Depending on the context, these divergences can have consequences on information displayed to the players by their game terminal or on the decision process of decision points. It is remarkable that the presence of an inconsistency between two nodes may not be represented at all time by a divergence between their game state.

Unlike the natural laws and principles driving the real world, simulated ideal laws can be (and are) violated in network games. Inconsistencies, as described
above, always induce violations of the virtual world one way or another. For example the presence of some terminal inconsistency may generate a non-zero response time which is a digression of the ideal laws of the virtual world where actions should not be delayed. Another example is the creation of a paradoxical situation such as the one described in Section 2.6.1, p.37 due to inconsistency between two decision points.

Violations can take different forms and are not always easy to evaluate. A perceived lag would be measured in ms. However causality violation or dead reckoning error are a lot harder to measure because of their probabilistic nature. For example, causality violation could be estimated in expected number of violations per unit of time. The evaluation of probabilistic violations is less objective than the measure of inconsistencies because it requires knowledge about the level of game interactivity to estimate the probability of triggering correlated actions. Such information is very difficult to obtain as it depends on multiple parameters including density and behaviour of players, game type, game rules, areas of interests and more.

Section 4.4 is dedicated to violations and will explain how different types of inconsistencies create them. The study of how they impact players will be the focus of Chapter 5.

4.2.3 Summary

In network games, the imperfections of the network infrastructure generate inconsistencies across the game state perception of decision points and terminals that can be precisely expressed in time units. These inconsistencies violate the ideal laws of the game. Violations are more difficult to quantify and it is their perception that impact the experience of users.
4.3 Inconsistencies

This section endeavours to decompose the classic synchronisation scheme / topology pairs into inconsistencies. We believe that our framework is generic and that similar analysis can be done for any specific game network infrastructure. To improve clarity, each inconsistency will be shortly illustrated with a short concrete example of an associated violation. Violations will be the subject of Section 4.4.

First, central server architectures will be examined and they will show to generate inconsistencies on game terminals. Next, we will analyse peer to peer topologies with three main family of possible synchronisation schemes. It will be shown that in these situations various type of inconsistencies appear in decision points. Finally hybrid architectures such as distributed servers will be considered. This analysis will show how terminal and decision points inconsistencies interacts with one another.

4.3.1 Inconsistencies in the central server architecture

In a central server architecture, the consistency of the referential game state is guaranteed by the presence of one unique decision point for all players and all actions. By definition there can not be any decision point inconsistency, it is therefore a good platform to explore the inconsistency created by the terminal-decision point delays.

The propagation time from a terminal to its associated decision point (the upstream delay, Figure 4.2, p.70) introduces a delay between the issuance of an action request by a player and the moment this request reaches the decision point. Let us call this delay the request age of the action. In a central server architecture, this delay is equal to the propagation time from the terminal to the central decision point.
Figure 4.2: Terminal inconsistencies in the central server architecture. Update age is the time it takes for the server update to reach the client and the request age is the time it takes for the server to receive the player’s action request. The action age is the sum of the other two inconsistencies. Note that this sum is also referred to as Round Trip Time (RTT) or response time in the literature, however in our framework, response time is classified as a violation and not an inconsistency.
Networking aspects of user experience

\[ RequestAge(P) = D(P, S) \]

Similarly, the propagation time from a decision point to one of its associated terminals (the downstream delay, Figure 4.2, p.70) generates a delay between the issuance of a state update by the server and the moment the terminal receive this new information and displays it to the participant. Let us call this latency the game state *update age*. Therefore, when an update reaches a terminal, the state information it contains are already outdated to some degree because the referential game state may have varied while the update packet was on the way.

\[ UpdateAge(P) = D(S, P) \]

Let us define *action age* as the total amount of inconsistency on a terminal. In the current scenario, action age is the sum of the request and the update ages. From the point of view of the player, the amount of time taken to see the result of the action he sent is in Figure 4.2, p.70 \( T_D - T_B \). The amount of time is equal to the total amount of his/her terminal inconsistency. This non-zero response time is in fact a violation of the universe’s laws (since actions should not be delayed) caused by the terminal being inconsistent with the referential game state.

\[ ActionAge(P) = RequestAge(P) - UpdateAge(P) = D(P, S) + D(S, P) \]

In the central architecture, request and update ages are direct products of the network distances between the terminal and its associated decision point. Therefore participants, such as the ones in Figure 4.3, p.72, with a signifi-
4.3.2 Inconsistencies in the peer to peer model

In a pure peer to peer architecture, each peer acts simultaneously as a terminal and a decision point. The local terminal is naturally served by the local decision point, therefore there is no terminal-decision point delay involved which makes it a good architecture to study inconsistencies generated by inter-decision point delay and synchronisation.

The propagation delays between the peers make it impossible for all decision points to maintain an identical view of the current game state at any given time. None of the peers hold the referential game state as in the central server architecture. This results in temporary game state discrepancies amongst the peers. There are two philosophies to deal with these discrepancies:

Correct synchronisation where peers exchange event request information. For example “Alystan attempt to shoots at Pazoo at time t1”. The consequences of the action, i.e. “the shot was successful” or “the shot was

1 unless the game stays static for long enough, which is not a situation of our interest
not successful", are then computed on each decision point independently.

**Loose synchronisation** where peers exchange *event consequences*. For example: “Alystan has eliminated Pazoo”. The consequences of the action are being pre-calculated on the peer doing the action and then the results are distributed to the other nodes.

Each method will be analysed separately.

**Correct synchronisation**

In correct synchronisation, decision points maintain correctness as defined by Mauve [99]. To that aim, they are time synchronised and exchange their respective players' action as *event requests* which are timestamped. Then, each decision point can enforce execution of events in order even though such an operation may require the decision point to roll back in time to reorder the execution. The correct ordering of events ensures causality is respected.

Since a decision point relays its player's action as *event requests* to other decision points, let us define the *event age* created by one decision point (source) on another (destination) as the difference between the time the event request has been received and the moment it is (or was) supposed to take place. *Event age* is the amount of decision point inconsistency generated by the event source on the receiver.

Any given peer suffers from a collection of event ages caused by the propagation delays from every other peer to itself. Because of the variability of the network distances, the collection of event ages from the other peers may not be uniform.

**Correct optimistic synchronisation**

In optimistic synchronisation, the event age is simply not dealt with. Peers are allowed to keep making decisions based on their perception of the game which may be inconsistent, dealing with the consequences later.
The different decision point event ages on a given peer may cause it to evaluate some actions out of order. In the case of non-correlated actions, the order of execution does not change the outcome. However out of order execution of correlated actions\cite{119} leads to incorrect decisions. In this situation different peers may determine different decisions for the same event; a situation also referred as \textit{paradoxical}\footnote{cf. 2.6.1, p.37}.

“Alystan attempts to shoot at Schrodinger’s cat” and “Pazoo attempts to move X metres in direction Y” is an example of two non-correlated actions. The result of either action is independent of the outcome of the other.

On the other hand, re-using the example from Section 2.6.1, “Alystan attempts to shoot at Schrodinger’s cat” and “Pazoo attempts to shoot at Alystan” are two correlated actions. If Alystan’s action is executed first, the cat end up dead, otherwise it stays alive.

To maintain correctness, causality needs to be restored. To that aim, one or more decision points might have to rollback in time\cite{20}, creating an unnatural modification of the virtual reality to heal the game state. Such rollbacks are also referred to in the literature as “Timewarp”\cite{99} in both areas of Distributed Event Simulations (DES)\cite{56} and networked games/virtual environments research. Cronin’s trailing state synchronisation\cite{37}\cite{35}\cite{36} implemented in a distributed Quake is a variation of timewarp. In correct optimistic synchronisation, the probability of a rollback is a form of violation created on a decision point by their event age and affecting all the terminals it serves.

It is worthwhile noticing that peer to peer architectures using pure optimistic synchronisation do not suffer any terminal inconsistencies. This is not the case when using conservative synchronisation as discussed in Section 4.3.2 below.

\textbf{Conservative synchronisation}

If a conservative synchronisation mechanism such as \textit{local lag} \cite{99} is used,
**Figure 4.4:** Event age in optimistic and conservative synchronisation. Receiving an event update on a decision point after it was scheduled to occur generates an inconsistency called *event age* on this decision point. Event age can be neutralised if the event execution is artificially delayed by a local lag.

Consistency between decision points is maintained by adding artificial delays to actions that attempt to modify the game state. This is done at the cost of reduced responsiveness and interactivity.

When using local lag, the would be event age generated by a particular peer on the other participants is neutralised with an artificial latency (local lag) aimed at delaying events. The artificial local lag allows events to reach the other peers in time for their execution. Figure 4.4 compares two actions: *Action1* issued on *PeerA* is immediately effective (no local lag) and generates an event age on *PeerB*. On the other hand, *PeerA* adds a local lag on *Action2*, making it effective only when it reaches *PeerB* and does not generate any event age on *PeerB*.

If all peers apply enough local lag, the event ages of all participants can be neutralised. Local lag enforces consistency between peers’ game states, however the added artificial lag delays the moment actions from participants become
effective: it reduces the event ages of other peers but at the cost of extra delay from the terminals to the decision point.

In this scenario, inter decision point inconsistencies are null but the terminal request age is artificially increased by the amount of local lag used. From a game state and decision perspective, conservative peers emulate the consistency of a central decision point. As in the central server scenario, the terminal inconsistencies generate a response time and related violations.

Hybrid or partially conservative synchronisation is also possible. This particular technique is explored in details in Chapter 6, p.119.

In all types of correct synchronisation, the game state stored on different decision points may not be exactly identical, but they all converge onto a single game state trajectory. At the end of a game session, all decision point would agree on the exact development of the session.

**Loose synchronisations**

We categorise as "loose synchronisation" all synchronisation schemes not maintaining correctness. Such synchronisation typically do not enforce causality leading to causality violations and paradoxes. In a way, loose synchronisation can be even more optimistic than correct optimistic synchronisation.

Instead of exchanging action orders, peers using the most simple form of loose synchronisation exchange game state updates to force the convergence of their game state. In other words, loose synchronisation keeps the game state divergences under control by enforcing the game state modifications from each peer on all the others. Instead of event age, decision points suffers from a different type of inconsistency we name *decision age*, which in some way resembles the update age of a terminal.

Since decision points send decisions about their players' actions to other decision points, let us define the *decision age* induced by one decision point to
another as the difference between the time a decision has been made and the moment it has been received.

Starting with an identical initial game state, the views of different peers could rapidly and dramatically diverge if they were exchanging action orders and executing them in order of arrival (which could be different amongst the peers). For example, let us imagine two players taking action simultaneously in a loosely synchronised P2P game: Alystan shoots and kills Pazoo (transmitted game state alteration is "set Pazoo health to 0") at the same time as Pazoo drinks a health potion (transmitted game state alteration is "increase Pazoo health by 50"). After exchange and application of the game state alterations, Pazoo’s avatar is dead on his own machine and has been resurrected on Alystan’s peer. It is important to note the differences with the Schrödinger's cat scenario presented in 2.6.1, p.37; as if servers $S_1$ and $S_2$ were loosely synchronised, both $P_2$ and the cat would be dead on all decision points and no paradox would have appeared.

Virtual environments based on loose synchronisation prevents such situations from happening use additional logic such as single entity ownership restriction used by many DVE[66][142]. As presented in Chapter 2, some games like ET Pro[7] and military simulations directly synchronise states with action consequences instead of action orders. Diot and Gauthier's distributed game, “MiniMaze” [43][63] also synchronises peers this way. This approach is only possible amongst trusted hosts.

Decision age is a different type of decision point inconsistency than event age. Decision points that are loosely synchronised do not travel on the same game state trajectory as correctly synchronised ones. After synchronisation, an avatar may be dead and different decision points may attribute the kill to different players. This situation may lead to incorrect or divergent scoring for example.

The violation introduced by the decision age inconsistency is the probability of violating causality in the game. Section 4.4.4, p.86 explains why some types
Networking aspects of user experience

of game state parameters simply cannot be loosely synchronised.

**Loose synchronisation in the central server**

Surprisingly, it is possible for a central server to become inconsistent with itself. This situation can happen in games using the type of latency compensation technique as documented in [12] and [6]. This technique attaches timestamps on game state updates and executes each user action back in time, in the context of the game state as perceived by the player’s terminal when the action was sent.

Because the decision point rolls back in time to recreate the context in which the player generated an action, decisions on actions are made as different logical processes using slightly unsynchronised game states. Consequently, exactly like using the result of an action computed by a different decision point, applying the effects of an action processed back in time on the current game state may violate causality. This technique has the same effect as using a loose synchronisation and therefore could break correctness in the game, potentially bringing the type of paradoxes discussed in Section 4.4.4, p.86.

**4.3.3 Inconsistencies in the distributed server model**

Distributed or mirrored server architectures, such as the ones described in [9], [98] and [37], may exhibit inconsistencies of both central and peer to peer architectures.

All participants are equally affected by the decision point inconsistencies of the server they are connected to. Players suffer from both their terminal and their server inconsistencies.

In conservative synchronisation, the distributed decision points perform as consistently as a central server but the required artificial delay added on each decision point affects all its connected terminals. When using the local lag scheme
for example, Figure 4.5, p.79 shows how the local lag on a decision point adds to the effective terminal inconsistency of all its connected participants.

In any case, the inconsistencies impacting a player are fully described by its terminal inconsistency and the collection of its decision point inconsistencies.

By definition, a terminal is not authoritative on the referential game state. Consequently its delay towards the decision point(s) and its synchronisation parameters do not affect inconsistencies of the other terminals or decision points. For example in a central server game topology such as the one presented in Figure 4.3, p.72 terminal C is more inconsistent with the referential game state of server S than the other terminals because it is further away in terms of network distance. However the inconsistency on terminal C only affect the player using it and does not affect anyone else. In other words, terminal inconsistencies are self-contained and do not propagate.
4.3.4 Aggregating of Update, Request ages and local lag

The two inconsistencies present on a terminal, update age and request age, and the local lag introduced by the decision point, if any, can safely be aggregated in the action age to represent the whole terminal inconsistency.

Let us imagine two terminals $T_1$ and $T_2$ connecting to the same server with the same amount round trip time $RTT$ but presenting asymmetric upstream and downstream latencies as shown in Figure 4.6 p. 80

- Terminal $T_1$ with its $D(T_1, S)$ upstream and $D(S, T_1)$ downstream delays.
- Terminal $T_2$ with its $D(T_2, S)$ upstream and $D(S, T_2)$ downstream delays.

Their respective upstream and downstream delays are set to be different but their total round trip time is equal:

$$D(T_1, S) < D(T_2, S)$$
$$D(S, T_1) > D(S, T_2)$$

$$D(T_1, S) + D(S, T_1) = D(T_2, S) + D(S, T_2) = RTT$$

Since these two terminals are connected to the same server, they suffer the same amount of local lag $LL$ if there is any. Therefore, their respective update
Networking aspects of user experience

ages UA and request age RA are equal to the following:

- $UA_1 = D(S, T_1)$ and $RA_1 = D(T_1, S) + LL$.
- $UA_2 = D(S, T_2)$ and $RA_2 = D(T_2, S) + LL$.

The amount of action age AA on each terminal is similar and equals to:

$$AA_{T_1} = D(T_1, S) + D(S, T_1) + LL$$
$$= RTT + LL$$
$$= D(T_2, S) + D(S, T_2) + LL$$
$$= AA_{T_2}$$

From its perspective, the decision point sends to both terminals the same game state update $Update_A$ at $t_0$. $T_1$ receives that update at time $t_1 = t_0 + D(S, T_1)$ and react to it immediately sending an order to the server. $T_2$ receives that update at a different time $t_2 = t_0 + D(S, T_2)$ and also react to it immediately. Both $T_1$ and $T_2$’s orders are received by the server at time $t_3 = t_0 + RTT$ and processed. The results of both actions are contained in server $Update_B$ sent to both terminals. The decision point only “see” the round trip time.

From the terminals point of view, they both perceive the update $Update_A$ and react to it sending an order back to the server. The results of their actions is perceived $RTT + LL$ ms later. Technically, these terminals do not receive the update simultaneously. However this does affects neither the outcome of their reactions nor the decision of the server.

4.3.5 Inconsistencies: summary

Regardless of the game architecture and synchronisation, delays involved in the transmission of information generate inconsistencies. We showed how our model can generically determine the amount of inconsistencies and its type for
Networking aspects of user experience

each game node in central server, peer to peer or hybrid architecture, using optimistic or conservative synchronisation.

4.4 Violations and user impact

In the previous Sections, we have presented the various inconsistencies arising in different game architecture and synchronisation combinations. We will now see how different violations caused by inconsistencies affect the game experience of participants.

4.4.1 Inconsistencies cause violations

As discussed in Section 4.2.2, p.67, violations can take different forms and are not always easy to evaluate. If no action is taken to alter, transform or trade an inconsistency, it will lead to one or more violations. We call the unaltered violation(s) generated by a particular inconsistency its natural violation(s). For example, response time is a natural violation generated by the terminal inconsistency. This section will present specific techniques, such as co-simulation, which allow violation alteration. How to use these techniques to improve the experience of users is the subject of Section 5.4, p.99.

Both terminals and decision points may create game violations as a result of inconsistencies. On a decision point, violations affect all its connected players while on a terminal, violations only affect its user.

4.4.2 Action evaluation distance

Ideally, the game state presented by the terminal which is used by the player to determine his course of action should be the same as the game state used by the decision point to evaluate that player’s action. In practice, there is often a difference between the game state displayed just before a player takes an action and the game state used by the decision point to compute the outcome of the
action. The difference between these two game states may alter the decision making process of the action which may not have the expected outcome the player intended. The difference between the presented and the decision game state in the game state space is a game state distance. This distance is a violation we will refer to as the action evaluation distance.

Any disturbance increasing the distance between the displayed game state vector and the game state vector used by the decision point to compute the outcome of the action increases the action evaluation distance. The total terminal inconsistency (action age) which includes request age, update age and local lag (if applicable) naturally contribute to the action evaluation distance. Event age on a decision point means that the game state held by the decision point is not up-to-date with the decisions from other decision points. Therefore the game state update sent to the terminal it serves is also not up-to-date meaning that event age is also a contributor to action evaluation distance.

It is worthwhile noticing that the decision age on a decision point using a simple loose synchronisation scheme (as described in 4.3.2, p.76) does not affect the action evaluation distance because a decision made on a given decision point is final and will not be rolled back even if it leads to a violation.

The game state evaluation distance of a player’s action can be modified by dead reckoning [97] [121]. This technique can be used to reduce the game state difference created by the action age in some of the game state parameters. Dead reckoning uses knowledge of some physical laws of the universe and previous values of a given parameter to try to predict its future values. In the case of avatar’s position, knowing the speed and the direction of movement of a avatar, its position can be extrapolated. In this example, error on the extrapolation is a function of the unknown acceleration of the avatar (such as change in direction) and the amount of lookahead done.

Dead reckoning attempts to reduce the action evaluation distance. However, the extrapolation may lead to other types of violations. In particular, a terminal
may display events which had never happened, such as an avatar running off a cliff while the player actually stopped on the edge.

4.4.3 Response time

The response time was previously introduced in Chapter 2 and further mentioned in Section 4.3.1, it is the delay between the time of the issuance of an action order by a player and the display of the action results on his terminal. Therefore it is a perceptual violation of the time law of the game (game state time error) if the consequences of avatar’s actions are supposed to be immediate. If not altered, the response time is equal to the total amount of terminal inconsistency: update age, request age and local lag. Players tend to be particularly sensitive to their perceived response time and most online games implement some form of co-simulation described below, to hide it.

Co-simulation or client side prediction [12][6] attempts to anticipate the response of the decision point to an action order for immediate rendering. It is equivalent to say that the terminal becomes a non-authoritative local decision point.

However there is a probability for the prediction not to match the decision of the authoritative decision point. In which case the terminal would display an erroneous state to the player. Moreover, correction of the local game state by a server update may create a local rollback on the terminal. In other words, co-simulation trades response time for display error and probability of creating a causality violation (which would be confined to the scope of the terminal).

If Alystan and Pazoo are playing a game using terminal co-simulation for avatar’s position. An example of co-simulation error would be Pazoo cast an immobilising spell on Alystan who is running toward a flag. Alystan’s terminal is co-simulating his own movements and shows him reaching the flag. However, once the game state update from the server containing the result of Pazoo’s spell is received by Alystan’s terminal, Alystan perceives a rollback as his avatar
is pulled back from the flag. The phenomenon is called “rubber banding” by gamers.

4.4.4 Causality related violations

Paradoxes and rollbacks are the two causality related violations. In Section 2.6.1, p.36, we defined a paradox as a decision made by an inconsistent decision point which is *incompatible* with the decision it would have made if it was consistent.

Rollbacks in co-simulation

Healing a paradox requires a rollback, a jump back in virtual time to correct the offending decision. When the prediction from co-simulation on a terminal does not match the outcome of the decision point as described above, the rollback happens when the game state update from the decision point is displayed.

Rollbacks and event age

On a correctly synchronised decision point, a paradox occurs when executing two correlated actions out of order because of the event age as explained in 3.4.2, p.57. Healing the paradox requires a rollback[20] or Timewarp[99] re-executing events in order. This operation creates a violation as an unnatural modification of the virtual reality.

A rollback on a decision point affects all its connected players. Also, a decision point may invalidate decisions previously made about actions from its own connected players because the game state previously used to decide them was not up to date. The probability of a rollback being required is exclusively caused by the presence of event age on the decision point. Frequency, amplitude and time depth of rollbacks will negatively affect the experience of participants and their enjoyment of the game.
The situation described in Section 2.6.1, p.37 is a good example of a paradox happening on a correctly synchronised server. On the server S1, Alystan eliminates Pazoo, but while the information was on the wire, server S2 allowed Pazoo to kill “Schrodinger’s cat”. To heal its game state, server S2 need to re-execute all actions in order, resurrecting Schrodinger’s cat.

**Causality violation and decision age**

In loosely synchronised systems, there is no absolute referential game state as all decision points share the authority. By definition, a decision made on a given decision point never seem to breach causality locally. However the subsequent game state update may violate causality on other decision points. The violation may look like a straight violation of physical laws, a paradox or a rollback as the following examples show.

Let us imagine that Pazoo is rushing behind a wall while being shot at by Alystan. Even if, time-wise on his terminal, Pazoo ran into cover before the shot was fired, if Alystan’s terminal decides the shot was a hit, Pazoo will see his avatar getting shot while already behind the wall. This situation may be perceived from Pazoo perspective as a violation of the physical laws (if we suppose that avatars are not supposed to get shot through walls) or a rollback if his avatar has been pulled back from behind the wall to be shot. But from Alystan’s point of view it all makes sense, Pazoo’s avatar was shot before he could reach his cover.

However, some types of causality violations may simply not be acceptable. For example, in a loosely synchronised peer to peer game, if two peers Alystan and Pazoo pick the same unique object α from the ground. Alystan’s machine advertises to all peers “Item α is in Alystan’s inventory” and Pazoo’s machine advertises “Item α is in Pazoo’s inventory”. In effect, the unique item becomes paradoxically duplicated. Therefore the naive loose synchronisation fails. A corrective negotiation could be implemented to check for object uniqueness and enforce the oldest decision to maintain the object in only one of the two.
players' inventory. Such a correction would appear as a rollback from the the player losing the object.

4.5 Conclusion

This chapter set up the tools to study the influence of network infrastructures on players. To that aim, we introduced a generic hierarchical organisation of network disturbances. This framework is capable of de-constructing the chain of events leading from the game network infrastructure down to the final impact on users.

Geographical distance amongst the network elements of a networked game create propagation delays which generate game state divergences. These divergences amongst game state of terminals and decision points are at the heart of the networked game synchronisation issues.

In our proposed model, inconsistencies measurable in time unit, capture all disturbances created by the the game network infrastructure. These inconsistencies, in turn, create violations of the game's ideal laws affecting coherence of the game which are more difficult to measure.

Parts of the contributions of this chapter were published along with results from other sections of this thesis in the November 2006 issue of "Communications of the ACM"[22].

Next, Chapter 5 derives definitions of playability and fairness from the concepts of inconsistencies and violations to develop an understanding of the network infrastructure’s impact on participants of a game. Later, Chapter 6 will propose novel synchronisation schemes balancing inconsistencies in order to maximise participants playability.
Chapter 5

Playability and fairness

Synopsis

This chapter explores the causal relationships between violations and users’ experience in online multiplayer games.

Content:

- *Introduction to fuzzy logic tools to evaluate the impact of network disturbances*
- *Definition of network playability and introduction to disturbance trading*
- *Exploration of causes of network fairness leading to its theoretical and experimental evaluation*
5.1 Introduction

As it is the case for the term "inconsistency", playability and fairness have no precise and commonly accepted definition in the literature. Different research groups usually discuss these terms in specific ways that suit a particular situation or model.

The previous chapter has presented the chain of network related disturbances leading to users’ discomfort. It exposed how delays across the network generate, regardless of the type of server architecture and synchronisation, violations of the game environment’s ideal laws perturbing the game play. However these different violations do not always have the same impact on players leading to the possibility of optimising user experience via the tuning of the game network architecture and synchronisation.

To formulate suitable optimisation objectives to improve user enjoyment it is necessary to link network infrastructure parameters to user experience appropriately. This Chapter connects network related playability and fairness to the network infrastructure parameters through the notion of violations and inconsistency introduced previously.

This chapter is organised as follows. Section 5.2 defines the notion of playability in the context of online games. In Section 5.3, we propose to use utility functions manipulable by standard fuzzy logic tools to estimate the final impact on the users’ experience. Section 5.4 uses this model to show how inconsistency and violation trading can improve game playability of players. Finally, Section 5.5 proposes a theoretical estimation of fairness and compare it to an experimental measure of fairness.
5.2 Playability

This section steps back a little to search the meaning of playability in the larger picture before linking it to measurable network disturbances discussed in the Chapter 4.

5.2.1 Defining playability

The online Oxford English Dictionary reports its definition of “playability” in the following terms:\footnote{\textsuperscript{1} as published in the English Oxford Dictionary, Second edition 1989}

playable, a
1. Given to play, playful, sportive. \textit{Obs.}\textsuperscript{2}
2. Capable of being played: in various senses.

\textbf{b.} Of a cricket or football ground or the like: Admitting of or fit for playing on.

Hence \textit{playability}, the quality of being playable.

The meaning of playable in the cricket or football ground context can be extended to network in online games. If the “ground” is interpreted as the facilities used to play the game, then the network infrastructure supporting a given online games is the “ground” of that game. In this context, \textit{network playability} can be lexically defined as the quality of the online game infrastructure of being playable. Network playability is a subset of the generic game playability which also include (but is not limited to) the quality of the human-computer interface and the artistic quality of the story, graphics and sounds.

5.2.2 Network playability

In the online game literature, the notion of network playability is often tied to responsiveness or interactivity. However, as shown in Section 4.3.4, p.80

\textsuperscript{2}The first meaning is indicated as obsolete and is irrelevant to our context
response time is not always the only network related inconsistency affecting participants. In this thesis, we consider the “network playability” a function of all and only disturbances originating from the network infrastructure supporting the game.

In the model presented by the previous chapter, network latency produces measurable inconsistencies on game terminals are decision points. These inconsistencies generate violations of the ideal law of the virtual universe which are perceived by players. The estimation of the perceived discomfort due to violation (as done in 5.3, p.92) is not a good candidate for the definition of network playability as it dependent on potentially variable and loosely estimated perception parameters. Violations are not are sometimes dependent on factors other than the network infrastructure. For example, the probability and amplitude of rollbacks required in optimistic correct synchronisation (cf. 4.3.2, p.73) depends largely on the amount and type of interaction occurring amongst players. The probability of a rollback chance is simply nil for a player standing alone while much higher in the middle of a battlefield with hundreds of participants.

It was shown in Section 4.3.4, p.80 that inconsistencies are measurable in time unit and their magnitudes only depend on the network topology and synchronisation choices of the infrastructure of online games. Furthermore, the collection of all inconsistencies precisely capture and quantify all the imperfections induced by the online game infrastructure independently of the architecture: central, distributed or hybrid. Because of these two properties, the collection of all inconsistencies endured by a player is a good candidate to represent his/her network playability. For simplicity and clarity, examples and simulations in this thesis use setups generating only one type of inconsistency which restrains network playability to scalar values. Systems generating more than one type of inconsistencies would feature multidimensional network playability for each participant which can be represented as a vector.

Considering two players with the same amount of inconsistencies, their comfort level may be different depending on the local configuration of their terminal vio-
lation (decided by the amount of co-simulation or dead reckoning for example). The state of this configuration does not influence the distribution of inconsistencies in the game. Therefore, from the game infrastructure point of view these two players are being treated equally and their violation can be tuned to create the exact same level of comfort.

5.3 Impact on players

Although the network playability of a player can be measured with his/her inconsistencies, violations of the laws of virtual environment can take different forms depending on the type of synchronisation and lag compensation techniques, if any, used. Although inconsistencies maybe easy to measure, violations can be more difficult to evaluate\(^1\) and their final impact on players is even harder to quantify because it depends on the subjective perception of humans.

This section introduces fuzzy logic tools to manipulate the concept of user impact which will lead to the optimisation of players’ comfort in Chapter 6, p.119 and Chapter 7, p.143.

5.3.1 Utility functions

As presented in the literature review Section 2.4, p.24 multiple research groups have studied the effect of latency and other disturbances on gamers.

In a modified car racing game, Pantel and Wolf\(^{120}\) evaluated the influence of artificial delay on players; objectively using performance metrics and subjectively in terms of players’ impression. The authors concluded that delays under 50\(ms\) are insignificant, 100\(ms\) was perceived as acceptable but annoying, 200\(ms\) makes the simulation unrealistic but cars can still be controlled, finally 500\(ms\) makes the game unplayable. Furthermore, the study shows that the amount of skills influences the players’ resilience to artificial delay and that the experienced

---

\(^1\)as seen in 4.4.1, p.82
driver in the test considered the game unplayable as early as 250ms of delay.

In another example using the Unreal Tournament first person shooter game, Quax et al. [124] conclude that a round trip delay below 30ms is not perceptible, players start getting appreciably disturbed from 60ms and seriously affected at 100ms.

In order to capture the results of empirical studies, one could imagine using a membership function associating the delay to a level of comfort or “utility”. Figure 5.1, p.94 shows an example of a utility function which estimates the impact of delay in the Unreal Tournament first person shooter game based on conclusions from Quax et al. in [124]: a delay from 0 to 30ms would stay below the perception threshold of most people and may have no impact on players’ comfort (utility of 1). Discomfort would start appearing from 40ms and growing stronger. At 100ms the utility is well below .5, it reaches 0 at 150ms limit where the users would be too annoyed to keep playing (utility tending towards 0).

The utility function presented in Figure 5.1, p.94 is a type-1 membership function (T1 MF) in the fuzzy set theory sense and can be manipulated with all associated fuzzy logic tools. However, this representation is not able to properly model the outcome from the work of Pantel and Wolf[120] which reveals that a skilled driver perceives the game experience degradation caused by the delay differently than average drivers. Such uncertainty on the membership itself can be represented using type-2 membership functions (T2 MF) which grades of membership are themselves fuzzy[101]. Figure 5.2, p.95 shows a possible representation of the utility of the driving game in [120] as an interval type-2 membership function (IT2 MF).

T2 fuzzy sets are difficult to manipulate and are a current active area of research. Most T2 fuzzy set applications uses interval type-2 membership functions (IT2 MF). IT2 MF are the sub-type of T2 MF that are fully described by two bounding type-1 membership functions which define the Footprint of
Figure 5.1: Approximation of a FPS response time utility with semi-trapezoidal or sigmoidal as a type-1 membership function. Based on empirical data from Quax et al. [124]
Figure 5.2: Approximation of the delay utility function as an Interval Type-2 (IT2) membership function. The utility Footprint of Uncertainty (FOU) is bounded by two type-1 membership functions representing the utility of delay for experienced and average players. Both player types stay insensitive to delay up to 50ms. From this point players perceive a degradation of the playability at different rate: the experienced player estimated the game unplayable as early as 250ms whereas less sensitive players potentially found the playability limit at 500ms. Based on empirical data from Pantel and Wolf [126].
Uncertainty (FOU). This property makes IT2 MF computationally tractable as mathematical tools applicable to regular (type-1) membership functions can be used with IT2 MF[102] whereas general T2 MF are harder to manipulate[101].

Any measurable delay, inconsistency or violation can be associated to a utility function in order to represent its perceptual impact on users in a scale between 0 and 1. Such utility functions model the relationships between inconsistency or violation types and their perceptual impact on players. Since these functions are purely subjective, their shapes depend on a multitude of variables which include, but are not limited to: game type, context, terminal capabilities and even individuals.

Confidence intervals and uncertainty in these utility functions can be taken into account using interval type-2 membership functions as shown in Figure 5.1, p.94. The characterization of these functions, which are context specific, is outside the scope of this thesis which aims to provide a generic model to analyse the effect of latency on player experience.

The utility of an inconsistency represents the combined impact of all its associated violations. By extension, the utility of a propagation delay is the combination of the utilities of all its associated inconsistencies. The additive composition of utilities is discussed in the next subsection.

For example Figure 5.1 represents the impact of the round trip time, which generates update and request age on players, and not the sole impact of the perceived response time as violation. The reason is that the Unreal Tournament game probably implements some type of co-simulation to reduce the perceived response time. The level of comfort measured in the study represents the accumulation of all perceived violations, as a whole, by the players and not the perceived response time by itself.

Even without knowing the exact shape of utility functions representing various impacts on players, we can assume some of their properties:
• \textit{Utility}(x) \in [0, 1]$, by definition.

• \textit{Utility}(0) = 1: The absence of disturbance does not reduce players' comfort.

• \textbf{Continuity}: The perception of a disturbance level by players is continuous.

• \textbf{Monotonic decrease}: The higher a disturbance level, the higher its impact on players.

Naturally, different disturbances may have different utility functions. However, a given type of disturbance may present different utilities depending on the context. For example in an online fantasy RPG, the utility of response time on fast actions (such as moving) may fall sooner and sharper than the utility of the response time on actions supposedly slow (such as casting a spell). This specific property is used in Chapter 6, p.119 to improve users' comfort by tailoring different synchronisation parameters.

In this thesis, we will approximate utility functions with semi-trapezoid type-1 membership such as the one presented in 5.1, p.94.

\section*{5.3.2 Composing utilities}

When multiple violations affect the game, their impacts accumulate. Intuitively, as violations pile up, the level of comfort can only decrease. In particular, if one disturbance makes the game un-playable (utility of 0) the presence of other disturbances cannot improve the situation. Also the absence of disturbance by a violation (when the violation is small enough to still provide a utility of 1) should not decrease the global utility of the game.

The utility of two disturbances stacking up can be computed using a fuzzy logic conjunction operator “\(\otimes\)” which happen to enforce these properties:
• The composition of two utilities is at least as bad as the worst one:

\[ a \otimes b \leq \text{Min}(a, b) \]

• Any violation with a utility of 0 reduces the utility of the whole game to 0 (0 is the invariant element)

\[ a \otimes 0 = 0 \]

• Any violation with a utility of 1 does not affect the game (1 is the identity element)

\[ a \otimes 1 = a \]

Multiple operators satisfy these criteria. The choice of the operator is, like the characterization of utility functions, context specific and should be validated with experimental subjective studies. To the best of our knowledge, such studies do not exist in the context of games. Nevertheless, even with uncertainty on the conjunction operator, Section 5.4 will show that significant information can be extracted from composed utility functions.

Violations are created by inconsistencies. Therefore the utility of an inconsistency is the composition of the utilities of all the violations it creates. The total utility of a game for a given player is the composition of the utilities of all inconsistencies, which is the same as the composition of all the violations.

\[
\text{GameUtility} = \bigotimes_{\text{All inconsistencies}} \text{Utility(Inconsistency)} = \bigotimes_{\text{All violations}} \text{Utility(Violation)}
\]  

A more mathematically precise formulation of utilities as fuzzy membership functions and their possible composition operators can be found in Appendix
C.1, p.200. Section 5.4, p.99 presents examples of utilities and the result of their conjunction with a few conjunction operators.

5.3.3 Utilities: summary

Different delays, inconsistencies and violations on players may not affect users in the same way and their impact may not be linear in relation to their magnitude. These variations are at the centre of the optimisation of user's comfort. Impact of inconsistencies and violations can be represented as fuzzy logic membership functions of type-1 or type-2 if membership uncertainty is to be taken into account. These functions can then be manipulated with fuzzy logic tools and operators.

Like most fuzzy logic membership functions, utilities are difficult to define because of their subjectivity [146]. Likewise, the conjunction operator to be used is another parameter. Empirical estimation of utility functions and conjunction operator is outside the scope of this thesis. When required for numerical simulations, membership functions will be approximated by semi-trapezoidal type-1 membership functions with sensible parameters. Whenever possible, we also avoid scenarios requiring the composition of utility functions to prevent the introduction of a conjunction operator as another chosen fixed parameters.

5.4 Altering violations and inconsistencies

Section 4.3.2, p.74 explained how by using local lag, decision points could trade two inconsistencies, decision age for action age, in order to prevent paradoxes. Furthermore, Section 4.4.1, p.82 has described how response time as a violation could be traded for local rollbacks using co-simulation on a terminal. It is therefore sometimes possible to trade disturbances at the inconsistency or violation levels. The idea behind these operations is the optimisation of user comfort by shifting some of the high impact disturbance towards a lower impact
5.4.1 Trading violations

Trading violations is always possible on terminals and sometimes applicable to decision points.

Trading violations on a terminal does not modify any inconsistencies. In other words, it is an iso-consistent operation which does not affect other terminals or decision points. Therefore, each player can tune his/her violation trade-off levels on his terminal to optimise his comfort without affecting the game or its perception by other participants.

Violation trading on terminals is one of the few techniques available in central server architectures. Consequently it has been widely used by the game industry and most of the recent highly interactive online games implement some form of dead reckoning and/or co-simulation. This is commonly done for the position of the users’ avatars which is often displayed on the terminal before being acknowledged by the game server.

Example of terminal violation trading in fuzzy logic terms

Expressed in natural language, the possible improvements to user playability through violation (or inconsistencies) trading is intuitive but is not quantified. Through an example, we show how this problem translates into fuzzy logic terms, opening ways towards the generation of optimisation problem formulation.

To illustrate the optimisation of combined utilities in fuzzy logic terms let us imagine a game in which violations utility can be estimated with the semi-trapezoidal curves of Figure 5.3, p.101. Let us name $Utility_{RT}(x)$ and $Utility_{CS}(x)$ the function returning the utility of perceived response time and co-simulation respectively. The perceived response time utility is the same as
the one presented in section 5.3.1 at the beginning of the chapter. As explained in 4.4.3, co-simulation reduces response time at the cost of probability of local rollbacks. \( Utility_{CS}(x) \) was constructed to represent the impact of local rollbacks caused by the amount of co-simulation. Because of the lack of data on the impact of co-simulation, \( Utility_{CS}(x) \) assumes that the utility of a game linearly decreases with the amount of co-simulation.

If a player is situated at a network distance of 100ms from the game central server, his total terminal inconsistency (sum of the update age and request ages = round trip time) is equal to 200ms and his perceived response time is linked to his terminal parameters by the following equality defining the trading relationship:

\[
PerceivedRT + CoSimulation = TotalTerminalInconsistency
= 200ms
\]

and:

\[
PerceivedRT = TotalTerminalInconsistency - CoSimulation
= 200ms - CoSimulation
\]
Figure 5.4: Total utility under various classic fuzzy logic conjunction operators. For any operator, the amount of co-simulation maximising the total utility $\in [A, B]$

Now, the total utility as a function of the amount of co-simulation $x$ in ms can be calculated:

$$TotalUtility = Utility_{CS}(CoSimulation) \otimes Utility_{RT}(PerceivedRT)$$

$$TotalUtility(x) = Utility_{CS}(x) \otimes Utility_{RT}(200 - x)$$

Figure 5.4 p. 102 shows the total utility as calculated by five classic fuzzy logic conjunction operators. Conjunction operators are upper bounded by the Minimum and lower bounded by the Drastic Product. Let us define the region of optimality as the zone in which the amount of co-simulation maximising the total utility is located independently of the conjunction operator used. In this example the region of optimality is the inclusive segment $[A, B]$ with $A$ and $B$ defined in Figure 5.4.
\( \forall \Theta : A \leq \text{Optimum amount of co-simulation} \leq B \)

This example shows that with defined utility functions and conjunction operators it becomes possible to find the optimum trade-off across violations. In this particular case, it was even possible to define a region of optimality even without specifying the conjunction operator.

### 5.4.2 Trading inconsistencies

Inconsistencies can sometimes be traded on decision points. However with the synchronisation scheme used by terminals in the framework of this thesis, terminal inconsistencies are not tradable. It is conceivable that alternate terminal synchronisation schemes could provide tradable inconsistencies.

Trading inconsistencies on a decision point, by definition, is not an iso-consistent operation. As explained in Section 4.3.2, p.74 and Figure 4.4, p.75, local lag increases the order age of all terminals connected to the decision point, and in turn reduces the state age it causes on all other decision points. Local lag on a decision point is a sacrifice as it does not benefit itself or any of its terminals; instead it benefits all other decision points and terminals. Because it alters the balance of inconsistencies across terminals and decision points, changing local lag can affect not only playability but also fairness.

**Network playability space**

As discussed in Section 4.3.4, p.80, the propagation delay introduces different inconsistencies depending on network topology and the synchronisation scheme. Let us consider a multidimensional space where each axis represents an inconsistency, hereafter referred to as playability space. A tangible measure of each inconsistency can always be obtained in a given game and players can be positioned as points in this space which then represents their network playability. The origin point is the ideal case of perfect game playability where
Figure 5.5: Playability space and playable zone. Players can be located in a space representing the inconsistency they endure. The further from the origin, the less playable the game is. Player located outside the playable zone may not enjoy the game because of their level of inconsistencies.

there is no inconsistencies whatsoever (cf. 4.3.4 p.80). Therefore, the closer a user is to the origin point, the more playable the game is for this player.

Depending on the perceptual impact (ie. utility function) of the different inconsistencies, a playable zone of a game could be defined as the zone where players would be considered as content to play. Such an area is illustrated in 5.5 p.104. Players outside the zone would consider the game not worth playing or unplayable. Playable and unplayable areas do not have a simple line boundary but are separated by a transition zone. The shape and boundaries of these zones would depend on the utility functions of the inconsistencies potentially present in the game.

Moving participants in the playability space

The two dimensional playability space with the three participants from the
Figure 5.6: A two decision points distributed architecture. This topology is used to illustrate the movement of players in the playability space when modifying the servers’ local lag.

topology shown in Figure 5.6 p.105 with various local lag configurations are presented in Figures 5.7 p.106 and 5.8 p.107. The horizontal and vertical axes respectively represent the terminal inconsistency and action age each player is subject to.

Trading inconsistencies moves players in the playability space: Figure 5.7 p.106 shows the evolution of the three players in the playability space is presented when local lag on server S2 alone increases from 0 to 100ms in 20ms steps. P2 moves horizontally, paying the price of the local lag on his server with an increased terminal inconsistency. P0 and P1 see the action age on their server improving. P2 moves away from the origin and its total amount of inconsistency increases while P0 and P1 get closer to the origin reducing their total level of inconsistencies.

Figure 5.8 p.107 shows the movement of the same three players when local lag on both servers S1 and S2 increase symmetrically from 0 to 100ms in 20ms steps. In the particular case of all servers applying the same amount of local lag, the transformation is pseudo iso-consistent since each player see his terminal inconsistency growing at the same rate the action age of his server is

\[1\text{The transformation is iso-consistent up to the saturation of one or more decision points: when the local lag becomes higher than the inter decision point delay, then the further loss of playability of some player may not be beneficial. This problem is discussed in 5.3.4 p.107.}\]
Figure 5.7: Moving players in the Playability Space. Adding local lag on only one of the two servers moves one of the players away from the origin (worse playability) and moves the other closer to the origin (better playability).
Figure 5.8: Moving players in the Playability Space (2). Adding local lag to both servers displace the players on a parallel line.
decreasing. Naturally when the servers apply enough local lag to cancel out the action age they induce on other decision point, the game becomes conservative.

With good approximation of the utility functions associated with various inconsistencies, the fuzzy logic tools and techniques presented in 5.4, p.99 can be used to model the combined impact of the various inconsistencies on individual players. Such a model could lead in future work to the formulation of a complex optimisation problem select the best synchronisation parameters in order to maximise user's comfort.

Another way of moving a participant in the playability space is to artificially decrease his playability by adding delay before sending update messages for example. This operation moves a user away from the origin and can be used to improve global game fairness as discussed in next section 5.5, p.109. However, one cannot pull a player towards the origin without sacrificing someone else's playability or modifying the underlying topology and distribution of decision points.

5.4.3 Trading inconsistencies and violations: summary

Inconsistencies and violations are sometimes tradable. The modelling of the user impact of tradable network disturbances using fuzzy logic tools can lead to the formulation of an optimisation problem. One example of violation trading optimisation was provided, chapter 6, p.119 will present an inconsistency trading optimisation formulation.

Up to now, all alterations imagined have supposed a constant network topology. However, network topology can also be modified to alter propagation delays and inconsistencies. For example, moving a central server may increase the terminal-decision point delay for some terminals but decrease it for others. In mirrored architectures, servers can be added or removed rearranging some terminal's delays and affecting server inconsistencies. Some types of topology alteration are discussed in Chapter 7, p.143.
5.5 Fairness

As discussed in the previous sections, network playability is influenced by the different inconsistencies for a given player and stems from the position of that player in the playability space. However, playability is not the only factor of user comfort: a user may well be located in the playable zone and still having a frustrating online game experience if it is found that the game is not fair.

In this section, we will define the concept of fairness and its connection to playability. Simulation will show the correlation between the theoretical expression and experimental measure of fairness.

5.5.1 Source of unfairness

While playability is an independent attribute of individual players, we would like to propose that fairness is a notion concerned with relative playability amongst the players. Our view is that variations in playability between players are the source of unfairness. Giving similar inconsistencies, and therefore network playability, to all players in an online game would provide them with similar average opportunities, creating a fair game from the network point of view.

A fair game is not necessarily playable. For example any game where all players are at the same distance (in terms of network delay) from a central server is perfectly fair network-wise. In general, it is easy to enforce fairness by artificially raising the level of inconsistencies of all participants to match the worse-off player. However, if the worse-off player is outside or at the edge of playable zone, the quality of the game is degraded for everyone beyond acceptable level.

For an enjoyable game experience, a game should be fair and playable, *ie.* all players should have similar locations in the playability space (fair game) and all within the playable zone (playable game).
5.5.2 Estimation of network fairness

In order to introduce a definition of network fairness, we first require a game set up in which the network playability of each player can be represented by a single scalar value. For this purpose we propose to consider a conservative game (ie. single central decision point or distributed decision point using a conservative synchronisation such as local lag) which enables to collapse all forms of inconsistencies as action age on the terminals for all players. In these conditions, only one type of inconsistency remains in the system and the network (un)playability of each player is represented by the amount of action age his terminal endures.

As a first step, the standard deviation of the terminal inconsistencies amongst the participants seemed to be a good candidate to estimate the (un)fairness of a game as it precisely quantifies the spread of inconsistencies. However, We discovered that fairness is not only a notion attached to the variations in playability amongst participants, but is also associated to the magnitude of these variations in relation to the average level of playability. And as such, standard deviation does not scale properly with the average amount of inconsistencies to be a good metric of fairness.

For example, in two game sessions presenting the same standard deviation but different average inconsistencies:

- Game A. Inconsistencies are $P1 = 20ms$, $P2 = 50ms$ and $P3 = 80ms$. Average inconsistency is $50ms$, standard deviation equals $24.5ms$.

- Game B. Inconsistencies are $P1 = 170ms$, $P2 = 200ms$ and $P3 = 230ms$. Average inconsistency is $200ms$, standard deviation equals $24.5ms$.

Although $P1$ has an advantage in both game, this advantage is not as important in game B as the level of inconsistency variations is less important relatively to the mean of inconsistencies endured by the players.
As a result, we propose to use the coefficient of variation (CV) of inconsistencies which is defined as the standard variation divided its mean. The CV captures both the variations in playability amongst participants and the intensity of these variations in relation to the average level of playability.

\[
CV(x) = \frac{StdDev(x)}{Mean(x)}
\]

In the last example, fairness calculated using the coefficient of variation show Game A (network fairness 0.49) to be less fair than Game B (network fairness = 0.12).

Note that if all participants have the same level of network playability (unfairness of 0) they have the same handicap and therefore should all have the same chances of winning the game as far as the network conditions are concerned. The particular setting of conservative synchronisation to measure the coefficient of variation of players’ action age will be used later in this thesis to measure the network playability provided by particular topologies in Chapter 7, p.143.

Our definition of fairness can be generalised to games with multiple concurrent type of inconsistencies by calculating the CV of each inconsistency type. As the network playability of each player would be a vector, the network fairness would also be represented as a vector of as many dimensions as playability as opposed to a scalar like we use in this thesis.

Aggarwal et al. [3] also measure fairness as the spread of a variable representative of playability. Fairness was estimated using the standard deviation of player’s dead reckoning error which is a measure of playability in the setting of their experiments with the open source game BZflag. In his empirical studies of FPS players[71][72], Henderson observed that "relative delay" is more important to players than their "absolute delay". The author measures this relative delay with four different metrics: the standard deviation of all delays, the ratio of one player’s delay to the average, the ratio of the one player’s delay to the
lowest delay in the game and the ranks of the player in terms of delay. Ferretti et al. also acknowledge in [51] that a fair game should guarantee the same possibilities of victory to all players. However their estimation of fairness based of the percentage of events reaching all players simultaneously is not applicable to our set up because of our different set of assumptions (such measure would return either 100% if all players have the same terminal inconsistency or 0% if any player has a slightly different value). Other research groups working on fairness enforcement define a game as fair if all players perceive all events simultaneously[148][10][62][29] or are given the same amount of time to react[69].

5.5.3 Experimentally measuring fairness

From a user's perspective, a game may be considered fair if it provides the same opportunity to all players in similar conditions. However, the evolution of the game state, depends on many variables other than the game infrastructure such as the initial game state, skill level of players and the intentional element of randomness of the game.

A special game competition

We have developed a technique to measure the fairness of a game infrastructure in a simulator by neutralising any variables other than the network topology and synchronisation parameters:

- Identical players: a software emulates players with absolutely identical reaction patterns.

- Controlled randomness: all randomness within the game is tied to an initial random seed which become part of the initial condition.

- Cancelling the influence of players's initial conditions by playing all their permutations.
Our method involves a scoring system and a technique for cancelling the influence of initial conditions which is not without similarities to techniques used in team sport competitions. In a soccer match, for example, each game is divided into two half sessions for the teams to switch their initial conditions (side and kick off). We also cancel the impact of initial conditions for an arbitrary number $N$ of players by running all the different possible permutations of the initial conditions (that is, $N!$ simulated games). A more complete explanation on how playing all permutations neutralises the initial conditions is provided in Appendix D, p.204.

Scores in sport competitions, such as number of goals in soccer, are used as a performance measure of individual players or teams. In general getting the best score is often the aim of a game. Most competitive video games also involve a scoring system to evaluate how well their participants play. A scoring system is a method to project the game state trajectory of a game session to a list of scalar values representative of the players and/or teams performance.

Assuming fair playing conditions the score of a competitive match determine which team has the best skills. Our aim is the opposite: using synthetic players with the exact same skill level, the game score will evaluate the fairness of the game infrastructure.

**Game simulator**

In order to validate our theoretical expression of fairness and illustrate the impact of game infrastructure on fairness, we use NetGameSim, a game simulator we developed capable of simulating a whole session of a simple online game under different conditions. The full description of this software along with its features can be found in Section 6.2 (p.121). Suffice to mention here that it can simulate both central and distributed architectures using correct inter-decision point synchronisation with variable local lag.

Here is a short description of the game itself. Artificial players, or bots, fight
in a configurable virtual battleground, a 75m side square in the case of the simulations in this chapter. Avatars have two different possible actions, moving to seek a target or shooting if a target is in their vicinity. The winner is the last standing avatar in the battleground.

After each game session, avatars are ranked in the increasing order of their lifetime. The first player to have died gains 0 point, the second 1 point, the third 2 points and so on. Exactly like two permutations of the initial conditions makes the two half-time of a soccer match between two teams, $N!$ permutations makes one game match of a $N$ players game. Scores are added individually over all the permutations. To obtain meaningful data, multiple permutation groups, or matches, are simulated and averaged on the same network topology.

To be fully comparable, the scores of players are normalised by the average number of points distributed. Therefore, if a bot has an advantage due to the topology, its normalised score will end up above 1.0, and less than 1.0 if it is being handicapped. We define our experimental measure of fairness as the standard deviation of the normalised player’s scores. The closer our fairness measure is to 0, the fairer the game is.

Typically we used 100 permutation groups. The comparison of the averages, over this large number of permutation groups, of each bot’s individual score playing in the same network topology, gives a good idea of the fairness of the game.
Figure 5.10: Score of players vs P0’s distance from the server. As P0 is pulled away from the server, his amount of network playability decrease relatively to the fixed players P1 and P2, hence P0’s score decreases as relative scores of the fixed players increased. At a distance of 100ms, all players have the same network playability and the scores all equal.

Figure 5.11: Game (un)fairness vs P0’s distance from the server. As P0 is pulled away from the server, the network unfairness closely match the observed unfairness based on the relative game score of players.
Comparing experimental and objective fairness

To validate the definition of network unfairness based on the spread of the players in the playability space, we compare this calculated network unfairness with the measured score unfairness in a series of game topologies where a bot endures increasing amount of consistencies. The simulated topology is shown in Figure 5.9, p.114. Players $P_1$ and $P_2$ delays are fixed at 100ms propagation delay from the central server while $P_0$, starting at 40ms is incrementally moved away from the server by 20ms increments.

In terms of inconsistencies, each player endures an amount of $ActionAge = RequestAge + UpdateAge$, equal to their round trip time to the game server. As $P_1$ and $P_2$ do not move, their action age is 200ms at all time. As $P_0$'s inconsistency varies linearly with its distance from the server, Figure 5.10, p.115 show the normalised score of each players. As the amount of $P_0$'s inconsistency increases relatively to fixed players $P_1$ and $P_2$, its game score decreases. Although the amount of inconsistency of bots $P_1$ and $P_2$ is constant, their game score increase as they gain an advantage over $P_0$ decreasing network playability. Note that all scores are equal in the perfectly balanced topology when $P_0$ distance from the server is also equal to 100ms.

Figure 5.11, p.115 presents the game unfairness calculated from the network playability and the unfairness derived from variation in the game score. As both these functions where normalised, they are comparable. As predicted, there is a strong correlation between the network fairness as the coefficient of variation of playability and the measured fairness as variation of the players' normalised scores in the game. When all players are equidistant from the server, their scores are equal and the game is perfectly fair according to both measures.

5.5.4 Fairness considerations

Fairness is the second aspect of user experience and is concerned with relative playability among the players. Our proposed definition of network fair-
ness based on the variation of network playability (or inconsistencies) correlates strongly with its experimental measure based on players’ score.

Network playability and fairness are only concerned with objective and measurable inconsistencies coming from the network infrastructure, these notions do not take into consideration human perceptions factor as discussed in 5.2.2, p.90.

If perceived playability is estimated, for example using the fuzzy logic tools introduced in 5.3, p.92, one can estimate a version of the perceived fairness using the coefficient of variation of the estimated perceived playability amongst players. Chapter 7.5.2, p.167 estimate perceived playability and fairness when solving a server topology optimisation problem.

In the same way utility functions may represent the perceived impact of inconsistencies or violation on players, utility functions can also be used to represent the impact of fairness (or unfairness) and complete the representation of participants’ game experience. Estimation of such utility functions would require significant amount of empirical research which is outside the scope of this thesis and is left for future research.

5.6 Conclusion

The framework of this thesis was opened in Chapter 4, p.63 with the analysis of the cascading disturbances introduced by the propagation delays in any networked game architecture. This chapter provided a complement for this setting with a definition of user experience as a function of these disturbances.

We define the network playability of a given participant the collection of all inconsistencies he endures. Fuzzy logic membership functions can represent the impact of violations and inconsistencies on users as utility functions which define the perceived playability.
The second element influencing the online experience of players is the amount of *fairness* provided by the game defined as the spread of (network or perceived) playability amongst participants. The quality of any player experience is a function of his playability and the relative fairness of the game.

Parts of the contributions of this chapter were published and presented at the international workshop on Networking Issues in Multimedia Entertainment (NIME) in the 3rd IEEE Communications and Networking Conference (CCNC) in January 2006 [21].

Chapter 4, p.63 and 5, p.88 aimed at setting a framework which the remainder of the thesis uses to develop enhancements of game infrastructure. The next chapter presents improvement to state synchronisation in games maintaining correctness. Chapter 7, p.143 focuses on the optimisation of decision point topology.
Chapter 6

Improving playability: distributing servers and tailoring local lag

Synopsis

Based on the framework, definitions and tools previously developed, this chapter focuses the synchronisation aspect of the network game infrastructure and propose two generic software techniques capable of improving playability in online games.

Content:

- Description of our game simulator NetGameSim
- Finding a trade-off in the continuum between conservative and optimistic synchronisation
- Unbinding the synchronisation fate of different type of actions
6.1 Introduction

In Chapter 4, we analysed the chain leading from network delays to violations of the game ideal laws with different synchronisation techniques. Chapter 5 studied the effect of these violations on users' game experience. We showed how the impact of different violations can be aggregated into a measure of playability using fuzzy logic operators. Fairness, the other component of user experience, has been presented as the variation of playability amongst players.

With a well defined framework to understand and analyse the influence of network on the players' game experience, we can now work on the two aspects of the game network infrastructure to improve playability and/or fairness. As introduced in Chapter 1 and further discussed in Chapter 4, the network infrastructure of a game is composed of (i) the network topology between the elements involved in the game and (ii) the scheme(s) used to synchronise decision-points with other decision-points and terminals.

Logical central server architectures\(^1\) cumulate all their network delay in the Terminal-DP inconsistency, leaving little room for improvement in the synchronisation for which dead reckoning has been already largely researched and discussed. The remainder of this thesis largely focuses on distributed architectures for which results can be applied to both peer to peer and distributed server variations.

While Chapter 7 will aim to enhance users' experience by altering topology, this one focuses on the synchronisation aspect of the network game infrastructure. We propose here two new synchronisation techniques to improve playability: the first one is to offer a trade-off between full conservative and pure optimistic state synchronisation via the tuning of the local lag on servers. With the second technique, we propose to unbind the fate of game state parameters by tailoring processing considering their different requirements.

\(^1\)Partitioning of the virtual space on multiple physical machines, such as Locales and Instance, is considered being a single logical central server
First, the next section will introduce \textit{NetGameSim}, the simulator we developed to study and test the influence of synchronisation parameters on user experience. In Section 6.3, we propose to tune the amount of local lag based on the topology of the decision points in order to find the best trade-off between a low number of roll-back and a good response time. The effectiveness of the scheme is shown through NetGameSim simulation which are also compared to experiments made by another researcher with a distributed Quake3 games. Section 4 extends the idea of local lag tuning by unbinding the different action types from the same synchronisation parameters. The unbinding technique is simulated in NetGameSim by applying different values of local lag to different actions. Finally Section 5 will conclude on synchronisation optimisation.

6.2 NetGameSim: a network game simulator

In order to measure the performance of various topology and synchronisation configurations we developed NetGameSim, a discrete event simulator capable of simulating whole sessions of a simple networked game under different conditions.

6.2.1 Overview

Entirely written in C, NetGameSim is a discrete event simulator composed of two modules. The ‘terminal’ module simulates players on their terminals and the ‘decision point’ module simulates the decision point(s) of the game. Each module instantiates as many independent terminal(s) and decision-point(s) as required by the simulation. Instantiated objects interact via an event-queue simulating network packets.

The three type of events (simulated network packet) are:

- action order from a terminal to associated decision point
**action forwarding** which is a action order forwarded by a decision point to another one

**state update** from a decision point to one of its associated terminals

Terminal instances simulate players' reactions to *state updates* (their individual perception of the game state) and send them to their associated decision-point in *action orders*. Information exchange is artificially delayed to simulate propagation delay of real network links. Decision-point instances forward their terminal(s) action order(s) to other decision-point(s) in *action forwarding* and process players' actions according to synchronisation protocols such as Local Lag and TimeWarp if necessary. The new game state is sent to terminals in individual *state update* events.

The topology of decision-point(s) and terminals in the simulator is fully configurable along with inter-decision-point synchronisation parameters. Because NetGameSim maintains correctness, convergence of game states across multiple decision-points is natural (cf. 4.3.2, p.73).

A game's behaviour is not only influenced by its network infrastructure but also by its game rules. The more rules in a game, the more complicated interactions appear. In such a context, the impact analysis of individual parameters such as Terminal-DP delay, inter-DP delay or local lag becomes difficult if not impossible. Moreover, a set of complex rules would specialises the simulator into producing less generalisable results without necessarily making the simulations more realistic. Many other research group developed simple games for academic research [43][85] [83][115].

For these reasons, the rules of the game in NetGameSim were kept simple. The game state is composed of the health and position of each avatar participating in the game session. Players can either move in the virtual environment or attack another avatar. An avatar being shot has a fixed chance of surviving and remaining in the game session. A game session ends when only one avatar
survives or when a time-out is reached, whichever comes first.

6.2.2 Player and Decision-Point state machines

Each simulated player attempts to shoot at the closest enemy within sight range. If no other avatar is in sight, the artificial player moves at maximum speed in a random direction for a random amount of time, or until an enemy appears in sight.

In simulated decision points, players’ actions are stored in buffers and can be re-evaluated if required. This enables all servers to independently reconstruct correct game state even when players’ actions are received in different orders. Correctness here has the same definition as described by Mauve in [99]. When an action is received later than the time it is supposed to have happened, a server has to re-evaluate the correctness of its game state.

To this aim, the simulated decision point first applies the late action to the latest known game state, in effect executing some actions out of order. It compares this possibly erroneous game state with the result of the re-execution of all actions in its buffer since and including the late action, which is in effect a true RollBack. If the two resulting game states are not identical, the late action was correlated (in the same sense as Ferretti [49] and Palazzi [118]) to one or more previous actions in the buffer and a rollback is required. Otherwise the late action was uncorrelated and could safely be executed out of order.

The use of deterministic random generator and synthetic players give NetGameSim a truly deterministic behaviour. That is, given the same initial conditions (including a seed to initialise the random generator) two simulations produce the exact same output.
Figure 6.1: A snapshot of a NetGameSim game session. Players are represented as crosses and labelled with their name and number of hit point left. Shooting actions are listed as text in the top left and represented by arrows in the virtual battleground. Position scale is in millimetre (mm), which is the resolution of the simulator.

6.2.3 Simulation outputs

NetGameSim can record all events as they happen in each simulated decision point including reception and execution of actions, rollbacks and resulting game states. A simulated game session cannot be visualised in real time because it runs too fast. However, snapshots of the players’ actions and positions in the virtual environment, like the one shown in Figure 6.1, can be used to visually reconstruct the game session afterwards.
6.3 Trading-off conservative and optimistic synchronisation

In distributed simulation cf. 2.6.2, p.37 there is no consensus on the use of conservative or optimistic synchronisation. Where some have warned about the usage of optimistic techniques [112], others have argued that it is possible to run distributed simulations without any synchronisation and still get useful results [126].

Also, in distributed network games, there is simply no “one size fits all” solution: what is best for a game depends on a large number of parameters. The aim of this section is to show that pure optimistic and conservative are not a binary synchronisation design choice but two extremes of a continuum of solutions.

6.3.1 Conservative vs. Optimistic

When Local Lag was introduced into distributed real-time media synchronisation [96], the author suggested adjusting the local lag values in order to maintain a truly conservative environment. Timewarp was presented as a fall-back in case of packet loss or high jitter.

On the other hand the Trailing State Synchronisation (TSS), a time warp variation, implemented by Cronin et al. on a distributed Quake platform is purely optimistic[37].

While conservative algorithms ensure the absence of paradoxes, aggregating all the network latency on the Terminal-DP inconsistency often results in a poor response time from the player’s point of view. Optimistic synchronisation tends to favour the terminal-DP inconsistency and provides a better response time at the price of possible paradoxes.
6.3.2 Tuning Local Lag

Tuning the amount of Local Lag is one method to control the balance between optimistic and conservative synchronisation: any amount of symmetric local lag\(^1\) tends to reduce the decision point inconsistency and therefore the probability of paradoxes. The closer the local lag is to its conservative threshold, the more conservative the system becomes. And reciprocally, the less local lag, the more the system behaviour tends to be optimistic.

Depending on the situation, a game may neither require full conservative paradox avoidance nor be acceptable using only an optimistic state healing or convergence. This is why it is necessary to tune the local lag to achieve optimum performances and comfort.

As an example, assuming human beings have a perception threshold of 30ms, a value close to what has been reported by different FPS studies \([124][11]\), players are not affected by a Terminal-DP inconsistency of less than 30ms. Therefore, a local lag of 30ms in a peer to peer game can only improve the playability by reducing the frequency of paradoxes without affecting the perceived responsiveness of the game.

Another example, assuming that both Alystan and Pazoo are trying to take an object from the ground. Let us say Alystan actually gets it first but due to insufficient server local lag, the same object is also given to Pazoo. The object is paradoxically inside the inventories of both players before a time warp removes it from Pazoo. Does it actually matter? It may not, if examining the inventory requires, like in many games, clicking on an icon in the game interface. Pazoo may not have noticed the paradox at all.

Figure 6.2 illustrates the continuum of possible local lag value between conservative and optimistic. The optimum game playability is probably found in a

\(^1\)A discussed in 5.4.2, p.103, Local Lag on a decision point is a sacrifice which only benefit other decision points. We assume here local lags are applied on all servers in a reciprocal fashion.
Figure 6.2: Variable local local in a time diagram. As the local lag on S1 increase, the inconsistency of payer P1 increases and the inconsistency of server S2 decreases.

trade-off between good game responsiveness (provided by a low local lag) and a function of paradox’s impact and probability of appearance (which is improved by a high Local Lag).

6.3.3 Simulations and results

In term of fuzzy logic, finding the optimum value of the local lag is a similar exercise as looking for the best amount of co-simulation on a terminal [5,4, p.102]. While we currently have an estimation of the utility function of the perceived response time[1], we now need to know how local lag (which reduces the decision point inconsistency) affects the number of paradoxes.

[1]In NetGameSim there is no co-simulation on the terminal, so all terminal inconsistencies generate perceived response time violation
Using NetGameSim, we can observe the effect of different local lag values on the response time and the occurrences of rollbacks which are the two components of network playability (cf. 5.2.1, p.90).

The network topology used in our simulation is shown in Figure 6.3. The virtual battleground is a 100 by 100 metres square. Avatars can move at the speed of 10 metres per second. They also need to be shot successfully 5 times to die. The simulator runs until only one (or none in some rare cases) avatar is alive or after expiration of 200 seconds, whichever comes first. The average length of a simulated game session was 120 seconds.

During the simulation run, the number of rollbacks is logged. All values presented in this paper have been compiled as the average over 100 simulation runs with identical parameters except the initial random seed.
Figure 6.4: Tuning Local Lag: Response Time and Rollbacks

Averaged results from a set of simulations under increasing local lag are presented in Figure 6.4. It shows the variations of the Response Time and the number of required rollbacks (forced by the appearance of paradoxes) when the Local Lag increases. For comparison, at any time, the distributed servers provide a better or equal response time than the optimally located central server. The average response time increase linearly with the local lag following the formula:

\[
RT = 2 \times D(P, P_{server}) + LocalLag
= 60ms + LocalLag
\]  

(6.1)

6.3.4 Discussion on Local Lag tuning

A predicted, a low local lag provides the best response time whereas a Local Lag of 100ms (the maximum delay between any two servers) guarantees a
Figure 6.5: Tuning local lag in a real game. Posterior to our publication [20], Liang implemented the technique in a modified distributed Quake3 game and published his results in [92] and [93].

paradox-free game with no rollback. However, for this particular game in this particular network topology and conditions, a local lag of 60ms also shows no rollbacks. In this situation it seems reasonable to choose 60ms as a local lag value as opposed to 100ms, the maximum delay between any two servers. And if the consequence of a paradox are not so important, the optimum trade off may even be found in a smaller value of the local lag.

These simulations show that tuning the local lag can improve a game’s playability by enhancing responsiveness without deteriorating causality.

It is interesting to compare our simulations with experimental data from Liang published in [91] and [92] posterior to our publication [20]. In this work, the author modified the Quake3 network code enabling the FPS game to run on distributed servers using both local lag and rollbacks synchronisation schemes. The experimental setting involved two distributed Quake3 servers at a distance
of 200ms serving one bot (artificial player) each. Figure 6.5 presents the frequency of rollbacks (called timewarp in Liang’s work) versus the amount of local lag on the servers published by Liang in [91] and [92].

These experimental results extracted from a real game are in agreement with the output of our simulations presented in Figure 6.4. This correlation supports the general reliability of NetGameSim as a simulator. Although there are differences between Liang’s experimental setup and our simulations, both experiments show a very similar non-linear trend in the reduction of rollbacks when increasing local lag. An increase in local lag tends to reduce the number of paradoxes significantly more in the low local lag range than in the higher range.

There are two reasons for the number of paradoxes not to be linearly connected to the local lag:

1. in many cases (and in our setup) local lag does not reduce decision points inconsistency in a truly linear fashion

2. the number of paradoxes do not increase linearly with the decision point inconsistency

Relation between Local Lag and decision point inconsistency

The amount of inconsistency on a given server \( \alpha \) (as event age) is a function of the network topology and the amount of local lag applied to actions on other decision points. Given that \( S \) is the set of servers, \( LL_\alpha \) the amount of local applied by server \( \alpha \) (in ms) and \( D(\alpha, \beta) \) the network distance between servers \( \alpha \) and \( \beta \) (in ms), the sum of event age inconsistency \( I_\alpha \) on server \( \alpha \) can be calculated in the following way:

\[
I_\alpha = \sum_{\beta \in S} \max(0, D(\beta, \alpha) - LL_\beta)
\] (6.2)
Each server $\beta$ introduces on server $\alpha$ an amount of event age equal to its distance to $\alpha$ ($D(\beta, \alpha)$) minus the local lag server $\beta$ applies locally ($LL_\beta$). The event age, however, cannot be negative even if the local lag is superior to the distance between the two servers ($LL_\beta > D(\alpha, \beta)$), hence the presence of the $Max$ function to ensure a minimum event age of 0. In our scenario, as all servers use the same amount of local lag $LL$, the sum of inconsistencies is:

$$I_\alpha = \sum_{\beta \in S} Max(0, D(\beta, \alpha) - LL)$$

(6.3)

For the particular topology we used in our simulation, Figure 6.6 p.[133] shows the relationship between the introduced local lag and sum of event age inconsistency on any one server as calculated by equation 6.3. In our simulation, as it would be case the case in most real life situation involving more than 2 servers, all decision point pairs are not at equal network distance. Looking back at Figure 6.3 p.[128] we can see that $D(S1, S2) = D(S1, S3) = 71ms$ while $D(S1, S4) = 100ms$. Because of this topological feature, the addition of local lag up to 71ms reduces the inter decision-point inconsistency from all three other servers on $S1$, but a higher Local Lag (up to 100ms) only reduces the inconsistency caused by $S4$. Hence, one local lag does not necessarily reduces the amount of inconsistency in the system in a linear fashion.

**Relation between inter-DP inconsistency and paradoxes occurrence**

As discussed in Chapter 5.2.2 p.[90] the amount of rollbacks as a violations is a function of the event age as a network inconsistency, but also depends on other parameters unrelated to the network infrastructures. The rules of the game and the actions of players also have a major influence of the chances of generating a paradox. Using the relationship between local lag and decision-point inconsistency, we could plot the number of rollbacks (or paradox occurrences) as a function of the event age inconsistency in Figure 6.7 p.[133]. For the particular setup of our simulation, the number of rollbacks presents an exponential trend.
Figure 6.6: Decision point inconsistency vs. Local Lag. In the topology used for our NetGameSim simulation (see Figure 6.3 p.128).

Figure 6.7: Rollbacks vs. inter-DP inconsistency. In the topology used for our NetGameSim simulation (see Figure 6.3 p.128).
in relation to the amount of event age inconsistency. Hence, the relationship between inconsistency and number of rollbacks is not necessarily linear.

**Locating the optimal Local Lag value**

Local lag controls the trade-off between the terminal and decision points inconsistencies. These inconsistencies get translated into their corresponding violations: perceived response time and probability of rollbacks. We need the utility functions of these violations representing their impact on the playability of participants. These utilities will enable us to search for the amount of Local Lag offering the best trade-off with the same method applied in 5.4.1, p.100 when looking for the best trade-off in the amount of co-simulation on terminal.

The membership utility functions for perceived response time and rollbacks we assume are shown in Figures 6.8a and 6.8b p.135 respectively. The perceived utility of response time is the same as previously defined in 5.3.1, p.92 and 5.4.1, p.100. The Rollbacks utility is assumed to be inversely proportional to the number of rollbacks with 8 rollbacks being unacceptable (utility :: 0).

Figure 6.9, p.135 presents the composed total utility of the simulated NetGameSim sessions game using different fuzzy logic operators. As explained in 5.4.1, p.100, the minimum and drastic product operators provide an upper and lower bound respectively to the total utility. Therefore, the amount of Local Lag optimising the overall utility for this particular setup (topology and utility functions) is located somewhere between 5 and 60ms.

**6.3.5 Conclusion on balancing optimistic and conservative synchronisation**

As in distributed event simulations, there is no one synchronisation solution that is suitable for all games. We propose not to restrict the search for the best synchronisation strategy to a binary choice between optimistic or conservative approaches, but rather toward a continuum of possible solutions between these
Figure 6.8: Semi-trapezoidal estimations of response time and rollback utility functions

Figure 6.9: Total utility when varying the local lag. The composed utility is calculated with various fuzzy logic conjunction operators. For any operator, the amount of local lag maximising the total utility is [5ms, 60ms] and is neither a purely optimistic nor a conservative value.
Improving playability: distributing servers and tailoring local lag
two extremes.

In our simulations, we used the local lag technique to control the balance between optimism and conservatism used to synchronise distributed servers. Analysis of the results showed that optimum playability was reached in a trade-off between pure optimistic and truly conservative synchronisation.

6.4 Tailoring synchronisation parameters per actions

After showing that finding a good trade-off between optimistic and conservative synchronisation could improve the game playability, we now propose to go further in the synchronisation tuning process by adjusting the synchronisation parameters of different action types independently.

6.4.1 Unbinding actions’ synchronisation fate

As explained in 3.4, p.55, the game’s virtual world is fully described at any time by its game state. This game state is composed of a list of parameters whose values can vary in time. The list of parameters can itself change while objects or players are added or removed from the virtual world. Examples of parameters are: avatar positions, position of in-game object, in-game time etc.

The architectures proposed in the literature on synchronisation of distributed games (such as [37][43][99][106][148][148][62]), synchronise all game state variables in the same way. However, different parameters of the virtual world may represent a variety of different in-game concepts, and actions influencing them may have totally different requirements in terms of response time and paradox avoidance.

For example, in the case of many online role playing games, an error on avatar’s position error may not affect actions of other participants in term of decisions
made on inconsistent servers (due to limited acceleration and speed). Yet, players would want to see their avatar moving quickly once they decide to do so. On the other hand, a paradox on an avatar’s life stat (dead or alive) may have significant effect on the game’s playability because the result of many actions depends on this variable meaning that optimistic synchronisation of actions that results in changing such a variable is more likely to create one or more paradoxes.

Therefore, binding actions with different needs (because they alter different type of state parameters for example) to the same synchronisation scheme may not be effective. We propose to tailor the state synchronisation process of each action to the specific requirements of their associated state parameters.

Considering our previous game example with a “per action type” local lag, we hypothesise it could be beneficial to apply little to no local lag to moving actions and higher local lag to shooting actions. This would provide a good response time to avatar’s movement and good paradox avoidance on the life state.

### 6.4.2 Simulations and results

The hypothesis was tested in NetGameSim with tailored Local Lag per action. Our simulated game allows players to perform only two types of actions: Moving and Shooting. All combinations of local lag values from 0 to 100ms have been applied in 20ms increments to these two types of actions in the simulator while recording the average response time and occurrence of rollbacks. Other simulation parameters were identical to the simulations run in Section 6.3.3, p.127 (network topology: Figure 6.3, p.128; virtual world: 100 by 100 metre square; Avatar speed=10 m/s. Avatar hit points: 5; Maximum game session time: 200s). Presented results are based on the average of 100 simulation run with identical parameters apart from the initial random seed.

Contours in Figures 6.10 and 6.11 respectively show how the number of rollback occurrences and the average response time changes when varying both the local
Figure 6.10: Rollbacks contours versus shooting and moving local lag. Contours show the bilinear interpolation of the raw simulation results from Table A.1 in Appendix A. The line labelled ‘2’ is the location where the number of rollbacks is equal to 2.

Figure 6.11: Response time (i.e. terminal inconsistency) contours versus shooting and moving local lag. Contours show the bilinear interpolation of the raw simulation results from Table A.1 in Appendix A. The line labelled ‘100’ is the location where the action age (response time) is equal to 100ms.
lag for moving and shooting actions. Both Figures have been projected from the two dimensional bilinear interpolation of the raw simulation which can be found in Appendix A, Table A.1 A.2 respectively.

In both figures, each line represents locations where the plotted parameter remains constant. For example in Figure 6.11 the line labelled ‘100’ is the location where the response time is equal to 100ms. These contours illustrate the Rollbacks and Response Time 3D surfaces.

6.4.3 Discussion on tailoring Local Lag to action

Results in Figures 6.10 and 6.11 show that local lag applied on Moving actions slightly reduces the number of Rollbacks but significantly increases the response time. On the other hand, local lag on Shooting actions provides better Rollback reduction with only little influence on the average response time. Therefore, the best local lag configuration would be: high Local Lag applied to shooting and small to no local lag on moving actions.

The simulations confirm the hypothesis that little local lag on moving actions and higher amount on shooting actions can provide good average response time and rollback reduction than binding these two action types to the same synchronisation parameter.

Interpretation on Response Time variations

These asymmetric results come from the intrinsic rules of the simulated game: avatars move more than they shoot. Therefore, Moving actions are far more numerous than Shooting actions and local lag on Moving action affect the average response time more than local lag on Shooting actions.

Interpretation on Rollbacks variations

Under the rules of this particular game and from a state parameters point of view, a paradox on the position of an avatar cannot happen. Since players
can only modify their own avatar’s position, and nothing else can affect it, two different servers will never take a paradoxical decision towards any avatar’s position.

Therefore all paradoxes detected are relative to avatar’s life state. However, avatar position is an input parameter to the result of a shot: an avatar cannot be shot if it moves outside the sight range of the shooter. Therefore local lag on moving actions reduces the occurrences of “out-of-sight kill” paradoxes on life state, but cannot reduce the number of “dead target killed” or “dead shooter killing” paradoxes. A conservative local lag value on all shooting actions would however avoid all these paradoxes regardless of the amount of local lag applied on moving actions.

6.4.4 Conclusion on tailoring synchronisation

Based on the observation that in a given game, various action types can affect the violations perceived by players differently, we argue these action types may have different requirements in term of synchronisation. Therefore synchronising them all in the exact same way, disregarding their differences, is probably suboptimal.

We proposed to tailor the synchronisation based on action requirements in order to improve game playability. Our simulations tested the application of different local lag values for moving and shooting actions in NetGameSim. The results confirm that tailoring local lag individually for moving and shooting actions can improve overall users’ experience.

In some situations, the variation in temporal shift between different types of actions may have a negative impact on playability. Hence, all groups of actions may not be suited for individually tailored synchronisation. However, the use of additional techniques, such as delaying animation as described by Armitage et al. (in [6], Chapter 6), can help relieving the temporal shift disparity. In the particular case of moving versus shooting actions, the higher response time for
shooting result could be smoothed by displaying gunshot animation and even blood spill immediately when a player uses his weapon.

We believe, tailoring synchronisation in a generic fashion can work in a large number of situations, and in particular with other synchronisation parameters than Local Lag. In fact, some games already allow avatar movement to be co-simulated on players' terminal. This technique however is iso-consistent\(^1\) as the co-simulation doesn't affect the decision point(s). In [67], Griwodz proposed a game architecture in which messages are classified with urgency and relevance levels and processed accordingly by the network and the servers. However in most modern games, there is probably more than these two parameters to take into account.

6.5 Conclusion

As in distributed event simulations, there is no one synchronisation solution that suits all games. We propose not to restrict the search for the best synchronisation strategy to a binary optimistic or conservative answer but rather toward a continuum of possible solutions between these two extremes. In the next step, we pushed the synchronisation tuning process further by adjusting the synchronisation parameters of different actions type independently.

Results show that the techniques we propose can improve playability in our simulated environment. While it is not possible to generalise this conclusion to all game situations (as no synchronisation system is the best solution for all scenarios) these ideas can work better than traditional techniques in some circumstances and therefore enrich the toolbox available to game designers.

Parts of the contributions of this chapter were presented at the Australian Telecommunication Network and Applications Conference (ATNAC) in November 2004 [24] and selected for publication in 2005 Autumn volume of the

\(^1\text{cf. 5.4.1, p.100}\)
“Telecommunication Journal of Australia” [20]. Selected results were also published in the May 2005 “IEEE Communications Magazine” [130].

After looking into improving the software synchronisation, the next chapter will look into optimising the second aspect of network game infrastructures: the network topology of decision points.
Chapter 7

Decision Point Topology Considerations in Online Games

Synopsis

In Chapter 6 we have shown how tuning and tailoring synchronisation parameters can improve playability and/or fairness.

This chapter will explore possible improvement of user experience via alteration of the other aspect of the network game infrastructure: topology of decision points.

Content:

- Formulation of the decision point placement problems.
- Optimal solution for small networks.
- Approximate solution for large networks using a heuristic.
7.1 Introduction

In Chapter 4, we have seen that the network infrastructure supporting the online game is composed of the set of synchronisation parameters and the topology of decision points. Chapter 5 explained how the imperfections of this infrastructure affect the user experience through reduced playability and fairness. The previous chapter was dedicated to improve playability through software tuning of synchronisation.

As server virtualization and other cloud computing technologies are becoming mainstream, it may become possible to choose and even migrate the location of game servers in an collection of data centres. Hence, the natural next step and focus of this chapter is the exploration of possible improvements of user experience via alteration of the decision point topology.

Technically, the topology of the network game infrastructure include both terminals and decision points. However, it is difficult to imagine changing the positions of players and their terminals in the network. If this was easily done, an obvious solution to network issues in any online game would be to gather all players in the same location and provide a game with no delays. For this reason, we consider in this thesis that only the position of decision points can be altered.

This chapter defines the concept of critical playability and use it as the objective function for the server selection optimisation problem we formulate. The specificity of this objective function is that it makes the system converge towards both a better playability and game fairness. An approximate heuristic solution to this optimisation problem usable for large networks is introduced and its performance compared to a calculable lower bound.

Significant research has been done proposing various protocols and network architectures for game servers. However, only a few studies tackle the selection of game servers amongst potential sites. Lee, Ko and Calo presented in [87] their
"zoom-in zoom-out" heuristic which selects a minimum number of servers satisfying given delay constrains. Their results assumed a 25% reduced inter-server latency compared to client-server delay to emulate well provisioned network paths. Our formulation of the server selection problem tends to optimise an objective function related to the quality of experience of the participants using a fixed number of servers. Moreover, we do not assume any specific advantage for delays of inter server paths.

As we did in 5.5.2, p.110, we propose to only consider conservative synchronisation in order to end up with a meaningful and consistent definition of playability and fairness for decision point topologies. Doing so enables us to collapse all inconsistencies on the action age (or terminal inconsistency) for all players. Playability of participants is then represented by their action age and the overall network game fairness can be estimated by the standard deviation of the terminal inconsistency amongst the players.

If we assume that terminals do not use co-simulation or dead reckoning, the only violation generated by the action age would be response time. With this particular assumption, the terms "inconsistency" and "action age" could be replaced with "response time" in this entire chapter. Doing so would keep the terminology consistent with the literature which doesn’t differentiate inconsistencies from violations. However, using a violation (which happen to be measurable) instead of an inconsistency (which are always measurable in time units) and adding an extra assumption would not be in line with the purpose of our research to stay as generalisable as possible.

Next section will formulate the integer programming problem aiming at selecting the best decision point location in order to optimise players’ game experience. Section 7.3, p.151 introduces the notion of critical inconsistency and propose to use it as the objective function to our server selection problem. It will also demonstrate that a lower bound on the critical inconsistency, which is not always reachable, can always be calculated. In Section 7.4 we solve the server selection problem for three types of small networks and discuss the influence of
different type of topologies on the behaviour of the optimal solution. Section 7.5 finally presents a heuristic solution converging towards a near optimal critical inconsistency. The performance of the heuristic is evaluated in simulations and compared with the calculable critical inconsistency lower bound.

7.2 Decision point placement problem formulation

In a scenario where game servers can be allocated or relocated in various location in the Internet cloud, it becomes possible to optimise the game server selection in order to ensure the best game conditions for players. This section mathematically formulates the integer programming problem which need to be resolved in order to select the optimal server locations in terms of quality of experience of players.

Given a network topology composed of fixed players and a set of possible server sites, this model aims to select the $n$ best decision points which minimise a given objective function. The objective function formulation depends on the desired improvement: minimal average action age, minimal unfairness or minimal critical inconsistency.

7.2.1 General Description

The topology is described by a graph $\Phi(V, E)$ composed of $|V|$ nodes and $|E|$ links joining 2 nodes bidirectionally. $V$ is divided into the subset of player nodes $P$ and the subset of potential server nodes $S$. $V = S \cup P$. $P$ and $S$ are not exclusive sets: a potential server can also be a player. That way, the problem formulation stays valid for all architectures: peer to peer, central and distributed servers. In the particular case when the potential servers are the player’s terminal themselves $V = P = S$. Each link in $E$ has an associated propagation delay and the delay $d_{ij}$ between any two node $(i, j) \in V^2$ is calculated using the
Dijkstra shortest path algorithm.

7.2.2 Problem Formulation

Given Parameters

- Nodes: $V$ the set of all network nodes composed of $P$ the subset of Player nodes and $S$ the subset of possible server site. $V = S \cup P$.
- Propagation delay: $d_{ij} \mid (i,j) \in V^2$ is the delay required by a packet to travel from node $i$ to node $j$ using the shortest possible path over the network.
- Number of decision points: $s$, the number of nodes acting as decision points to serve the terminals. $s \leq |S|$. In central server case, $s = 1$.

Decision Variables

We define the set of decision variable $x_{ij}$ with $(i, j) \in S \times P$ as:

$$x_{ij} \overset{\text{def}}{=} \begin{cases} 
1 & \text{if server } i \text{ serves player } j \\
0 & \text{otherwise} 
\end{cases} \quad \text{(7.1)}$$

Dependent Variables

In order to simplify notations and formulas, we introduce two sets of dependent variables:

- Active server: $y_i$ with $i \in S$ is defined as
\[ y_i = \begin{cases} 
1 & \text{if potential server site } i \text{ is active} \\
0 & \text{otherwise} 
\end{cases} \]  
\[ = \begin{cases} 
1 & \text{if } \sum_{j \in P} x_{ij} \neq 0 \\
0 & \text{otherwise} 
\end{cases} \]  
(7.2)

- Local Lag \( L_i \) introduced in server \( i \) to keep a conservative synchronisation:
\[ L_i = \max_{j \in S} (d_{ij} * y_i * y_j) \]  
(7.3)

**Constraints**

- Number of servers in use:
\[ \sum_{i \in S} y_i = s \]  
(7.4)

- Each terminal connects to only one server:
\[ \forall j \in P : \sum_{i \in S} x_{ij} = 1 \]  
(7.5)

**Objective functions**

With the core of the problem now formulated we now need to express its objective function. In fact, different objective functions can be introduced to suit different optimisation purposes. As our aim is to improve the experience of players, we first need access to various metrics.

The action age of a given player \( j \in P \) is:
\[ AA_j = \sum_{i \in S} x_{ij} * (2 * d_{ij} + L_i) \]  
(7.6)
The average of players’ inconsistency $\bar{AA}$ (as action age) can be calculated:

$$\bar{AA} = \frac{1}{|P|} \sum_{j \in P} AA_j$$  \hspace{1cm} (7.7)

The coefficient of variability of players’ inconsistency $CV(AA)$ is:

$$CV(AA) = \sqrt{\frac{1}{|P|} \sum_{j \in P} \left( \frac{\bar{AA} - AA_j}{\bar{AA}} \right)^2}$$  \hspace{1cm} (7.8)

In order to optimise the average network playability of players in the game, the objective corresponding objective function to minimise is the average amount of inconsistency. As the $|P|$ is not dependent on the decision variable, it is sufficient to minimise the sum of all players’ action age:

$$Obj_{Playability} = \sum_{j \in P} AA_j$$  \hspace{1cm} (7.9)

Optimising the network fairness without artificially degrading the inconsistency level of any player requires to minimise the coefficient of variation of players’ inconsistency:

$$Obj_{Fairness} = \frac{StdDev(AA)}{\bar{AA}}$$  \hspace{1cm} (7.10)

In the particular case of a game requiring perfect fairness, and therefore enforcing the same amount of inconsistency as the worse-off player to all other participants, the objective function to minimise is:

$$Obj_{CriticalPlayability} = Max_{j \in P}(AA_j)$$  \hspace{1cm} (7.11)
7.2.3 Concluding remarks on the formulation

**Complexity and linearisation**

This problem is non-linear in the formulations of the active server dependent variable 7.2 used to constrain the maximum number of servers. Many others non-linearity affect the objective functions through the calculations of the player inconsistency 7.6 and the local lag of servers 7.3. Finally, objection functions may be non-linear themselves.

A linearised version of this problem has been developed by adding numbers of constrained artificial decision variables to remove the multiplication between decision variables in (7.2)(7.3)(7.6) and the Max functions in (7.3)(7.11). It is presented in Appendix E, p.206 and is not used in this chapter because the gain obtained from solving the linearised problem is offset, in this specific situation, by the complexity added by the number of required artificial decision variables.

**Objective functions experimentation**

While experimenting with various server location problems and objective functions, we observed that fairness optimization functions lead most of the time to impractical solutions. Optimising a topology to satisfy a fairness-only objective function results in extremely poor overall playability in the vast majority of cases. The reason is that, if fairness is the sole discriminant, the solver selects a solution inconsiderately from the level of playability. In fact, the solution found is usually so bad that, for the same topology, even the worse-off player (in terms of playability) in the optimal playability solution would have a better level of inconsistencies than the average players in the optimal fairness solution. Hence, using the optimal playability solution and artificially degrading the inconsistency level of all players to match the worse-off one typically provide a better solution than the optimal fairness solution.

Our exploration for an objective function which could satisfy both the playability and the fairness criteria simultaneously led us to study the special situation
of the worse-off player (in terms of inconsistencies) of any game topology configuration. Using the specific characteristic of the worse off player in a given game, the next section propose an objective function aiming at maximising both playability and fairness at the same time.

7.3 Critical Inconsistency

Playability and fairness are the two aspects of user experience depending on the supporting network game infrastructure. Chapter 5, p.88 described the relations between inconsistencies, playability and fairness.

One particular situation of interest is applications requiring perfect fairness amongst the participants. Situations requiring perfect fairness arises when the stakes of a game match are high, like online game competitions. Many of the concept developed for fairness in games can also be applied to other networked real time applications requiring fairness such as stock market trading for example.

As discussed in Chapter 5.5, p.109, a game is perfectly fair only if all participants suffer the same amount of disturbances. That is, true network fairness is achieved if all users have the same level of inconsistencies (action age in a conservative architecture) and perceived fairness is achieved if users are indistinguishably (from a human perception point of view) affected by violations.

Since the variation of playability amongst participants is the source of unfairness, game fairness can be improved, in any topology, by artificially increasing the inconsistency levels of the better-off players towards the playability of the worse-off ones. At the limit, enforcing the response time of the worse-off player to all would achieve perfect game fairness with identical playability amongst all players. Let us call the worse-off participant in terms of playability the critical player. This user endures the game’s critical inconsistency.
We choose critical playability as our objective function because critical inconsistency optimisation can achieve a double goal: first, in a game where perfect fairness is required (enforcing same inconsistency to all players), minimising the critical inconsistency would minimise the inconsistency of all players. Secondly, in a game where fairness is not enforced, the critical player is the one at the highest disadvantage and a major contributor to the overall game unfairness and the average playability. Therefore reducing the action age of the critical player tend to improve both the game’s average playability and global fairness. Simulations in sections 7.4 and 7.5 will show such optimisation converges towards a balanced configuration between playability and fairness. Also, if this critical player found the game playable, by definition all other players would too.

### 7.3.1 The critical path

The critical inconsistency identifies the critical network path. The critical network path is the set of critical links which contribute to the critical inconsistency. Like in a Gantt chart, if the latency on one of the network link of the critical path increases, the critical inconsistency increases as well.
Figure 7.1 illustrates the concepts of critical path, critical links and critical player(s) in central, distributed and peer to peer network architectures. In the case of a central server, the critical path is the route from the server to its furthest player. The critical inconsistency is equal to the return trip time over the critical path.

In a pure peer to peer architecture, the critical path is the path between the two most distant peers and the critical inconsistency is equal to the local lag required to conservatively synchronise these two critical peers, that is, the delay of the critical path.

In a distributed architecture scheme, there is at least one critical player and two critical servers: the critical path is a combination of the path from that critical player to his/her server, which is the first critical server, and the path between the first critical server and the most distant server, which is the second critical server. The critical inconsistency therefore is the sum of twice the delay of the critical player to the first critical server path plus the delay introduced by the distance between the two critical servers.

### 7.3.2 A lower bound to Critical Inconsistency

Let us consider a set $P$ of $|P|$ players interconnected through a telecommunication network $T$. $D(\alpha, \beta)$ the delay introduced by the network when using the shortest path between any two nodes $\alpha$ and $\beta$ being either player terminals or decision points in the network.

**Peer to Peer**

In a full peer to peer scenario with conservative synchronisation and maintaining perfect game fairness, the minimum commonly achievable response time (noted $AA_{P2P}$) is equal to the local lag required to synchronise the two most distant and therefore critical peer players noted $P_A$ and $P_B$: 
\[ AA_{P2P} = \max_{(P_i, P_j) \in P_2^2} D(P_i, P_j) = D(P_A, P_B) \]  

(7.12)

**Central Server**

Let us imagine a reduced game where only \( P_A \) and \( P_B \) are playing. If this game was controlled by a single central server (still maintaining perfect fairness), optimal response time performances is achieved if the server is situated in the exact middle of the shortest path between the two most distant players in the game. We refer to this particular server position as \( \text{Mid}_{AB} \) and note \( AA_{CS_{Red}}(ServerLocation) \) the critical inconsistency of a central server configuration in the reduced game. The action age of both critical players \( P_A \) and \( P_B \) would be:

\[
AA_{CS_{Red}}(Mid_{AB}) = 2 \times D(Mid_{AB}, P_A) \\
= 2 \times D(Mid_{AB}, P_B) \\
= 2 \times \frac{1}{2} \times D(P_A, P_B) \\
= AA_{P2P} \tag{7.13}
\]

If the server is positioned anywhere else but in \( Mid_{AB} \), the critical path will be longer than \( D(Mid_{AB}, P_A) \) and therefore:

\[
AA_{CS_{Red}}(Anywhere) \geq AA_{CS_{Red}}(Mid_{AB}) \tag{7.14}
\]

In these conditions, regardless of other players involved in the game, the action age of \( P_A \) and \( P_B \) when connected to a central server in \( Mid_{AB} \) establishes a lower bound of the critical inconsistency of any central server. This lower bound is also equal to the critical inconsistency of the peer to peer configuration.

In the game with all the players, if another player becomes critical when the
central server is positioned in $Mid_{AB}$ then the lower bound is unreachable with a central server. Therefore the critical inconsistency of a central server in the game with all players located anywhere is bounded:

$$AAP_{2P} = AAC_{CS_{Reduced}}(Mid_{AB}) \leq AAC_{CS}(Anywhere) \leq AAC_{CS}$$

(7.15)

**Distributed Servers**

Using a similar process, we can generalise this result and show that $AAP_{2P}$ also bound the action age of a distributed server architecture: in the same reduced game with only $P_A$ and $P_B$ involved, the best possible response time would be achieved if the two servers $S_A$ and $S_B$ respectively serving $P_A$ and $P_B$ were both located somewhere on the shortest path from $P_A$ to $P_B$ and if $D(S_A, P_A) = D(S_B, P_B)$. In this situation, both $P_A$ and $P_B$ would remain critical and the critical inconsistency would be:

$$AAP_{DS_{Reduced}}(S_A, S_B) = 2 \cdot D(P_A, S_A) + D(S_A, S_B) = 2 \cdot D(P_B, S_B) + D(S_A, S_B) = D(S_A, P_A) + D(S_A, S_B) + D(S_B, P_B) = D(P_A, P_B) = AAP_{P_{2P}}$$

(7.16)

If the distributed servers are not located in that specific way or, in the game with all the players, if another player becomes critical, $P_A$ or $P_B$ or both of them will lose their critical properties and therefore the new critical inconsistency imposed to everyone can only be higher than $AAP_{DS_{Reduced}}(S_A, S_B)$. Therefore:
\[ AA_{P2P} = AA_{DSRed}(S_A, S_B) \]
\[ \leq AA_{DS}(AnyNumberOfServersAnywhere) \]
\[ \leq AA_{DS} \] (7.17)

\( AA_{P2P} \) is therefore a lower bound to any form of conservative and perfectly fair network game architecture. On one hand, conservative synchronisation requires information to travel between the different entities, the delays involved in these exchanges limit the performance in terms of action age. On the other hand, enforcing perfect fairness imposes a common action age amongst all players. Thus, it actually makes sense that the highest delay between any two entities requiring synchronisation bound the performance of the whole system.

### 7.4 Server selection in small networks

The next logical step after the formulation of the optimisation problem is to attempt to solve it. However, at the time of writing, there is no effective way to solve the problem formulated. In order to give some insight into the behaviours of the solution we opted to solve small instances of the problem by doing an exhaustive search of the solution space.

#### 7.4.1 Simulations setup

We developed a game topology evaluation tool which can quickly compute the critical inconsistency of a given servers topology and its synchronisation parameters (local lags). A second routine feeds the evaluation tool with all the combinations possible given a fixed number of servers \( s \) and a game topology composed of a set of \( |P| \) players and \( |S| \) potential server sites as described in the introductory paragraph of this section.
To allow the simulation to systematically reach the lower bound on critical inconsistency $AAP_{P2P}$ if the maximum number of servers $s$ allows it, player nodes are always considered as potential server sites ($P \in S$).

### Complexity

In effect, an exhaustive search is done on all possible server combinations for the optimal results to be extracted. We have, at the time of writing, no other known way to exactly solve this optimisation problem.

The number of assessments $A$ required to find the configuration minimising the objective function depends only on the number of potential server sites $|S|$ and the imposed number of servers required $s$ and is equal to:

$$A = C_{s}^{S}$$  \hspace{1cm} (7.18)

Numerical example: searching for 10 servers within 20 potential sites requires 184756 assessments. If each operation last 0.01 seconds, the overall computation time is around 30 minutes.

The resolution of the decision point placement problem by exhaustive search can only be achieved for small networks composed of less than 25 potential server sites.

### 7.4.2 Simulations results

The optimum critical inconsistency server selection problem is solved for three types of topologies with different level of organisation. One pure random and one transit-stub series of networks generated using the gt-itm[26] software, and one regular hierarchical topology manually created.

All networks are composed of either 21 or 22 nodes which are all potential servers, 12 of them being players as well, which have been selected preferably
at the edge of the network. In the case of the random and the transit-stub networks, the optimal critical inconsistency of 100 topologies have been averaged while only one hierarchical network was solved. The random networks created by gt-itm had between 29 and 34 bidirectional links (26.7 Average, 2.77 Std-Dev), the transit stub networks had between 20 and 27 links (22.5 Average, 1.41 StdDev).

Because of differences in organisation between random, transit stub and balanced tree networks, comparison of their raw optimal critical inconsistency has no real meaning: Even if their network diameter are made equal (providing them with the same critical P2P action age) optimal critical inconsistency for low number of servers end up in different ranges, with the random networks providing extremely high critical inconsistencies compared to the other two types. Therefore, the trends of the reduction are the focus this section. Figure 7.3 presents the evolution of the optimal critical inconsistency (normalised between 0 and 1) as a function of the number of servers for comparison of the different trends.
Figure 7.3: Optimum critical inconsistency vs number of servers. Only the trend in the variation of critical inconsistency are comparable as result have been normalised.
7.4.3 Discussion

The presented balanced tree topology (Figure 7.2) can be divided into four distinct layers playing an important role in the optimal server location. Four collections of sweet spots of server location combination distinctly appear in Figure 7.3 when selecting 1, 3, 6 and 12 servers. These in fact correspond to the selection of each of the topology’s distinct four layers. Because of the extreme regularity of this topology, all of these solutions are in fact optimal since they reach the lower bound defined as the critical inconsistency in the peer to peer scenario in Section 2.

Pure random networks do not show any minima in the evolution of the critical inconsistency when the number of servers is increased. While the trend of individual random networks is not necessarily monotonic decreasing like the average shows, there is no visible correlation amongst occasional individual minima.

After averaging, the evolution of the transit stub networks show a minimum in the 3-5 servers region. Study of server positions solution on individual network reveals that in many cases, but not all, there is a sweet spot in these transit stub networks consisting of 3 to 5 servers located in or directly attached to the transit domain. These particular sweet spots do not however provide a critical inconsistency as good as the peer to peer solution.

The reason why the sweet spots found provide unusual critical inconsistency is because they tend to satisfy three criteria more closely than the other configurations:

- Players are all approximately at the same distance from their decision points.  
  \[(7.19)\]
- Decision points are all approximately at the same distance from each other.  
  \[(7.20)\]
• Decision points are located close to the direct communication path between players.

(7.21)

In fact, when all these criteria are truly satisfied (as opposed to approximately satisfied), all players and decision points are simultaneously critical and the optimal solution is reached. In the balanced tree topology, this ideal situation happens in four different configurations (corresponding to the selection of each nodes in the four distinct layers).

In more realistic topologies, such perfect sweet spots are improbable. However, one can suspect some configurations could satisfy these criteria better than others, specially since the Internet loosely follows the transit-stub network model.

In the case where player nodes cannot be used as server sites (breaking the \( P \in S \) assumption used in this simulation) it is unlikely that any server configuration can reach the lower bound of the critical inconsistency \( AA_{P2P} \) (as defined in 7.3.2, p.153) because no solution is likely to fully satisfy criteria 7.21. Furthermore, if none of the potential server sites are close (in a network sense) to the players, then any configuration will strongly dissatisfy the criteria 7.21 and therefore will result in critical inconsistency far from the optimum lower bound.

7.4.4 Conclusions on optimisation of small networks

The resolution of the decision point placement problem in small networks reveal the possible presence of sweet spots which are collections of server locations providing good critical inconsistency. Our results suggest that the presence of these sweet spots is associated to the level of regularity and hierarchical structure in network topology. This may have important implications for the optimal location of decision points for online games in real networks, as they
are neither fully random, nor perfectly organised.

7.5 Server selection in large networks

In order to solve the decision point placement problem for large network we have developed an iterative heuristic called “minimum critical inconsistency growth” which converges towards a set of servers with close to optimal critical inconsistency.

Based on knowledge acquired in sections 7.3 7.4 on the properties of criticality, two principles drove the conception of this heuristic:

- Optimisation effort should go towards critical player(s) and critical servers as they are the ones responsible for the critical inconsistency.
- Convergence should be aimed towards possible sweet spots.

The main idea of the algorithm is to select a set of servers with similar distance from the players at the edge of the network. To that aim, it starts from a partial solution of two servers minimising the response time of the two most distant players. The heuristic then improves the solution step by step by adding new servers in order to optimise the response time of the worse-off player. This process ends when the addition of a new server to the existing solution does not improve the situation.

7.5.1 Heuristic description

**Given parameters:** a network topology with a set of players, a set of potential server sites and a maximum bound to the local lag of any server, $MaxLL$, which defines the maximum distance between any two servers.

**Step 1: Initialisation** Find the two most distant players in the network. These would be the critical players in a peer to peer scenario. Choose the two
best servers (respecting the local lag constraint $MaxLL$) which would provide the best response time to these critical players if they were to play alone in a restricted game.

**Step 2: Generalisation** Generalise the game to all players with the present server solution. If the generalised solution does not unveil a new critical player, the current solution cannot be improved by adding a new server and the heuristic ends. Otherwise go to step 3.

**Step 3: Expansion** Add the new critical player to the restricted game’s participant list and find the server which minimises the new critical inconsistency when added to the current solution while satisfying the maximum local lag constraints. If no server can be added, the current solution cannot be improved further and the heuristic ends. Otherwise, go back to Step 2.

The maximum local lag constraint limits the spread of the selected servers: the lower the maximum local lag, the smaller the solution in terms of number of servers. At the limit, a maximum local lag of zero will end up with a central server and probably a poor critical inconsistency performance. On the other hand, an infinite value for $MaxLL$ allows the heuristic to converge toward the peer to peer solution with as many servers as players, reaching the absolute lower bound in terms of critical inconsistency.

However, as shown in Section 7.4, more servers does not always mean better critical inconsistency. Therefore $MaxLL$ should be tuned to converge towards a set of "sweet spots". Since the heuristic is not computationally expensive, it is possible to try several values of the maximum local lag constraint.

### 7.5.2 Performance simulations

**Simulation setup**

This heuristic was run 100 times over a simulated Internet like network topology, generated by gt-itm, composed of 600 nodes with 48 randomly positioned
Figure 7.4: Iterative evolution of a typical heuristic convergence. In each step, the heuristic add an extra server to improve the critical inconsistency. Once additional server do not improve the system, the heuristic stops.
players. The diameter (maximum distance between any two nodes) of this network is around 450ms. We allowed a maximum local lag of $\text{MaxLL} = 250\text{ms}$ to converge towards a relatively small number of server without sacrificing performances too much.

**Heuristic convergence**

On average, the gap between the final solution and the absolute lower bound was found to be 5.19% (2.95 standard deviation) and the average number of servers in the solution required for the heuristic to stop is 7.56 (2.23 standard deviation).

Figure 7.4 shows one representative instance of the iterative evolution of the heuristic solution in terms of critical inconsistency compared to the lower bound and the critical inconsistency of two other selection strategies:

- Best central server in term of average response time which purely optimises the overall game playability.
- Best central server in term of critical inconsistency which finds a balance between playability and fairness.

As can be seen, the critical inconsistency of the heuristic, even for only 6 distributed servers is very close to the lower bound and is significantly better than even the best central server solution. In other words, a properly designed distributed architecture is likely to outperform any central server model in terms of critical inconsistency under a wide range of conditions.

**Performance and comparison with central server strategies**

Figure 7.5 presents the performances of the three server topology selection strategies in the same simulated network. The horizontal and vertical axes represent the level of playability and fairness respectively; the closer to the origin the better. Each of the 100 simulations of three selection strategies
Figure 7.5: Heuristic performance using network playability. Action age (i.e., inconsistency) is used as the playability metric. Fairness is measured as the CV of playability metric (action age).

Figure 7.6: Average and standard deviation of inconsistencies (action age). For the same set of simulations Figure 7.5, the fairness axis is replaced by the standard deviation of action age.
generated solutions that are represented as a single point on the figure. The combination of these points creates distinct clouds. The fourth cloud, labelled "Average Central Server", represents the expected playability and fairness of a randomly chosen central server for comparison.

**Perceived playability as metric**

In a second set of simulations presented in Figure 7.8, we substituted our regular network playability metric by an evaluation of the perceived playability of an RPG game. The utility function representing the perceptual impact of action age was based on the empirical study of Fritsch *et al.* [55] stating that group combat in Everquest2 are affected from a delay of 200-500ms and are uncoordinated at 1250ms. The heuristic was modified to optimise critical perceived playability instead of critical inconsistency. Naturally, the fairness metric considered is changed the coefficient of variation of the perceived playability. Algorithms search for optimal central servers have also been modified accordingly to maximise perceived playability and critical perceived playability.

**Discussion**

In Figure 7.5, the central server solutions chosen for optimal playability provide consistently low inconsistency with a high level of unfairness. The outcome of the best critical inconsistency central server location is more variable: solutions are spread and sometimes mix in the cloud of best playability servers and other times with provide a better fairness with an inferior playability. The distributed solution from the heuristic is consistently better than the other two strategies in terms of fairness at the cost of a slight increase in average inconsistency compared to the optimal playability central servers.

It can be observed that, the fairness of randomly chosen central servers is better than the fairness in optimal playability central servers. Although, both server groups present similar deviation in their inconsistencies (as shown in Figure 7.6, random central servers present significantly higher level of inconsistencies,
Figure 7.7: Utility function of playability for Role Playing Games (RPGs)

Figure 7.8: Heuristic performance using perceived playability. The utility of action age (i.e., perceived playability) is used as the playability metric. Fairness is measured as the CV of playability metric.
hence their unfairness which is the relative spread of inconsistencies relative to its average is smaller.

With the RPG playability metric shown in Figures 7.7, the heuristic outperforms all forms of optimal central server selection both in terms of playability and fairness.

7.6 Conclusion

This chapter introduced the notion of critical inconsistency and showed how it could be used as an objective function to optimise game user experience. Also the critical inconsistency has been shown to be bounded, in any conservative architecture, by the delay between the most two distant players.

We formulated the optimal critical inconsistency server selection problem. The resolution of the decision point placement problem in small networks reveals the possible presence of “sweet spots”: collection of server locations providing better critical inconsistency. Our results suggest that the presence of these sweet spots is associated to the level of regularity and hierarchical structure in network topology which may have important implications in the optimal location of decision points for online games.

An iterative heuristic providing an approximate solution to the server location problem in large networks was introduced. Its performance in a 600 nodes network was compared to the calculable critical inconsistency lower bound and two central server selection strategies. In our simulations, the heuristic converges towards a close to optimum critical inconsistency. It also outperforms the central server selection strategies in terms of fairness. When optimising the critical perceived playability through a utility function, the heuristic provides better solution than the central server selection strategies in terms playability as well.
Parts of the contributions of this chapter were published and presented at the ACM workshop on Network and system support for games (NetGames) in November 2006 [23]. Selected results were also published in the November 2006 issue of “Communications of the ACM” [22].
Chapter 8

Conclusions and future work

Synopsis

This last chapter conclude the thesis.

Content:

- Summary of contributions
- Future work
8.1 Overview

Geographical distances introduce unavoidable delays in telecommunications network. These network latencies deteriorate the quality of players’ experience in online games. For that reason, this thesis had attempted to analyse the processes leading from network delay to the loss of game quality and proposed novel solutions to improve the experience of users in online games. This final chapter will summarise the main contributions and findings of our research and discuss the opportunity for future work in this area.

8.2 Summary of contributions and findings

The first part of this thesis focused on developing the framework required to understand, model and analyse how latency originating from the telecommunication network impact the participants. In the second part of the dissertation, we used the knowledge previously acquired to propose novel techniques aimed at improving the overall experience of online game participants.

8.2.1 A framework for the analysis of network game disturbances

Our first contribution is a clear model for the propagation of the disturbances which emerge from the imperfections of the telecommunication network and deteriorate the playability of a game. In this model, three separate stages lead from network delays to the final perception of players:

1. The delays introduced by the network generate inconsistencies on game terminal and decision points. The amount and type of inconsistencies present on each node (terminal or decision point) also depends on the synchronisation scheme used by the game. Inconsistencies can always be objectively measured in time units. As an example, a decision point in a
1. A distributed architecture may bear some "event age" inconsistency which represents the amount of time this decision point will not be aware of a game state changing event which happened on another decision point.

2. The presence of inconsistencies generate violations of the ideal laws of the virtual world. The type of violation created by each inconsistency also depends on the game logic. Unlike inconsistencies, violations can be more difficult to measure and their units are variable. An example of violation on a decision point caused by its "event age" is the probability that, because of a late event update, the decision point need to do a rollback to re-execute events in a proper causal order. Such a rollback, may introduce a more or less noticeable game discrepancy to players connected to this decision point.

3. Violations are perceived by the players through their human perception filters. To quantify the impact we used fuzzy logic tools to model player's perception of single and combined violations.

There is no commonly accepted definition for playability in the literature and this concept is often simplified to Response Time (RT) or Round Trip Time (RTT) which do not count for some violations such as the probability of rollbacks. Our model of disturbances leads to a clear definition of measurable network playability as the collection of all inconsistencies endured by a player. This network playability is always measurable and capture all the disturbances originating from the network delays.

Besides playability, fairness is the other factor significantly contributing to the level of enjoyment of players. In fact, an network game may be playable to a participant but not enjoyable if other competitors have significantly less network impairment. Fairness is a function of the relative playability amongst participants. As we defined a measurable metric for network playability using inconsistencies, we derived a definition of network fairness as the standard deviation of the playability amongst participants. This definition of fairness was
validated against an experimental measure of fairness using score in a game simulation.

### 8.2.2 Improving the network infrastructure supporting online games

The network infrastructure supporting an online game is composed of two elements:

- The topology of the telecommunication network connecting the player’s terminals and their decision points.

- The synchronisation scheme and its parameters used by the game software running on terminal and decision points.

Our research lead us to propose improvements in both the network topology and the synchronisation scheme used by online games.

The analysis of our model of disturbances revealed the potential to manipulate (shift or swap) violations and inconsistencies in some circumstances. In this new light, conservative and fully optimistic synchronisation schemes emerged as the two extremes in a continuum of possible synchronisation strategies. Instead of a binary optimistic or conservative choice, our first enhancement consists in searching for (adequate) optimum synchronisation (in terms of game playability) within this continuous space of trade-offs. As a further improvement, we propose not to bind all actions to the same synchronisation fate but rather tailor synchronisation parameters independently and accordingly to the specific requirements of each type of action. Inside our simulated network game environment, both these techniques presented improvements in game playability.

Although the location of players in the telecommunication is a provided parameter, server virtualization and cloud computing technologies allow us to imagine that in the future the location of game servers could be optimised for
each game session. We introduced the concept of critical inconsistency and demonstrated that the search for server sites optimising this criteria would tend to converge simultaneously towards better playability and fairness. We proved that a lower bound on the critical inconsistency can always be calculated but is not always reachable.

Finally, this thesis formulated an integer programming problem aiming for the optimisation of critical inconsistency when deciding the location of decision points from a set of potential sites. The formulation presents multiple non-linearities and appeared intractable for instances of the problem with more than about 25 potential sites. The problem was solved for small random, hierarchical and transit-stub types of small networks. The analysis of the optimal solutions in hierarchical and transit-stub networks revealed the existence of specific network regions in which optimal servers tends to be located. In order to solve the server location problem for large instances, a heuristic called "minimum critical inconsistency growth" was introduced. In a 600 nodes network, this heuristic tends to converge closely towards the calculable critical inconsistency lower bound. When optimising the critical perceived playability, the heuristic tends to provide better playability and fairness than any central server solutions.

8.3 Future work

The output of this thesis opens the door to new research topics in the field. Here a list of investigation we leave to future work:

Relaxing the absence of jitter and packet loss assumptions The research in this thesis assumed no jitter or packet loss. Although jitter can always be compensated by using properly dimensioned jitter buffers, such buffers increase the effective latency of the system. Our work could be extended to include a model of the effect caused by jitter and our-of-order or loss of information. With this knowledge it would be possible for example
Conclusions and future work

to determine the minimum amount of jitter buffer required before the discrepancies linked to fast delay variations and out-of-order packets outweigh the benefits of a reduced latency.

Research of accurate utility functions  In this thesis we approximated utility functions as semi-trapezoidal curves to represent the impact of specific violation on players. More accurate and realistic utility functions could be derived from additional empirical studies. In particular, there is a lack of data on the perception of unfairness and violations such as probability of rollbacks. Likewise, we could not determine the conjunction operator to be used to estimate the combined impairment generated by multiple violation.

Extending our discrepancies model  This thesis has analysed the inconsistencies present the major synchronisation strategies. In future research, our framework of inconsistencies and violations could be extended to include variations over main synchronisation models such as specific implementations of loose synchronisation.

Dynamic game infrastructure adaptation  Over time, conditions impacting the gaming experience of players may evolve. Such changes may come from the telecommunication network as fluctuation in congestion levels or Internet routes. Players themselves are a source of variations as they join and leave game sessions or modify their behaviour, changing the levels of interactivity. In order to adapt to these fluctuating conditions, it is necessary to develop tools and techniques to detect and measure these changes.

Dynamic server migration  Additional research is required to enable smooth addition, retirement of migration of decision point processes in a running game session.
Bibliography

Printed Sources


[48] Stefano Ferretti, Claudio E. Palazzi, and Marco Roccetti. “An Optimistic obsolescence-based approach to event synchronization for massively multiplayer online games”. In: International Journal of Computers and Applications (2007), was this really published???


Online Sources


Appendix A

NetGameSim tables of simulations results

Tables A.1 and A.2 respectively contain the raw data from which the bilinear interpolation in Figures 6.10 p. 138 and 6.11 p. 138 have been calculated.

From top to bottom, local lag for Moving actions increases by steps of 20ms; from left to right, it is the Local Lag for Shooting actions which increases by steps of 20ms. The maximum simulated local lag is 100ms which is the value at which the synchronisation becomes fully conservative: the maximum delay between any two servers. Table A.1 shows the variation of the average number of Rollbacks while Table A.2 shows the variation of the average response time.
### Table A.1
Average number of Rollbacks when varying the Shooting and Moving local lags

<table>
<thead>
<tr>
<th>Moving Local Lag</th>
<th>0ms</th>
<th>20ms</th>
<th>40ms</th>
<th>60ms</th>
<th>80ms</th>
<th>100ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0ms</td>
<td>9.67</td>
<td>5.3</td>
<td>2.26</td>
<td>1.2</td>
<td>0.62</td>
<td>0</td>
</tr>
<tr>
<td>20ms</td>
<td>8.46</td>
<td>3.82</td>
<td>1.11</td>
<td>0.31</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40ms</td>
<td>7.3</td>
<td>2.83</td>
<td>0.49</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60ms</td>
<td>6.32</td>
<td>2.51</td>
<td>0.26</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>80ms</td>
<td>6.23</td>
<td>2.34</td>
<td>0.29</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100ms</td>
<td>5.91</td>
<td>2.32</td>
<td>0.33</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table A.2
Average Response Time (in ms) when varying the Shooting and Moving Local Lags

<table>
<thead>
<tr>
<th>Moving Local Lag</th>
<th>0ms</th>
<th>20ms</th>
<th>40ms</th>
<th>60ms</th>
<th>80ms</th>
<th>100ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0ms</td>
<td>60</td>
<td>64.18</td>
<td>68.69</td>
<td>73.32</td>
<td>78.17</td>
<td>83.47</td>
</tr>
<tr>
<td>20ms</td>
<td>75.81</td>
<td>80</td>
<td>84.34</td>
<td>88.81</td>
<td>93.34</td>
<td>98.72</td>
</tr>
<tr>
<td>40ms</td>
<td>91.57</td>
<td>95.67</td>
<td>100</td>
<td>104.44</td>
<td>109.12</td>
<td>114.59</td>
</tr>
<tr>
<td>60ms</td>
<td>107.11</td>
<td>110.90</td>
<td>115.24</td>
<td>120</td>
<td>124.86</td>
<td>129.64</td>
</tr>
<tr>
<td>80ms</td>
<td>122.60</td>
<td>126.68</td>
<td>131.07</td>
<td>135.24</td>
<td>140</td>
<td>145.12</td>
</tr>
<tr>
<td>100ms</td>
<td>139.00</td>
<td>142.30</td>
<td>146.51</td>
<td>151.11</td>
<td>155.25</td>
<td>160</td>
</tr>
</tbody>
</table>
Appendix B

More about inconsistencies

B.1 Mathematical Sequence formulation of correct synchronisation

Game state sequence formulation

The evolution of the game state can be mathematically formulated as a recursive sequence of vectors.

\(a_n\) is the nth action of the game session and depends on the participants and the network infrastructure. \(GS_n\) is the game state vector resulting the execution of action \(a_n\). \(Exec(a, GS)\) is the function computing the game state variation resulting from action \(a\) on the game state \(GS\). This function applies all the game rules, including some element of randomness sometimes including. By definition \(Exec(0, GS) = 0\) \(GS_0\) is the initial game state.

\[
GS_n = GS_{n-1} + Exec(a_n, GS_{n-1}) \\
= GS_0 + \sum_{i=1}^{n} Exec(a_i, GS_{i-1})
\] (B.1)
Relationship between continuous time and discreet action index number

$T(a)$ is the function returning the time at which action $a$ is received by the imaginary perfect server which receive all actions instantly. $T_s(a)$ is the function returning the time at which action $a$ is received by server $s$.

\[ GS(t) = GS_n \mid T(a_n) \leq t \text{ and } T(a_{n+1}) > t \]  \hspace{1cm} (B.2)

$GS_n^s$: Game state vector after $n$th action ($A_n$) execution on Decision Point $s$

$GS_n$: Referential Game state vector after $n$th action ($A_n$) execution (on the imaginary perfect decision point receiving all actions in time)

In theory, all actions arrive in order:

**Game state error on a decision point**

$GS_n^s$: Game state vector after $n$th action ($A_n$) execution on Decision Point $s$

$GS_n$: Referential Game state vector after $n$th action ($A_n$) execution (on the imaginary perfect decision point receiving all actions in time) $A_n^s$: the $n$th action as know by server $s$

\[ A_n^s = \begin{cases} A_n & \text{if action } i \text{ was received by server } s \\ \emptyset & \text{otherwise} \end{cases} \]  \hspace{1cm} (B.3)

We assume action $p$ is the last to have been received by server $s$. All actions after $p$ have not arrived on server $s$ and are $\overrightarrow{0}$

\[ p = i \mid (A_1 = A_1^s, ..., A_i = A_i^s, A_{i+1} \neq A_{i+1}^s, ..., A_n \neq A_n^s) \]  \hspace{1cm} (B.4)

The game state error on server $s$ is:

\[ E_n^s = GS_n^s - GS_n \]  \hspace{1cm} (B.5)
And can be calculated:

\[
GS_n^\xi - GS_n = GS_0^\xi + \sum_{i=1...p} \text{Exec}(A_i^\xi, GS_{i-1}^\xi) + \sum_{i=p+1...n} \text{Exec}(A_i^\xi, GS_{i-1}^\xi)
- GS_0 - \sum_{i=1...p} \text{Exec}(A_i, GS_{i-1}) + \sum_{i=p+1...n} \text{Exec}(A_i, GS_{i-1})
= GS_p^\xi + \sum_{i=p+1...n} \text{Exec}(A_i^\xi, GS_{i-1}^\xi) - GS_p - \sum_{i=p+1...n} \text{Exec}(A_i, GS_{i-1})
= \sum_{i=p+1...n} \text{Exec}(A_i^\xi, GS_{i-1}^\xi) - \sum_{i=p+1...n} \text{Exec}(A_i, GS_{i-1})
\]

(B.6)

**Condition of rollback**

\[
\text{Exec}(A_i, GS(T(A_{i-1}))) \neq \text{Exec}(A_i, GS^\xi(T(A_{i-1})))
\]

(B.7)

**Error in the optimistic supposition**

**The optimistic supposition**

\[
\forall i > p : \text{Exec}(a_i^\xi, GS^\xi(T(a_{i-1}))) = \text{Exec}(a_i^\xi, GS^\xi(T(a_{i-1})))
\]

(B.8)

**GM error on a decision point**

\[
GS^\xi(t) - GS^\xi(t) = \sum_{i=p+1...n} \text{Exec}(A_i^\xi(t), GS_{i-1}^\xi(t)) - \sum_{i=p+1...n} \text{Exec}(A_i^\xi, GS_{i-1}^\xi)
= \sum_{i|d_i(t) = 0 \text{ and } T(A_i)} \text{Exec}(A_i^\xi(t), GS_{i-1}^\xi(t))
\]

(B.9)

The evolution of the game state can be mathematically formulated as a recur-
sive sequence of vectors:

\[
GS_n = GS_{n-1} + \text{Exec}(A_n, GS_{n-1})
\]

\[
= GS_0 + \sum_{i=1}^{n} \text{Exec}(A_i, GS_{i-1})
\]  \hspace{1cm} (B.10)

Where \(A_n\) is the \(n\)th action of the game session and depends on the participants and the network infrastructure. \(GS_n\) is the game state vector resulting the execution of \(A_n\). \(\text{Exec}(A, GS)\) is the function computing the game state variation resulting from action \(A\) on the game state \(GS\). This function applies all the game rules, including some element of randomness sometimes including. \(GS_0\) is the initial game state.
Appendix C

More about Utility functions

C.1 Mathematical consideration of game utility

Utility and utility functions enable us to understand the relationships between inconsistencies, violations type and their perceptual impacts on players.

C.1.1 Definition of a utility function

Since these functions are purely subjective, their shapes depend on a multitude of variables which include game type, terminal capabilities and even individuals.

Let us define a utility function $U_V$ as a function connecting a level $x \in \mathbb{R}^+$ of a violation type $V$ to a level of playability $\in [0, 1]$. “0” being a nil and useless playability level and “1” a level of perfect playability.

$$U_V : x \in \mathbb{R}^+ \to U_V(x) \in [0, 1] \quad (C.1)$$

Even without knowing the exact shape of utility functions representing various impacts on players, we can assume some of their properties:

- $\lim_{x \to y} U_V(x) = U_V(y)$ Continuity: The perception of a violation level by
More about Utility functions

Players is continuous.

- $\forall (x, y) \in \mathbb{R}^2 | x < y : U_V(x) \leq U_V(y)$ **Monotonic decrease:** The higher a disturbance level, the higher its impact on players.

- $U_V(0) = 1$: The absence of violation does not reduce players’ comfort.

The utility of a violation $v$ of type $V$ and level $x$ is a scalar:

$$Utility(v) = U_V(x) = U_v$$  \hspace{1cm} (C.2)

Utility functions $U_V$ and $U_W$ of different violation types $V$ and $W$ may be very different but will both satisfy the above properties.

### C.1.2 Composing utilities

When two violations affect the game, their impacts cumulate. The level of comfort can only decrease and would not end up higher than the one of the worst violation alone. In particular if one violation makes the game un-playable (utility close to 0) the presence of other disturbances cannot improve the situation.

Let us define “$\otimes$” as the composition binary operator of the utilities of two violations: $v$ and $w$ two violations of respective type $V$, $W$ with respective levels $x$ and $y$. The composition of the two utilities is equal to the total utility generated by the combination of both violations:

$$U_{v \& w} = U_{V \& W}(x, y)$$

$$= U_V(x) \otimes U_W(y)$$

$$= U_v \otimes U_w$$

This $\otimes$ operator must have the following property $\forall (a, b, c) \in [0, 1]^3$:
• Any violation with a utility of 1 does not affect the game: 1 is the identity element

\[ a \otimes 1 = a \]

• Any violation with a utility of 0 reduces the utility of the whole game to 0: 0 is the invariant element

\[ a \otimes 0 = 0 \]

• The operation is associative:

\[ a \otimes (b \otimes c) = (a \otimes b) \otimes c \]

• The operation is commutative:

\[ a \otimes b = b \otimes a \]

• The composition of two utilities is at least as bad as the worst one:

\[ a \otimes b \leq \text{Min}(a, b) \]

In algebraic terms, \([0, 1], \otimes\) is a commutative monoid.

Under these assumptions, we propose two ways to compose utilities which respect all above criteria. The first one C.3 is the most optimistic of both and assumes that only the worst violation affect the player. The second proposition C.4 is more pessimistic and cumulate the handicap on both violations.

\[ a \otimes b = \text{Min}(a, b) \quad \text{(C.3)} \]

\[ a \otimes b = a \ast b \quad \text{(C.4)} \]

Finally, both propositions can be unified in the following definition of the utility
composition operator which include a parameter \( \alpha \in [0, 1] \). If \( \alpha = 0 \) the composition is defined as \[ \text{C.3} \] and if \( \alpha = 1 \) it is defined as \[ \text{C.4} \]

\[
a \otimes b = \alpha (a \ast b) + (1 - \alpha) \text{Min}(a, b) \tag{C.5}
\]

### C.1.3 Inconsistencies and total game utility

Violations are created by inconsistencies. Therefore the utility of an inconsistency is the composition of the utilities of all the violations it creates. \( I \) an inconsistency present in a game, \( I_V \) the set of violations the set of violations generated by this inconsistency

\[
\text{Utility}(I) = \bigotimes_{v \in I_V} \text{Utility}(v)
\]

\[
= \bigotimes_{v \in I_V} U_v \tag{C.6}
\]

\[
= U_I
\]

The total utility of a game to a given player is the composition of the utilities of all inconsistencies, which is the same as the composition of the violations present:

\[
\text{Utility}(\text{Game}) = \bigotimes_{I \in G_i} \text{Utility}(I)
\]

\[
= \bigotimes_{I \in G_i} \bigotimes_{v \in I_V} \text{Utility}(v) \tag{C.7}
\]

\[
= \bigotimes_{I \in G_i} \bigotimes_{v \in I_V} U_v
\]

\[
= U_{\text{Game}}
\]
Appendix D

Cancelling a game’s initial conditions

D.1 Initial game state conditions and fairness

D.1.0.1 On the importance of initial conditions

Initial conditions of a game session is the game state at the beginning of the game. It can include a seed used for events randomisation during the game. The subset of the initial game state fully describing a given avatar is the avatar’s initial conditions and may include the avatar’s initial position and health level. The other subset of the initial game state is the rest of the universe initial conditions. Such state includes the layout of the virtual universe (often called map in FPS) along with the positions and states of all non avatar entities.

In a game session with $N$ avatars $A_1$ to $A_N$, the initial conditions $IC$ is composed as following:

$$IC = IC(A_1) \cup IC(A_2) \cup ... \cup IC(A_N) \cup IC(\text{rest})$$

Initial conditions may provide advantages or disadvantages to some players. For

\footnote{Avatar’s conditions may not be limited to this examples of game state.}
example an avatar which is spawning close to a strategic item may benefit of an early lead over the other players. These potential unbalances have implications on the game fairness which are not network related.

Let us consider $N$ synthetics players, with perfectly identical and invariable reaction patterns, playing an online game which randomness is controlled by an initial seed. Given a set of $N$ initial conditions for the $N$ identical players, the unfolding of all $N!$ possible game sessions, from the permutations of the players' initial conditions, in a perfectly fair game infrastructure should make each player go through the same set of game state trajectories, in different order.

Therefore, in these specific situation it is possible to detect if a game was fair by comparing the game state trajectories of all the players, simply because playing all the permutations cancels out any advantage/disadvantage associated with particular initial condition.
Appendix E

Linearisation of decision point problem

E.1 Introduction

This appendix describes a linearised version of the binary integer programming problem described in [7.2] p[146]

E.1.1 Given Parameters

Exactly as described in [7.2] p[146]

- Nodes: $V$ the set of all network nodes composed of $P$ the subset of Player nodes and $S$ the subset of possible server site. $V = S \cup P$.

- Propagation delay: $d_{ij}$ | $(i, j) \in V^2$ is the delay required by a packet to travel from node $i$ to node $j$ using the shortest possible path over the network.

- Number of decision points: $s$, the number of potential servers to act as decision points to serve the terminals. $s \leq |S|$. In central server case, $s = 1$. 

206
E.1.2 Decision Variables

This version of the problem formulation uses the set of original decision variables $x_{ij}$ with $(i, j) \in S \times P$ as:

$$x_{ij} \overset{\text{def}}{=} \begin{cases} 1 & \text{if server } i \text{ serves player } j \\ 0 & \text{otherwise} \end{cases} \quad (E.1)$$

The original set of dependent variable $y_i$ is transformed into a second set of decision variables to transform their original polynomial (and hence non-linear) definition into a linear constraint (defined in $E.7$):

Active server selector: $y_i$ with $i \in S$ is defined as

$$y_i = \begin{cases} 1 & \text{if potential server site } i \text{ is active} \\ 0 & \text{otherwise} \end{cases} \quad (E.2)$$

A last set of extra decision variables need to be created to linearise the original local lag constraint containing a $Max$ operator (done in $E.8$ via the dependent variable set $L_i$). They are defined as follows:

$z_{ij}$ with $(i, j) \in S^2$ as:

$$z_{ij} \overset{\text{def}}{=} \begin{cases} 1 & \text{if servers } i \text{ and } j \text{ are active and } j \text{ is the furthest active server from } i \\ 0 & \text{otherwise} \end{cases} \quad (E.3)$$
E.1.3 Dependent Variables

As in the original problem, we introduce a set of linearly dependent variables to define the local lag:

Local Lag $L_i$ introduced in server $i \in S$ to keep a conservative synchronisation:

$$L_i = \sum_{j \in S} z_{ij} \cdot d_{ij}$$  \hspace{1cm} (E.4)

E.1.4 Constraints

- As in the original problem, the number of servers in use:

$$\sum_{i \in S} y_i = s$$  \hspace{1cm} (E.5)

- Also from the original problem, each terminal connects to only one server:

$$\forall j \in P : \sum_{i \in S} x_{ij} = 1$$  \hspace{1cm} (E.6)

- Constraining the active server selector $y_j$ of server $j \in S$ to be 1 if any client is connected to it and 0 otherwise:

$$\forall i \in S : y_j \leq \sum_{i \in P} x_{ij}$$  \hspace{1cm} (E.7)

- Constraining the local lag $L_i$ on active servers $i \in S$ to the maximum distance to any other active server:

$$\forall i \in S : L_i \geq \sum_{j \in S} y_j \cdot d_{ij}$$  \hspace{1cm} (E.8)
E.1.5 Objective functions and resolution

Although this problem is now linear, the number of decision variables has been significantly increased. Where the original problem had $||P|| \times ||S||$ decision variables, this version requires $2 \times ||P|| \times ||S|| + ||S||$ decision variables.

Furthermore, the definition of response time as described in [722] p.146 has a quadratic form:

$$ AA_j = \sum_{i \in S} x_{ij} \times (2 \times d_{ij} + L_i) $$ (E.9)

Since all objective functions of interest are a function of the players’ response time, extra linearisation steps (including extra decision variables) would be required to fully linearise any of our objective functions.

As an experiment, we used the commercial ILOG CPLEX quadratic optimiser to solve the re-formulated problem with the objective of minimising the average response times of players:

$$ \overline{AA} = \frac{1}{|P|} \sum_{j \in P} AA_j $$ (E.10)

The resolution time of CPLEX for this problem was not faster than parsing the entire solution space of the original formulation.