

École Doctorale
d'Informatique,
Télécommunications
et Électronique de Paris

Thèse

présentée pour obtenir le grade de docteur

de TELECOM ParisTech

Spécialité : Informatique et réseaux

Matteo VARVELLO

Une Architecture de Communication Pair-á-Pair pour les Environnements Virtuels en Ligne

Soutenue le 7 Decembre 2009 devant le jury composé de

Prof. Daniel Kofman	**Président**
Prof. Jim Kurose	**Rapporteur**
Prof. Geoff Voelker	**Rapporteur**
Dr. Marc Shapiro	**Examinateur**
Dr. Christophe Diot	**Examinateur**
Prof. Ernst Biersack	**Directeur de thèse**

Copyright ©2009 by Matteo Varvello (varvello@eurecom.fr).
Typeset in LaTeX.

Abstract

A Networked Virtual Environment (NVE) is a synthetic world where human-controlled *avatars* can interact. Multiplayer on-line games such as Quake and World of Warcraft are the most popular applications for NVEs. In early 2003, Second Life (SL), a NVE where avatars can invent a new social life, was launched. The main innovative feature of SL is user-generated content: avatars participate in the development of the virtual environment by creating *objects* such as cars, trees, and buildings. SL rapidly became the most popular NVE, reaching more than 16 million registered users in September 2009.

The state of the art for NVEs design is a Client/Server architecture where multiple servers maintain the state of the virtual world and distribute it to the users. This architecture is very expensive as large amount of servers need to be deployed, operated and maintained. Moreover, scalability is an issue. These drawbacks motivate alternative designs such as Peer-to-Peer (P2P). Ideally, a P2P virtual world can scale with the number of its users as each user dedicates some of its resources (storage, CPU, bandwidth) to the management of the virtual world. Moreover, P2P can dramatically cut server and network cost for the virtual world provider.

The contribution of the thesis is threefold. First, due to the lack of publicly available data about NVEs such as avatar movement patterns or object distribution, we perform an extensive analysis of SL. We deploy a *crawler* and a *player* application and monitor objects, avatars, user Quality of Experience and servers performance in the public part of SL over one month.

Second, we design and build a *distributed object management* for user-generated NVEs. We first integrate this distributed object management on the top of KAD, the P2P network that supports millions of eMule users, and perform large-scale experiments. Then, we propose *Walkad*, a structured P2P network designed to manage user-generated objects in P2P-based NVEs.

Third, we investigate the feasibility of a distributed avatar management using the *Delaunay triangulation*. To start with, we evaluate the performance of the Delaunay triangulation via realistic experiments performed in SL using a modified client we developed. Then, we design two optimizations for Delaunay triangulation: (1) a clustering algorithm to efficiently handle large avatar groups, and (2) a secured extension to the Delaunay triangulation that leverages the social component of NVEs.

Résumé

Un *Environnement Virtuel en Ligne* (NVE) est un monde synthétique où les utilisateurs peuvent intéragir à travers des *avatars*. Les jeux en ligne comme "Quake" et "World of Warcraft" sont les applications les plus populaires pour les NVEs. Au début de l'année 2003, est apparu Second Life (SL), un monde virtuel où les avatars peuvent créer une nouvelle vie sociale. La principale caractéristique innovatrice de SL est le contenu créé par les utilisateurs : les avatars participent au développement de l'environnement virtuel par la création d'*objets* tels que des voitures, des arbres et des bâtiments. SL est devenu rapidement le monde virtuel le plus populaire, avec plus de 16 millions d'utilisateurs en Septembre 2009.

L'état de l'art pour la conception des NVEs est principalement une architecture Client/Serveur où plusieurs serveurs maintiennent l'état du monde virtuel pour le transférer aux utilisateurs. Cette architecture est très chère puisqu'une grande quantité de serveurs doivent être déployés, exploités et maintenus. Par ailleurs, cette architecture ne passe pas à l'échelle. Ces inconvénients motivent la conception des architectures innovatrices tels que l'architecture Pair-à-Pair (P2P). Idéalement, une architecture P2P pour les NVEs passe à l'échelle, car chaque utilisateur consacre une partie de ses ressources (stockage, CPU, bande passante) à la gestion du monde virtuel. En outre, le P2P peut considérablement réduire le coût pour le fournisseur du monde virtuel.

Il y a trois contributions majeures dans cette thèse. Premièrement, en raison du manque de statistiques sur les mondes virtuels (par exemple, mouvement des avatars et distribution des objets), nous procèdons à une analyse exhaustive de SL. Pour cette raison, on a implémenté deux applications qui surveillent, pendant un mois, objets, avatars, Qualité d'Expérience des utilisateurs et performance des serveurs dans la partie publique de SL.

Deuxièmement, nous concevons une *gestion distribuée des objets* pour les NVEs générées par ses utilisateurs. Nous intégrons cette infrastructure de communication sur KAD, le réseau P2P utilisé par des millions d'utilisateurs d'eMule, et réalisons des

expériences à grande échelle. Ensuite, nous concevons *Walkad* un réseau P2P structuré spécialement conçu pour gérer les objets générés par les utilisateurs dans un monde virtuel.

Troisièmement, nous étudions la faisabilité d'une *gestion distribuée des avatars* en utilisant la *triangulation de Delaunay*. D'abord, nous évaluons la performance de la triangulation de Delaunay par des expériences réalisées dans SL grâce à un client modifié que nous avons développé. Successivement, nous concevons et évaluons deux optimisations pour la triangulation de Delaunay : (1) un algorithme de clustering qui améliore la *réactivité* des interactions entre les avatars en présence de larges groupes d'avatars ; (2) le *réseau Delaunay social*, un réseau P2P qui résout le problème de la sécurité dans un NVE en utilisant les relations d'amitié qui existent entre les avatars.

Acknowledgements

This is the first time in my life I have the opportunity to write some acknowledgements. Frankly speaking, I did not think it was so hard to come up with a concrete list of names and thoughts. I hope I did not forget too many people.

Thanks to Chiara, who was there during my entire PhD iter spanning Sophia Antipolis, Paris and San Diego. We shared the joy of the first paper accepted, the sufferance of a Sigcomm submission as well as its easily predictable rejection. As a couple, we grow a lot during these three years, and a few days after I will be a Doctor, you will be my wife. Your support, from creating avatars for the Second Life crawler to accept travelling across the world, has been the most precious thing I had in these years.

Thanks to my parents for supporting all my own decisions even when these drove me being far from home. I owe you everything I have, and I will be always grateful to you for the opportunities you gave me. Thanks to all my friends. A special thank to Alessandro, Daniele and Diego for the great time (and food and wine) spent together respectively in Sophia Antipolis, Turin and Paris discussing our common PhD frustrations. Thanks also to Vincent Macaud, the most "italianized" of my french friends. We had great time together watching and playing football, precious time I am certainly going to miss.

This thesis was part of a collaboration involving Thomson and Eurecom, i.e., respectively north and south of France as well as industry versus academia. While it was hard to move across France several times, this experience gave me the possibility to discover an amazing country, so close to Italy and so different. At the same time, I was exposed to researchers and professors working in different domains. The time spent discussing and brainstorming with all of them was precious to drive my PhD and future research.

This project gave me the opportunity also to have two PhD advisors, Ernst Biersack and Christophe Diot. They are very different – from a social and work standpoint – but both told me how to do top-level research. Thanks to the both of you for the

time, advise and discussions that taught me how to be critical with my own work. Thanks also for allowing me to spend three months at UCSD in San Diego. This was a great life experience as well as a great opportunity to meet an outstanding research group. A special thank to Geoff Voelker for hosting me at UCSD. Working with you has been a pleasure.

Thanks to all my colleagues in Thomson (Boulogne). I am most grateful to Fabio Picconi for his friendship, support (especially late night) and technical help. Thanks to Laurent Massoulie and Stratis Ioannidis who helped me to formalize many of the problems I had to solve. Thanks to Augustin Chaintreau and Augustin Soule for precious discussions and time. Thanks also to Abderrahmen Mtibaa for reviewing the French summary of this thesis. Thanks to Anna-Kaisa, Dan-Cristian, Henrik, Italo, Marianna and Theodoros who shared with me great research discussions and most importantly a lot of fun. A special thank to Fernando Silveira for the Sigcomm/Infocom/Imc campaigns conducted together in the Thomson lab, I will never forget those nights. Thanks to Venus Apovo and Vincent Alloy for their precious administration and system management help. Last but not least, thanks to my intern Stefano Ferrari who has deployed the Peer-to-Peer Second Life client. I had a great time working with you and supervising your Master thesis.

Thanks to all my colleagues in Eurecom (Sophia Antipolis). I am most grateful to Fabio Pianese and his insightful discussions when, as a young PhD student, I was trying to understand how to do research. Thanks to Moritz Steiner, a great friend who helped me a lot with the implementation of my system over KAD. Thanks to Taoufik En-Najjary for sharing most of our lunch-breaks and for initiating me to the beauty of Matlab. Thanks to Pietro Michiardi who suggested me to look into Second Life. Thanks to Marco Balduzzi, Daniele Croce, Leucio Antonio Cutillo, Alessandro Duminuco, Marco Paleari, Abdullatif Shikfa, Slim Trabelsi and all the other PhD students.

Contents

List of Figures xi

List of Tables xv

1 Introduction 1
- 1.1 Networked Virtual Environments . . . 1
- 1.2 The Future of Networked Virtual Environments . . . 4
- 1.3 Architectures for Networked Virtual Environments . . . 5
- 1.4 Thesis Organization . . . 6
- 1.5 Thesis Contributions . . . 7

2 Related Work 9
- 2.1 Networked Virtual Environments . . . 9
- 2.2 Peer-to-Peer . . . 11
- 2.3 P2P Networked Virtual Environments . . . 12
- 2.4 Second Life . . . 14
- 2.5 On Line Social Networks . . . 15
- 2.6 Range Queries . . . 15
- 2.7 Delaunay Network . . . 17

3 Exploring Second Life 19
- 3.1 Introduction . . . 19
- 3.2 Second Life . . . 20
- 3.3 Methodology . . . 23
- 3.4 A Global View at Second Life . . . 29
- 3.5 In-Depth Analysis of the Most Popular Regions . . . 38
- 3.6 Second Life as a Social Network Service . . . 43
- 3.7 Playing Second Life . . . 49
- 3.8 Conclusions . . . 57

4 Distributed Object Management **59**
4.1 Introduction . . . 59
4.2 Background . . . 61
4.3 A Simple Approach . . . 64
4.4 Walkad . . . 80
4.5 Conclusions . . . 91

5 Distributed Avatar Management **95**
5.1 Introduction . . . 95
5.2 The Delaunay Network . . . 96
5.3 A Distributed Avatar Management for Second Life . . . 99
5.4 Dynamic Clustering . . . 104
5.5 Social Delaunay Network . . . 115
5.6 Conclusions . . . 119

6 Conclusions **121**
6.1 Summary . . . 121
6.2 Outlook . . . 123
6.3 What is Missing? . . . 125

Bibliography **127**

Appendices **133**

A List of Abbreviations **137**

B Synthèse en français **139**
B.1 A la Decouverte de Second Life . . . 139
B.2 Gestion Distribuée des Objets . . . 145
B.3 Gestion Distribuée des Avatars . . . 150

List of Publications **155**

List of Figures

3.1 Architecture of the crawler. . . . 24
3.2 Architecture of the player. . . . 25
3.3 Crawling performance [Statistics subcrawler ; 18 hrs experience]. . . . 27
3.4 Completeness of the Avatar subcrawler [Avatar subcrawler ; Map subcrawler ; 12 hrs experience]. . . . 28
3.5 CDF of region availability [Statistics subcrawler]. . . . 31
3.6 Active population over time [Map subcrawler ; SL Login Server ; Coordinated Universal Time - 5]. . . . 32
3.7 CCDF of the object distribution across regions ; [Object subcrawler]. . . . 33
3.8 Percentiles of the variation of the no. of objects per region ; [Object subcrawler]. . . . 34
3.9 CCDF of several percentiles of the population CDFs ; [Statistics subcrawler]. . . . 35
3.10 CDF of virtual group sizes [Map subcrawler]. . . . 36
3.11 CDF of spot lifetimes for different average group size S [Map subcrawler]. . . . 37
3.12 Population and server load ; [Stat/Map/Av subcrawler]. . . . 39
3.13 No. of avatars versus server load ; [Stat/Map/Av subcrawler]. . . . 41
3.14 CDF of user session times ; [Stat/Map/Av subcrawler]. . . . 42
3.15 Avatar movement patterns ; [Stat/Map/Av subcrawler]. . . . 42
3.16 CDF of avatar visits to the same virtual group ; [Stat/Map/Av subcrawler]. . . . 43
3.17 Power law analysis of the node degree over time for α, x_{min}, D. . . . 47
3.18 Node degree (log scale) vs. clustering coefficient ; $t = 256$ hrs. . . . 48
3.19 Percentiles of the betweenness centrality distribution over time. . . . 49
3.20 Percentiles of the avatar inconsistency over time. . . . 54
3.21 CDFs of the fraction of time an avatar is inconsistent. . . . 55
3.22 CDF of the duration of inconsistencies. . . . 56
3.23 CDFs of the discovery latency. . . . 57

4.1 Kademlia k-bucket organization ; k=2. . . . 62

4.2 Adaptive cell-based partition ; $D_{max} = 3$. 65
4.3 Cell-IDs organization. 66
4.4 Region consistency analysis ; $R = 20$ 73
4.5 CCDF of region consistency ; $R = [20; 15; 10; 5]$; $AoI_r = 35\ units$. 74
4.6 CDF of the recovery latency ; $R = 20$; $AoI_r = 35\ units$. 75
4.7 Persistency analysis ; $R = [20; 15; 10; 5]$. 77
4.8 CCDF of region consistency ; $R = 20$; $AoI_r = 35\ units$; $N = [586; 112; 82; 29]$. 79
4.9 CDF of the recovery latency ; $R = 20$; $AoI_r = 35\ units$; $N = [586; 112; 82; 29]$. 80
4.10 Example of the Gray Code construction. 81
4.11 1-dimensional virtual world indexed by Walkad. 82
4.12 Uniform cell division of a 2-dimensional virtual world. 85
4.13 Skewed cell division of a 2-dimensional virtual world. 86
4.14 $D_{max} = [10; 30; 60; 100]$; $N = 1024$; Avatar=$[Walk]$. 88
4.15 $D_{max} = 10$; $N = [16 - 1024]$; Avatar=$[Walk; Run; Fly; Teleport]$. 89
4.16 $D_{max} = 10$; $N = 1024$; $N_a = [10; 20; 40; 80; 100]\%$; Avatar=$[Walk]$. . . . 90
4.17 Some percentiles of the distribution of the fraction of cell-IDs per peer ; five Second Life regions ; $D_{max} = 10$; $N = [16 - 1024]$. 90
4.18 Some percentiles of the distribution of the fraction of cell-IDs per peer ; $D_{max} = 10$; $N = 1024$; No. of regions=$[1 - 100]$. 91

5.1 Distributed avatar management leveraging the Delaunay Network. . . 97
5.2 Evolution of a Delaunay triangulation via flip operations. 98
5.3 The P2P-SL client design. 100
5.4 Evolution over time of the no. of connected avatars during our experiment. 102
5.5 CDF of the inconsistency. 102
5.6 CDF of the fraction of time an avatar AoI is inconsistent. 103
5.7 CDF of inconsistency duration. 104
5.8 Delauany triangulation generated by introducing $d1$ and $d2$. 107
5.9 Example of clustering in a Delaunay Network. 110
5.10 Maintenance cost C_m ; Public Help Island (SL) ; $N = 89$; $T = 30\ mins$. 113
5.11 CDF of propagation hops ; $B_t = 18\ kbps$; $AoI_r = 35\ units$. 114
5.12 Several percentiles of the distribution of the propagation-set sizes over time ; $AoI_r = 35\ units$; $B_t = 18\ kbps$. 115
5.13 Ratio of P2P traffic and total traffic ; $W = [0, 1, 3, 10]\%$; $AoI_r = [5; 35]\ units$. 117
5.14 CDF of propagation hops ; $W = 3\%$; $AoI_r = 35\ units$. 118

5.15 CDF of peer-set sizes ;$W = 3\%$; $AoI_r = 35\ units$. 119

B.1 Architecture du crawler. 140
B.2 Architecture du player. 141
B.3 Population active au cours du temps [Map subcrawler; SL Login Server ; Temps universel coordonné - 5]. 142
B.4 Centiles de la distribution de la variation de nombre d'objets par région ; [Object subcrawler]. 143
B.5 CDFs de la fraction du temps que l'AoI d'un avatar est incohérent. . . 144
B.6 CDFs de la durée d'incohérence. 144
B.7 CDFs de la latence de découverte. 145
B.8 Monde virtuel avec une seule dimension indexée par Walkad. 147
B.9 $D_{max} = 10$; $N = [16 - 1024]$; Avatar=$[Walk; Run; Fly; Teleport]$. 149
B.10 Centiles de la distribution de la fraction des cell-IDs par pair ; cinq régions Second Life ; $D_{max} = 10$; $N = [16 - 1024]$. 150
B.11 Triangulation de Delaunay entre les avatars d'un monde virtuel. 152
B.12 Le client P2P-SL. 153
B.13 CDF de l'incohérence. 154
B.14 CDF de la durée de l'incohérence. 154

List of Tables

1.1 Comparison of MOGs requirements. 4

3.1 Second Life crawling summary. 30
3.2 Number of regions in SL (RS=Region subcrawler, SLW=Second Life website). 30
3.3 Comparison of the SL social network with several real life networks and on-line social networks. 45

4.1 Causes of inconsistency. 76
4.2 Cell configuration. 79

5.1 Table of parameters. 105

A.1 List of Abbreviations 137

B.1 Synthèse du crawling de Second Life. 141
B.2 Nombre de régions dans SL (SR = Region subcrawler, SLW=site internet de Second Life). 142

CHAPTER 1
Introduction

"...and you will see, if you look, a low wall built along the way, like the screen which marionette players have in front of them, over which they show the puppets."

– Plato –

1.1 Networked Virtual Environments

Plato introduces the concept of *virtual reality* in the Republic, a dialogue written approximately in 380 BC, through the famous allegory of *the cave*. For Plato, human beings live in a form of virtual reality that we are deceived into thinking is true [43].

Technology realizes Plato's allegory of the cave through *virtual environments*. A virtual environment is a computer-simulated environment that reproduces either a real or an imaginary world. The virtual environment can be enriched by *objects* such as cars, trees, and buildings. Users access a virtual environment through their digital representation called *avatar*. Avatars are controlled using keyboard and mouse, while the environment simulation is displayed on a screen.

Avatars and objects are very different entities. Avatars are characterized by a small state defined by simple information such as the avatar position and appearance. Avatars are generally dynamic entities as they allow their users to explore the virtual world. Moreover, avatars are non-persistent entities: when a user disconnects from a virtual world, its avatar leaves too. Objects are generally described by a rich state that may include textures, music and video. They are generally static, i.e., they re-

side in the place where they are created, but some of them may move, e.g., cars and bullets. With few exceptions such as potions or food, objects are mostly persistent.

Virtual environments rapidly evolve into *Networked Virtual Environments* (NVEs). NVEs are Internet-based virtual environments accessible to multiple concurrent users located world-wide. NVEs were originally introduced by the SIMNET project [18] in order to interconnect several war simulators. Successively, NVEs were applied to *Multi-player Online Games* (MOGs), video games that aim at supporting thousands or even millions of concurrent players. Below, we describe the three main MOGs genres [41] that are of interest for the understanding of the thesis.

- *Role-Playing Games (RPGs)* are MOGs where users assume the roles of fictional characters [50]. RPG users behave accordingly to a formal system of rules and actions defined by the character they are playing. The main goal of RPG players is increase their character abilities. In most RPGs there is a game master, either human or computer controlled, that chooses the setting in which players play. The decisions of the game master as well as participants improvisation define the outcome of the game. Currently, the most popular RPG is World of Warcraft [115] counting more than 15 Millions subscribers, i.e., players who pay monthly fees in order to take part to the virtual world. Other very popular RPGs are Everquest [30], Final Fantasy [32] and Ultima Online [97].

- *First-Person Shooters (FPSs)* are MOGs centered on weapon-based combat where the player experiences the game through the eyes of its avatar. The main goal of FPS players is to kill the maximum number of opponents. Some of the most popular FPS games are Wolfenstein [114], Doom [26], Half-Life [37], and more recently Quake [77] and Halo [38].

- *Social Virtual Worlds (SVWs)* are recent MOGs where users invent and emulate a virtual social life. Other innovative features of SVWs are world-building and virtual economy. World-building is the possibility to participate in the deployment and modification of the virtual environment through the creation of *user-generated* objects. A virtual economy is the possibility to buy and sell objects, services and lands. Second Life [83] (SL), launched in 2003 by Linden Lab, has become the most popular SVW, reaching more than 16 million registered users in September 2009. Other very popular SVWs are Active Worlds [3], Kaneva [51], There [96], PS3 Home [75] and The Sims Online [95].

Different applications running on top of a NVE may have different requirements. However, there are some fundamental requirements that are common to most NVE-based applications, namely:

- **Consistency**: In order to have meaningful user experiences in a NVE, each avatar has to perceive the same state of the virtual world, e.g., land and avatars appearance. To do so, the NVE architecture needs to ensure an almost simultaneous distribution of the virtual world state to the active users. However, a number of issues such as delayed traffic on the Internet or unpredictability of the user demand make consistency in NVEs a very challenging task.

- **Responsiveness**: User experience in virtual worlds is positive when avatars perceive quickly enough changes in the virtual world state [22], e.g., neighbor avatar movements or the creation of a new object. Thus, avatars need to be informed in real-time about any action performed as well as its consequence. Different NVE-based applications have different responsiveness requirements, e.g., $300\ ms$ for FPSs and $1\ sec$ for RPGs [22].

- **Scalability**: NVEs aims at interconnecting millions of users within a shared environment, i.e., NVEs should ideally scale to an unlimited number of participants. Precisely, a NVE architecture is required to sustain a theoretically unbounded growth in the user demand with a sub-linear impact on the user experience.

- **Persistency**: Contents within a NVE may need to exist persistently. For instance, information about a user score as well as user-generated objects should not be lost during the evolution of the virtual world.

- **Security**: In computer science, security is the prevention of risk or danger such as the exposure, destruction or alteration of personal information [72]. Applications running on top of a NVE have created complex virtual economies as well as strong competitions among users. Therefore, a NVE architecture needs to guarantee security in order to ensure fairness among its users as well as privacy of user information. Nevertheless, cheating has recently emerged as a major security concern in NVEs. NVE users cheat in several ways, e.g., delaying packet transmission or modifying the user client. The reader can found in [116] a detailed classification of cheating in NVEs.

Table 1.1 compares the requirements of FPSs, RPGs and SVWs. All three MOGs genres require an high level of *consistency* and *security*. *Scalability* is fundamental for RPGs and SVWs where the level of fun is determined also by the number of concurrent users, e.g., World of Warcraft recently claimed to have reached peaks of one million concurrent players [115]. Conversely, typical game sessions of current FPSs involve only up to a few tens of players, and too crowded sessions make the game hard to play. For this reason, scalability is not a strong requirement for FPSs. *Persis-*

	RPGs	FPSs	SVWs
Consistency	high	high	high
Responsiveness	medium	high	low
Scalability	high	low	high
Persistency	low	low	high
Security	high	high	high

Table 1.1: Comparison of MOGs requirements.

tency is an important requirement for SVWs where the virtual world is mostly user-generated. On the contrary, the virtual world object composition of both RPGs and FPSs is pre-located at the clients, e.g., via CD distribution or BitTorrent like download [115]. Thus, persistency is a negligible requirement for RPGs and FPSs. Finally, *responsiveness* is not an issue in SVWs where users mostly interact by chat and barely move. RPGs require an higher level of responsiveness since avatars perform more interactive actions, e.g., combats or competitions. Instead, fast paced avatar interactions characterize FPSs that so require high level of responsiveness.

1.2 The Future of Networked Virtual Environments

Future NVEs are expected to become more complex and realistic, somehow competitive with real life.[1] Improvements in computational power, three dimensional rendering, and available bandwidth may make the vision come true in the near future.

NVEs could also be run on a variety of devices, e.g., not only computers or TVs, but also on Personal Digital Assistants and mobile phones. They may be build on top of infrastructure-less networks such as Delay Tolerant Networks [53] where users are either connected to closeby devices or they have only intermittent Internet connectivity. For example, NVEs running on infrastructure-less communication architectures seem a promising platform for "mixed reality games" [110]. These are MOGs where the physical location of a player drives its avatar location in the virtual world.

Our vision of a future NVE is a *user-generated* virtual environment where avatars can have *fast paced* interactions. Such a NVE would allow a tremendous evolution in MOGs: we can imagine a FPS game, e.g., Quake, where avatars interact in a user-generated environment like Second Life. Thus, future NVEs have much stronger requirements than common NVEs, and need to leverage an architecture that provides

[1] http://www.wired.com/gamelife/2008/02/ray-kurzweil-lo/

high level of Consistency, Responsiveness, Scalability, Persistency and Security. Nevertheless, an architecture for future NVEs must be flexible enough to be run over a dedicated infrastructure as well as over infrastructure-less environments.

We identify in Second Life (SL) the first step towards future NVEs. In fact, SL is much more than a simple virtual world. Indeed, SL is a *virtual reality-based Internet* in the spirit of the *metaverse* described in "Snow Crash" [93]. In the thesis, we make a major effort to analyze SL. The rational is that studying user behavior in SL as well as the dynamics of the virtual world evolution, e.g., user-generated content, is crucial in order to drive the architectural design of future NVEs.

1.3 Architectures for Networked Virtual Environments

Today, most NVEs are implemented using a Client/Server (C/S) architecture. The virtual world is divided into independent lands each managed by a dedicated server. The tasks of a NVE server are: **avatar management** and **object management**. The avatar management consists in updating each avatar about the status of its neighbor avatars in real time. The object management consists in maintaining the user-created objects over time, and informing each avatar about the objects in their visibility area. The role of the client in a C/S NVE is to simply capture user input, e.g., avatar movements or the creation of a new object, and send this information to the server.

The advantages of a C/S architecture for NVEs are multiple. C/S architectures are naturally *consistent* and *persistent* when communication is reliable. Moreover, *security* is enforced as all significant information, e.g.,user scores and accounts, are maintained by trusted authorities, i.e., the servers. However, a C/S architecture for NVEs has also many drawbacks:

- Servers have to be deployed, operated and maintained.
- *Scalability*: each server can only handle a finite number of users, so the cluster of servers must be dimensioned according to the expected peak demand.
- The server is a single point of failure, and may limit service availability.
- C/S is not optimal from a delay point of view since all the communications are relayed by the server. Consequently, the *responsiveness* of the NVE may be reduced.

These drawbacks, the high cost of a centralized solution, and players frustration due to low performance are the motivations to investigate alternative designs. Specifically, Peer-to-Peer (P2P) solutions where the virtual world is maintained leveraging user own resources seem to be a promising alternative.

The key point for the feasibility of P2P NVEs is *locality of interest*. Avatars within a NVE are only interested in a small portion of the virtual world called **Area-Of-Interest**. For example, an avatar requires only to see the objects that are located in its surroundings or it only interacts with closeby avatars. This means that each peer needs only to be connected to the peers whose avatars are located closeby or that are responsible for closeby objects. Conversely, without locality of interest each peer would need to broadcast the data it manages to all peers whose avatars are active in the NVE.

P2P for NVEs have several advantages. (1) If we make the assumption that each user joins the NVE with enough spare resources to maintain its share of the virtual world, *scalability* comes for free. (2) A P2P architecture for NVEs significantly reduces the required investment compared to C/S. In fact, there is no need to deploy a large amount of resources, and all players can benefit from the shared resources (storage, CPU and bandwidth). (3) P2P is more robust than C/S as there is no single point of failure. (4) P2P allows direct communication among peers, i.e., an optimal design to achieve *responsiveness*. Aside these practical advantages of P2P, future NVEs may require to be based on P2P technologies, e.g., infrastructure-less mixed reality games.

The goal of this thesis is to design a P2P architecture for future NVEs. This is a very challenging task since the architecture needs to deal with the different requirements and characteristics of objects and avatars. For this reason, we choose a *divide and conquer* approach: we conceive two overlay designs dedicated respectively to a **distributed object management** and a **distributed avatar management**.

1.4 Thesis Organization

The organization of the thesis is driven by a style of research that consists in understanding experimentally the behavior of an existing large scale system, before to propose "yet-another" distributed architecture. The information gathered during this understanding phase drives the system design. A quick but realistic implementation of the system allows to evaluate the feasibility of the design. Successive iterations on system design and implementation allow to make it more complete and realistic.

We took Second Life as the most representative example of future NVEs. In Chapter 3, we conduct a large study of SL. This study allows us to identify the main design principles of a P2P NVE while collecting useful avatar and object traces for the evaluation of our P2P architecture. Chapter 4 addresses the problem of a distributed object management for NVEs. We propose a simple but realistic communication infrastructure that allows us to perform large scale experiments over the Internet. Then, we design and evaluate a P2P network dedicated to object management in NVEs. Chapter 5 focuses on the problem of a distributed avatar management. We first conduct an experimental evaluation of a simple distributed avatar management that we integrate in the SL client. Then, we design and evaluate several enhancements to this simple distributed avatar management that allow to overcome the weaknesses identified by the previous experimental evaluation. Finally, Chapter 6 gives a summary of the work presented in the thesis and discusses future work and open issues.

1.5 Thesis Contributions

This thesis makes several contributions:

- We perform a large analysis of SL. To do so, we deploy a **crawler** and a **player** application that gather traces about objects, avatars, user Quality of Experience and server statistics in the public portion of SL.

- We integrate a distributed object management for NVEs over KAD [29], currently the most popular P2P network. We exploit KAD as an experimental platform to evaluate how SL would perform over a standard P2P network already deployed over the Internet.

- We design and deploy **Walkad**, an extension to Kademlia [63] that ensures fast responsiveness for object lookups in user-generated NVEs. Walkad exploits the predictability of avatar movement patterns to improve the distributed storage and retrieval of virtual objects.

- We deploy a **P2P-SL client** that allows end-users to directly manage the propagation of their avatar state updates in SL. To do so, the client relies on the *Delaunay Network* [69][9], a very popular design for distributed avatar management. We use the P2P-SL client in order to compare a centralized versus a distributed avatar management for SL.

- We analyze the strengths and weaknesses of a distributed avatar management constructed on top of the Delaunay Network. Based on this analysis, we design

a **clustering** algorithm that improves user Quality of Experience in Delaunay-based NVEs.

- We design the **Social Delaunay Network**, a distributed avatar management that enforces security in a NVE leveraging a network of virtual friendships.

CHAPTER 2

Related Work

This Chapter surveys the subjects of Peer-to-Peer (P2P) and Networked Virtual Environments (NVEs) in a broad perspective. Section 2.1 describes the most popular applications for NVEs. Section 2.2 describes the P2P designs related to this thesis or that have been an inspiration for our design. Section 2.3 focuses on the most popular P2P architectures for NVEs. Section 2.4 presents the studies conducted on Second Life (SL). Section 2.5 summarizes the previous research conducted on on-line social networks. Section 2.6 presents the most popular solutions to handle range-queries over Distributed Has Tables (DHTs). Section 2.7 discusses the pieces of work that are relevant to the Delaunay Network.

2.1 Networked Virtual Environments

Military Simulation Military simulation was the original application for NVEs. Simulator Networking or SIMNET [18] was a research project of early 80s that aims at networking war simulators located world-wide in real-time. SIMNET describes the virtual world as a collection of *objects*, e.g., vehicles and weapons, that interact through *events*, e.g., "tank t destroys building b". The SIMNET architecture is completely distributed. Each war simulator continually broadcasts to all other war simulators the objects and events it manages. The rationale of this design choice is that the architecture has to be robust, with no single point of failure. SIMNET introduces also *dead reckoning* [7], a mechanism to predict the movement of a remote object given its previous position and velocity.

The protocol used in SIMNET was successively formalized and standardized by IEEE with the Distributed Interactive Simulation (DIS) standard [76]. DIS extends

SIMNET by introducing the possibility to simulate more military tasks, e.g., training operators of tanks, aircraft and ships, and to use multiple types of vehicles in the same simulation.

Recently, military simulation research evolved into the High Level Architecture (HLA) [46] IEEE standard. HLA does not specify any particular technology, but simply provides a general architecture for distributed simulation. Thus, HLA can be used for military simulations as well as for on-line games.

Multiplayer On-Line Games Multi-User Dungeon (MUD) [11] created in 1978 by Roy Trubshaw and Richard Bartle is considered the first Multiplayer On-line Game (MOG). Precisely, MUD is a text-based MOG that can be classified as a Role Playing Game (RPG) (cf. Chapter 1). The virtual world of MUD was stored into a database running on a DECsystem-10 computer [25] located at the University of Essex. Multiple players could access MUD simultaneously and interact with other users as well as with the virtual world. The objective of MUD players was to accumulate skill points by collecting objects and treasures, or by killing other players. Interestingly, players could also augment the environment by adding content into the database using a programming language.

In 1993, id Software released Doom [26], the first multi-player First Person Shooter (FPS) (cf. Chapter 1) game for personal computers. Doom multi-player mode leverages a simple P2P strategy where each player broadcasts its avatar state (e.g., position and health status) to all other players. Doom became soon the scourge of network administrators due to its broadcast nature [19] and the lack of any congestion control mechanisms. Successively, id Software released its next generation FPS game, Quake [77], that abandons the P2P design and introduces a Client/Server (C/S) architecture.

In late 90s, the text-based MUD inspired the design of several three dimensional RPGs. EverQuest [30], launched from Sony in 1999, quickly became the most successful RPG. In EverQuest, players use their characters to explore a fantasy world and fight enemies to gather treasures and experience points. From a system prospective, EverQuest leverages a C/S architecture. The game is replicated on multiple "game servers", where a game server consists in a cluster of machines. Each player is associated to the game server where it creates its character, and thus can only play on that server.

Recently, World of Warcraft (WoW) [115] (launched in November 2004) became the most famous MOG counting more than 15 Million players [115], i.e., the largest play-

ers community in the history of MOGs. The main reason of WoW success is its attention to new players: WoW makes very easy to its players to get into the game and feel immediately very important in the community. Similarly to EverQuest, WoW leverages a C/S architecture where the game-play is replicated over different servers.

2.2 Peer-to-Peer

Peer-to-Peer (P2P) networks are distributed systems where peers form an overlay network on top of the Internet. P2P networks are classified into *unstructured* and *structured* according to how peers are connected to each other. In unstructured P2P networks (e.g., Gnutella [54]) the overlay links among peers are formed arbitrarily. In structured P2P networks (e.g., CAN [80] and Kademlia [63]) links between peers are formed according to a set of rules. The P2P architecture for NVEs we design leverages a structured P2P network. For this reason, we now discuss several research works that investigate the design and analysis of structured P2P networks that are relevant to the thesis.

CAN [80] is a structured P2P network that provides hash table-like functionality. CAN introduces a relationship between a logical Cartesian space and a key-space. At any point in time, the Cartesian space can be dynamically partitioned to achieve a uniform distribution of the key-space responsibilities among peers. The authors show by simulation that CAN is scalable, fault-tolerant and guarantees low latency. CAN inspired the mechanism we use to dynamically partition the virtual world responsibilities among NVE users (cf. Chapter 4).

The P2P communication infrastructure we develop to manage objects in a NVE is designed as a Kademlia [63] extension. Kademlia is a P2P network that uses the XOR distance between peer identifiers in order to organize peers as a structured overlay. Interestingly, the Kademlia topology has the property that every message exchanged in the network refreshes useful contact information both at the sender and at the receiver. Kademlia uses this property to tolerate node failures without active probing.

The Kademlia protocol was implemented by several P2P applications such as Overnet [71], eMule [29], aMule [1] and Azureus [8]. The Kademlia implementation adopted by eMule and aMule, named KAD, is a largely studied research topic as it has the highest number of simultaneously connected users. Since we use KAD as a test-bed to evaluate a distributed object management for NVEs, we now briefly describe the most important research studies related to KAD.

Stutzbach et al. [94] are the first to look at KAD. To do so, they crawl a subset of the KAD identifier space for few days. They compare KAD with other two widely deployed P2P systems, Gnutella [54] and BitTorrent [15], in order to analyze *churn* in large P2P systems. Their main results are: (i) overall dynamics are similar across different P2P systems, (ii) session lengths are not exponential, (iii) a large portion of active peers are highly stable while the remaining peers leave very quickly, and (iv) peer session lengths are correlated over a short time scale.

Successively, Steiner et al. [90][91] conduct a comprehensive analysis of KAD. They develop a crawler application to explore KAD and monitor a fraction of the KAD key-space for about six months. They identify two classes of peers: *long-lived* peers that participate in KAD for weeks and *short-lived* peers that remain in KAD no more than few days. Moreover, they observe that the distribution of peer arrivals and departures is well described by a Negative Binomial distribution, and that user session lengths follow a Weibull distribution.

2.3 P2P Networked Virtual Environments

Recently, several P2P architectures for virtual worlds have been proposed. These P2P architectures mainly address the problem of distributed avatar management, i.e., neighbor avatars discovery and avatar state updates dissemination. For the object management, these solutions simply assume that the virtual world object composition is pre-located at the clients.

To our knowledge, MiMaze [34][35] was the first server-less on-line game, deployed using IP multicast [78]. MiMaze introduces a distributed synchronization mechanism that guarantees consistency despite the presence of heterogeneous network delays. In MiMaze, avatar entities are globally replicated and kept consistent using lock-step synchronization and broadcast updates. As a consequence, the P2P game can only scale to a limited number of users, and responsiveness is limited to the speed of the slowest client.

Knutsson et al. [55] address the limitations of MiMaze designing SimMud, a distributed massively multi-player on-line game. SimMud leverages a P2P network to handle the transient game state (e.g., avatar state information) and a central server to manage persistent user state (e.g., landscape information and user accounts). SimMud users exchange avatar state updates through Scribe [82], an application-layer multicast built on top of the Pastry [81] DHT.

A different approach is adopted by Solipsis [52]. In Solipsis, each peer is directly connected to the peers whose avatars are contained within its avatar Area of Interest (AoI). Collaboration among neighbor peers is required to achieve local awareness, i.e., detect whether avatars enter or leave an avatar AoI. Interestingly, the authors show that global connectivity is obtained if each peer is connected to at least the peers whose neighbor avatars describe a polygon around the avatar coordinates. However, neighbor discovery is occasionally incomplete as newcomers may be unknown to directly-connected neighbors.

S.-Y. Hu et al. [42] design VON, a P2P-based NVE that leverages the *Voronoi Network* (i.e., the dual of the Delaunay Network) to solve the neighbor discovery problem of Solipsis. Similarly to Solipsis, each VON peer is directly connected to its avatar AoI neighbors, i.e., peers whose avatars are contained in an avatar AoI. In addition, peers are connected to Voronoi neighbors, i.e., peers whose avatars are responsible of closeby Voronoi zones. These Voronoi neighbors ensure that avatars are always included in a polygon as required to solve the neighbor discovery problem [52].

Bharambe et al. [14] design Colyseus, a P2P architecture for NVEs that uses a publish/subscribe mechanism over a DHT. Users *publish* their avatar states in the DHT and *subscribe* to zones of the NVE to discover information about their neighbor avatars. Once the peers responsible of neighbor avatars are located in the DHT, a direct overlay link is formed in order to allow an efficient dissemination of avatar state updates. Since lookups in DHTs can be very slow, Colyseus uses also predictability in data access patterns to prefetch data to the NVE users and hide long lookup latencies. Colyseus can work on top of common DHTs as well as on range-based DHTs [12].

Recently, Bharambe et al [13] design and deploy Donnybrook, a P2P architecture dedicated to build epic-scale battles in fast paced on-line games. Donnybrook proposes a shift from the conventional approach in P2P NVEs based on *AoI filtering* to *interest filtering*: peers dynamically estimate to which avatars their avatar is paying attention to, and consequently build their peer-sets. Since human beings have a limited budget of attention [64], peer-sets have a finite size even with high avatar density. Finally, avatars not included into an avatar interest set are represented as *bots*, i.e., automated avatars. These bots are *guidable artificial intelligences* that perform realistically through low-rate guidance information updates [27].

2.4 Second Life

Given the lack of information released by the SL provider, recent work has focused on studying some of its aspects. Some authors have analyzed in detail the SL client [58] and the network traffic it generates [92][47][6], while others have characterized avatar mobility [59][60]. We are not aware of any large scale data collection performed in SL, as well as of any analysis of its user Quality of Experience.

Fernandes et al. [92] propose the first study related to SL. They collect the traffic exchanged in a SL Client/Server session, measuring bandwidth consumption, packet size and packet inter-arrival times. They show that SL makes an intensive use of network resources, much more than other existing applications for virtual worlds such as on-line games. Moreover, they show that the down-link traffic is strongly impacted by avatar actions: an avatar that simply stands in SL consumes about 20 Kbps in the down-link, whereas as soon as the avatar moves the down-link traffic grows up to 110 Kbps.

Kinicky et al. [47] extend the SL traffic analysis proposed by Fernandes et al. [92] by focusing on regions with different object compositions. They reproduce some of the results obtained in [92], and they show that regions with high avatar and object density require a bandwidth 10 times larger than empty regions. In [6], the same authors develop traffic models for SL.

Kumar et al. [58] analyze the CPU usage of a high-end desktop machine running the SL client. They find that sorting translucent objects and decompressing textures stored as JPEG are the most CPU expensive operations. Similarly to [92] they also analyze the network traffic exchanged between client and server. Their results confirm the high bandwidth demand of SL, and also underline the benefits of object caching to reduce network traffic. Finally, they analyze server performance and show that the management of a region with only 5,000 rigid-body objects requires about 72% of the server computational power. As SL-like virtual worlds are expected to become more complex and realistic several CPU cores will be required.

In a different approach, La and Michiardi [59] retrieve information directly from SL servers. They deploy a simple crawler application that monitors avatar movements for short periods of time. The analysis of their traces reveals that avatars tend to interact similarly to their human counterpart, e.g., the distribution of avatar contact-times is similar to that observed in real-world experiments. In Chapter 3, we adopt a similar methodology to deploy the crawler application we used to monitor the entire SL.

Liang et al. [60] use a similar approach to [59] and collect mobility traces of 84,208 avatars spanning 22 SL regions over two months. Based on their analysis of avatar mobility, the authors suggest an hybrid avatar mobility model that incorporates both random way-point mobility model (for regions poor of objects) and pathway mobility model (for regions rich of objects).

2.5 On Line Social Networks

In Chapter 3, we study the social network formed by avatars that interact in SL. We now present the measurement studies conducted on the most popular on-line social networks.

Mislove et al. [65] study in detail the structural properties of Flickr [33] and Orkut [70], currently two very popular on-line social networks. They show that on-line social networks have structural properties very different from natural networks (e.g., they exhibit a high level of local clustering). Subsequently, Chun et al. [20] analyze Cyworld [23], a large South Korean social networking service. They compare the *friend relationships network* — the network defined by friendships among Cyworld users — with the *activity network* — the network defined by user activities, e.g., making friend relationships, sharing photos, and writing comments. They show that the two networks have a similar structure, suggesting that interactions between users in a social network tend to follow the declaration of friend relationships.

Recently, Wilson et al. [112] conduct a similar study to the one in [20]. They focus on Facebook [31], currently the most popular social networking service. In contrast to [20], they show that the activity network derived from user activities in Facebook exhibits significantly lower levels of the small-world properties shown in the Facebook social network.

2.6 Range Queries

Distributed Hash Tables are popular P2P architectures used to store and retrieve content. DHTs use a hash function (e.g., SHA-1 [68]) in order to distribute content equally among peers. This design is very efficient to build a scalable P2P lookup system, but allows only to address content punctually. However, innovative applications such as P2P NVEs require the capability to respond to more complex queries

such as *range queries*. These are requests for content whose attributes fall within a given range of the attributes space.

Range-queries over DHTs have been a very fertile research area. Ramabhadran et al. [79] propose the usage of a *binary trie* to handle range queries over a DHT. A binary trie is a tree-based data structure that uses prefix bits to direct branching in a tree. Each node in the trie is associated to a label composed by the set of bits that indicate the path in the trie to reach the node. They use the binary trie to design the Prefix Hash Tree (PHT), a distributed data structure that organizes keys in the DHT as a binary trie, and uses the DHT lookup operation to handle range queries.

P-Grid [2] goes one step further than PHT and integrates the indexing algorithm with the underlying routing algorithm. Specifically, P-Grid uses a self-organization process to structure peers directly in a binary trie. In this way, P-Grid dramatically reduces the number of routing hops to answer range queries compared to PHT.

In a different approach, Bharambe et al. [12] propose *Mercury* a scalable protocol to handle range queries. Mercury differs from previous range-based query systems in that it does not use a cryptographic hash function to index peers and content. Mercury organizes peers in several circular (logical) overlays each dedicated to an attribute of the query space. Then, each peer is statically assigned to a portion of each attribute space and it is responsible of answering the queries that fall in this range. In this way, Mercury eliminates the randomness introduced by the hash function in DHTs and allows to easily handle range queries. The drawback of this approach is that load in Mercury may be unevenly distributed. Thus, explicit overlay modifications are needed to adapt to load variations. This additional cost due to network rewirings can become intolerable in presence of churn, i.e., peer arrivals and departures, and large object sizes.

We also make a contribution in the field of range-queries over DHTs. Chapter 4 designs *Walkad*, an extension to the Kademlia [63] DHT specifically designed to handle range queries. Walkad is similar to Mercury [12] for the design rational of removing the randomness introduced by the hash function in classic DHTs. However, Walkad balances the load in the P2P network without requiring to move responsibilities around nodes in the network. Similarly to P-Grid [2], Walkad also integrates the indexing algorithm with the underlying routing algorithm. However, Walkad uses an innovative indexing algorithm optimized for P2P Networked Virtual Environments.

2.7 Delaunay Network

The usage of the Voronoi/Delaunay Network for avatar management in virtual worlds has been a very fertile area of research. There are several pieces of work in this area that are relevant to the thesis.

Backus and Krause [9] study benefits and problems of Voronoi-based P2P virtual worlds. They look at bandwidth usage, scalability and consistency of Voronoi-based virtual worlds. They show by simulation that while Voronoi-based virtual worlds perform quite well when avatars move according to a Random Waypoint model [66], consistency is greatly reduced when groups of avatars get close and avatar speed exceeds the mean distance among avatars. We also study the performance of a Delaunay-based NVE (cf. Chapter 5). Our work is complementary to [9] as we use an experimental approach. Furthermore, we introduce user Quality of Experience as a measure of the Delaunay Network performance.

Jiang et al. [49] analyze the problem of the dissemination of avatar state updates on top of a Voronoi/Delaunay network. They propose two innovative mechanisms: *VoroCast* and *FiboCast*. Vorocast constructs a spanning tree across all AoI neighbors to improve data forwarding. FiboCast dynamically adjusts the frequencies at which avatar state updates are sent by a Fibonacci sequence, such that nodes with smaller hop counts from the sender receive messages more frequently than others. We deal with a similar problem in the thesis in order to efficiently disseminate avatar state updates within a group of avatars (cf. Chapter 5). However, our solution leverages an efficient utilization of peer resources rather than a dynamic adaptation of the dissemination rate.

Steiner and Biersack [87] design a clustering algorithm for a three dimensional Delaunay Network. The main idea of their clustering algorithm is to classify links between adjacent peers in the Delaunay network into *short intra-cluster* and *long inter-cluster* links. The clustering is useful for faster navigation in the Delaunay Network and to reduce the number of messages a node receives when it travels through the NVE. In Chapter 5, we design a clustering algorithm to efficiently handle large groups of avatars in Delaunay-based virtual worlds. The main difference with our clustering algorithm is that the described solution does not take into account mobility. Conversely, our solution takes into account avatar mobility to define the clustering threshold.

CHAPTER 3

Exploring Second Life

3.1 Introduction

This Chapter presents a comprehensive study of Second Life (SL). Our motivation is twofold: (1) understand avatar behavior and object characteristics in order to allow further improvements of the SL architecture, (2) improve user Quality of Experience (QoE) in SL-like virtual worlds. Part of this work has been published in [107].

In order to conduct this study, we design and deploy a **crawler** and a **player**. The crawler is an application that connects to SL servers and exploits standard avatar capabilities to collect information about the virtual world. The player is a modified SL client that emulates the behavior of an avatar in SL, while capturing its user QoE.

We use the crawler to explore SL at different time and spatial resolutions. (1) We monitor the object composition of approximately $13,000$ public regions during one month (Section 3.4). (2) We track avatar distribution and server statistics in the entire virtual world (i.e., public and private regions) over one week (Section 3.4). (3) We fine-grain monitor five very popular regions over three days (Section 3.5). (4) We monitor avatar social behavior in the most popular region for 10 days (Section 3.6).

We use the player to explore user QoE in SL (Section 3.7). To do so, we run the player over multiple Planetlab [74] machines located worldwide, and we associate to each player a realistic avatar and user behavior extracted from the avatar traces previously collected by the crawler. Three SL servers whose regions exhibit a different object composition are used as a playground for our controlled avatars.

Our findings are as follows: at a global scale, the number of objects per region was roughly constant during one month. The active population at any point of time was between $30,000$ and $50,000$ avatars, i.e., about 0.3% of the registered avatars. Quite surprisingly, about 30% of the regions were continuously empty during six days. Servers were often overloaded even when managing only 40 concurrent avatars.

We observe that avatars tend to organize in small groups of 2 to 10 avatars. This observation suggests that the human "attention budget" theory [64][73] (i.e., human attention is bounded by a constant) may also hold in social virtual worlds. Large groups of avatars are very rare, and are mostly driven by events such as concerts or shows. Avatars tend to visit the same virtual places across different sessions. Surprisingly, around 5% of the avatar population stays connected almost all the time, and is most likely identifiable as *bots* (i.e., automated avatars).

We then construct the *social network* defined by avatar interactions, i.e., the time avatars spend being close to each other in the virtual world. We find that the SL social network is a *small-world* network that is much more similar to real life networks [108][109] than popular on-line social networks [65][112].

From a user perspective, we observe that the density of objects in a region can have a negative impact on the QoE. Specifically, we measure that in a region crowded with virtual objects, half of the time avatars have an inconsistent view of their neighbor avatars, i.e., either they do not see them or they see them at a wrong location. Moreover, in 50% of the cases this inconsistency lasts more than one second. Since acceptable values for responsiveness in on-line games varies between 300 ms and 1 sec [22], these results indicate poor user QoE under the current SL architecture.

3.2 Second Life

3.2.1 Virtual World

The virtual world of SL is composed of *regions*, independent lands of 256x256 meters in size. Each region has a maximum of four adjacent regions and can be either *public* or *private*. The public regions are owned by Linden Lab, while the private regions are purchased by individuals or companies. Owners of private regions have total control over their virtual land. They can, for instance, limit access to a selected set of avatars. Both region types run on Linden Lab servers called "Simulators".

The appearance of a region is defined by the objects it contains. Each region has a specific policy on object creation and destruction. For instance, "Sandbox" regions are used by avatars to test new objects, which are automatically destroyed shortly after their creation.

SL provides a *map* of the virtual world, i.e., a compact visual representation of both public and private regions. The map shows the number of avatars connected to each region by displaying points located at the avatar coordinates. Avatar identities are not visible on the map.

3.2.2 Client/Server Architecture

The SL design is based on a Client/Server architecture: each region is managed by a dedicated server, and users run "thin" clients that simply perform the three-dimensional rendering of the virtual world and cache the virtual objects located in recently visited regions [58]. The SL Client/Server protocol is not public, however reverse engineering efforts are underway. The libsecondlife [61] project released a set of $C\#$ libraries that allow third party applications to interact with SL servers.

Since our crawler interacts with SL servers, we now shortly describe the server-side of SL. A more complete description of the SL architecture can be found in [92] and [6].

The **Login Server** is the entry point in SL, and handles username and password verifications. The Login Server is also responsible for granting or denying access to the regions (e.g., access may be denied during server failures or maintenance operations). It maintains the following statistics: number of connected users and number of logins in the last 24 hours.

Simulators are the servers responsible for SL regions. Each simulator maintains the state of a region and performs the *visibility computation* [58], i.e., identify for each avatar located in the region the information about objects, land features and avatars that need to be transmitted to the clients. It also manages chats among avatars located within the region. A Simulator handles a maximum of 100 avatars [111]. However, larger populations are possible by mirroring the region server [58]. For clarity, we refer to simulators as *servers*.

We do not know the total number of servers in SL, nor whether SL employs load balancing techniques among servers. Moreover, SL does not mention protective measures against Denial of Services attacks and crawling operations.

3.2.3 Avatar Capabilities

A user creates an avatar by registering at the SL website [83]. This registration requires filling out an on-line form with personal information and a valid e-mail address. We now give a short description of the avatar capabilities that are used by our crawler to monitor SL.

A user entering SL must perform a login procedure. After authentication, its avatar joins the virtual world. The region where the avatar appears is either specified in the login request or derived from the avatar coordinates at its last connection. An avatar can *walk*,*run* and *fly* within a region, and also directly move to adjacent regions, provided they are public. It is also possible to perform a *teleport* operation to rapidly cover large distances. The target destination of the teleport can be within either the same region, or any other region selected from the map.

Avatars have a limited visibility area called *Area of Interest* (AoI). This area corresponds roughly to a sphere with a radius of 35 meters. Avatars teleporting to a region are informed by the server about the locations and identifiers of all objects in the region. In the following, we refer to this event as "initialization phase". Finally, an avatar can request several region statistics to a server. A complete description of these statistics can be found at *wiki.secondlife.com*.

Automated avatars called **bots** are frequently used by region owners to show some activity in their regions or simply to welcome visitors. In order to prevent the usage of bots, SL disconnects avatars that have not moved during the last 15 minutes. Not surprisingly, simple scripts allow bots to perform repetitive actions, such as head movements, which is enough to circumvent SL's bot detection mechanism. Thus, we can assume that avatars that barely move and remain connected for a long time are most likely bots. We recognize that this strategy may incorrectly identify some human-controlled avatars as bots, but it still intuitively captures the presence of bots in SL.

3.2.4 Message Types

Second Life currently uses 473 different message types in the communication between servers and clients [58]. However, three main packet types can be identified: *control*, *region* and *avatar* packets.

- The *control packets* are used to enforce security in SL and to check the state of the client connections.

- The *region packets* carry information about a region appearance and its object composition. The region packets received by a user contain the description of the portion of the region intersecting its avatar AoI.
- The *avatar packets* carry information about the state, i.e., position and body appearance, of an avatar, and about the chat messages exchanged by avatars. The avatar packets received by a user refer to the state of the avatars located within its avatar AoI.

3.3 Methodology

We use two different methodologies to study SL. First, we crawl all the information publicly available in SL. Second, we "play" SL and monitor the performance experienced by several controlled players. We first describe the *crawler* and the *player*, and then discuss the limitations and problems we encountered in our study.

3.3.1 Crawler

The main idea behind the crawler is to exploit standard avatar capabilities to obtain information about the virtual world. Our crawler is composed of multiple *subcrawlers*, each specialized in a different monitoring task (see Figure 3.1). The reasons for this are twofold. First, different types of information can be collected using different crawling techniques. Second, splitting the crawling into different tasks allows us to control the temporal resolution of each type of information we collect. For instance, tracing the movement of avatars requires sampling their position very frequently (e.g., every 30 seconds), whereas determining the total number of objects in the system can be done much less often (e.g., once per day).

Each subcrawler is a modified SL client implemented using the libsecondlife [61] libraries. A subcrawler must be associated to an avatar registered on the SL website in order to be able to log in to the virtual world. We use multiple instances of each subcrawler (associated to different avatar identities) in order to parallelize the crawling.

For each subcrawler, we describe its role, crawling technique and relationship with the other subcrawlers.

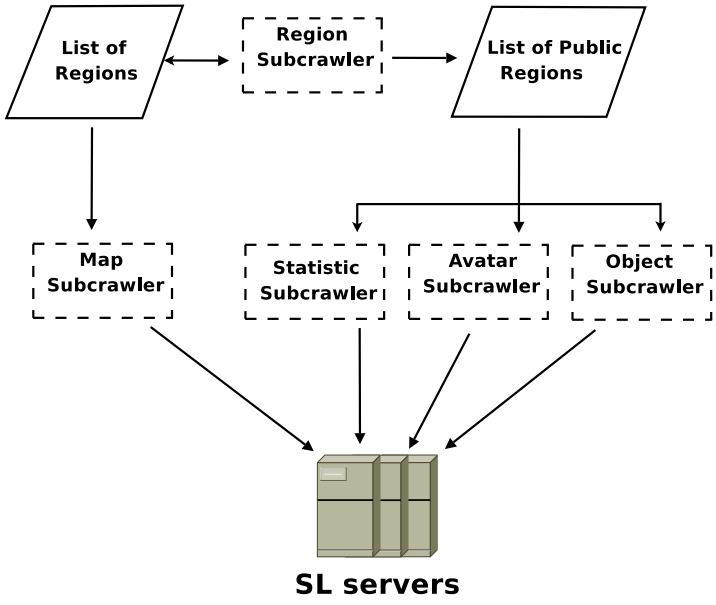


Figure 3.1: Architecture of the crawler.

- The *Region subcrawler* monitors SL to maintain an up-to-date list of its regions. This information is dynamically updated as new regions can be added to the SL virtual world. The region discovery is performed via a random walk among adjacent regions (we use a list of regions obtained at *http://stats.slbuzz.com/* to bootstrap). The Region subcrawler teleports to each region in the list to retrieve the set of adjacent regions. As new regions are discovered they are added to the list. In addition, region accessibility is verified to determine whether a region is public or private. The list of public regions is then used by the Statistics, Avatar and Object subcrawler, while the complete list (public and private regions) is used by the Map subcrawler.

- The *Object subcrawler* tracks the evolution of objects in all public regions. It teleports to a public region and accomplishes the initialization phase, during which it is informed by the server of the coordinates and identifiers of all objects on the region. Then, it dumps this information and teleports to a new region.

- The *Statistics subcrawler* collects the statistics maintained by the servers of the public regions. It teleports to a public region, queries its server, dumps the results, and then moves to another public region. We collect the following server statistics: number of connected avatars, time dilation, which expresses the load on the server, and total number of packets going in and out from the server.

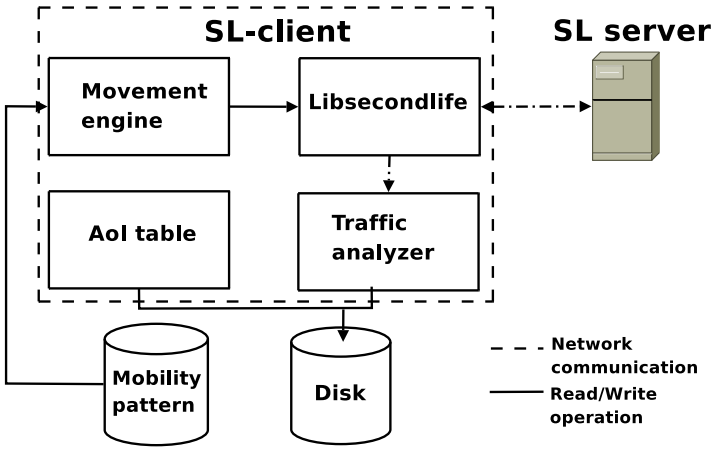


Figure 3.2: Architecture of the player.

- The *Map subcrawler* monitors the location of avatars as shown on the SL map. For each region, the subcrawler locates it on the map, and collects the coordinates of all the avatars currently connected to it. This task simply requires logging in to SL. Unfortunately, the Map subcrawler cannot identify the avatar identities as they are not shown on the map, which led us to develop the Avatar subcrawler.

- The *Avatar subcrawler* obtains the identity and position of the avatars connected to public regions. First, it uses the map to determine the position of avatars within a region. Then, it teleports to each of these coordinates to obtain the identities of nearby avatars and to determine their coordinates with greater accuracy. Therefore, the Avatar subcrawler may need to teleport several times in order to crawl the entire region.

Note that the Avatar, Statistics and Map subcrawlers collect partially redundant information. We exploit this redundancy to check the correctness of our methodology.

3.3.2 Player

We reproduce avatar behaviors via controlled SL clients in order to evaluate user QoE. To do so, we use libsecondlife [61] to implement a *player* that emulates and automates avatar behavior while collecting traces related to user QoE.

The player performs the login of an avatar to a target region and moves the avatar in the region according to an input mobility pattern. Figure 3.2 shows the architecture of the player, which consists of the following components.

- The *libsecondlife* module handles the communication between an avatar and a server. It contains the API that allows an avatar to perform actions such as login or movements.
- The *movement engine* manages the movements of an avatar according to a given mobility pattern. It uses the libsecondlife module to inform a server of the avatar behavior, e.g., movements in the region.
- The *AoI table* is a data structure that contains the positions and names of all avatars within an avatar AoI. The AoI table is updated according to the information received from the server. A snapshot of the AoI table is copied to the disk every 200 ms or when a modification of its content occurs.
- The *traffic analyzer* records all the incoming and outgoing packets exchanged between client and server. Subsequently, it parses each packet to extract information about its content, e.g., avatar, region or control traffic (Section 3.2.4), and volume.

3.3.3 Problems and Limitations

To collect data in a scalable and accurate way, we had to solve several problems. We now discuss these problems as well as the limitations of our crawling strategies.

Crawling Performance

We refer to *crawling performance* as the number of regions a subcrawler monitors in a given time. As mentioned in Section 3.3.1, we run multiple instances of each subcrawler in parallel in order to increase the crawling performance. However, we observed that increasing the number of parallel instances does not necessarily improve performance. Figure 3.3 shows that the performance of the Statistics subcrawler degrades beyond 60 concurrent instances. We suspect that SL employs a rate-limiting policy against IP addresses that generate a large amount of traffic.

Experimental Hazards

Officially, SL does not mention any banning policies against avatars with unusual behaviors. However, many of the avatars associated to our subcrawlers were banned. The banning procedure consists of an exclusion due to “account verification”. While

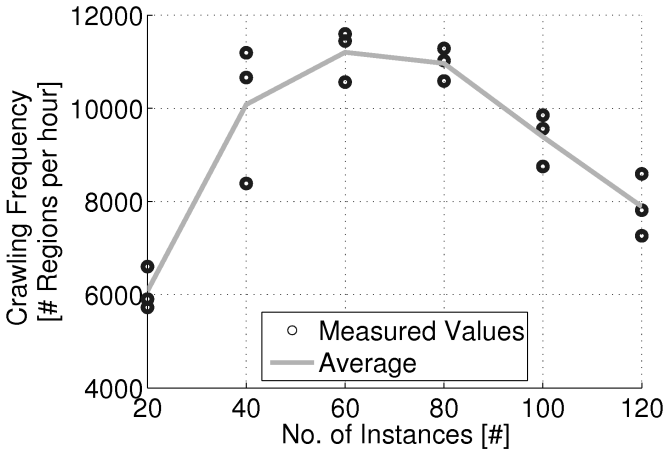


Figure 3.3: Crawling performance [Statistics subcrawler ; 18 hrs experience].

we could not identify the exact behavior that causes banning, we found that a heavy usage of the teleport action increases the chances of being banned.

If a banned avatar attempts to login several times, its IP address is blacklisted. To avoid getting blacklisted, our each of our subcrawler detects when it has been banned and automatically replaces the associated avatar identity with a new one.

Finally, we also observed a high degree of instability in SL. During a period of one month, the service was down multiple times due to maintenance, server updates, or crashes [84]. While our short traces were mostly unaffected, outages had an impact on our long-term ones (see gap in Figure 3.8).

Teleport Inefficiency

The Avatar subcrawler uses the teleport operation to get close to avatars on a region and collect their identities. We observed that a teleport can "fail", i.e., reach an incorrect location, when the coordinates of its destination lie inside an object. This "teleport inefficiency" can limit the completeness of the traces. In order to quantify this crawling error, we compared the number of avatars observed by the Avatar subcrawler with those shown on the SL map during a 12-hour period. Figure 3.4 shows the Complementary Cumulative Distribution Function (CCDF) of the percentage of avatars per region the Avatar subcrawler correctly identifies. We notice that the Avatar subcrawler identifies all avatars present on the region only in 50% of the regions. In the other regions, a fraction of the avatar identities is not retrieved introducing a crawling error.

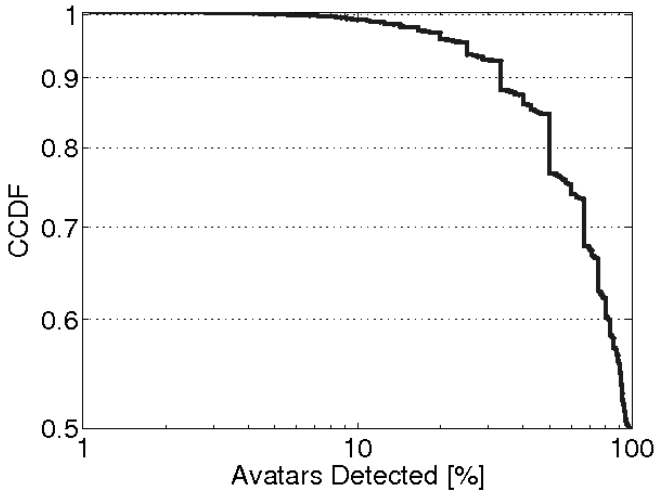


Figure 3.4: Completeness of the Avatar subcrawler [Avatar subcrawler ; Map subcrawler ; 12 hrs experience].

Our original Avatar subcrawler suffered also from heavy banning due to a high number of teleports. We solved this by assigning a unique IP address and subcrawler instance to a given region, and modifying the assignment every hour. This avoids banning, but requires a number of IP addresses and avatars that is linear with the number of regions, limiting crawling scalability. For this reason, we decided to limit the scope of the Avatar subcrawler to a few very popular regions. Furthermore, we verify that the object composition of the selected regions does not cause any teleport failure and the consequent crawling error.

Realism of the Player

Libsecondlife implements a simplified Client/Server communication protocol compared to the official SL protocol [61]. Analyzing the incoming traffic at the player, we notice that many packets are ignored since they are unknown to the libsecondlife libraries. For this reason, the traffic volume exchanged between our player and the servers is generally smaller than what we would observe with the official client [92][47]. However, building a player with the official SL client would have the following limitations: (i) need of real users to control the avatars (ii), limited access to the data retrieved by the clients.

Finally, we measure user QoE by "re-playing" real (monitored) avatar behaviors using bots. Intuitively, avatar behavior on a region changes according to factors such as the performance perceived by its user, user interest in the region, and objects encountered. Our methodology cannot capture these factors. We also use bots that do

not have any customization (e.g., fancy clothes) and that perform only simple movements. However, creating SL sessions that involve real players is an hard task, and does not allow a fair comparison of different regions.

3.4 A Global View at Second Life

We focus on the data collected by our crawler and on some data provided by SL through their website and Login Server in order to analyze SL-wide characteristics. When possible, we compare our results with these two sources to check their consistency. Finally, we use visual inspection to confirm some of our observations and interpretations.

3.4.1 Data Collection

Each subcrawler collects data at a different time resolution. In addition, some subcrawlers can traverse a set of regions much faster than others, according to the technique they use to collect information. We call *crawling frequency*, the frequency at which a subcrawler completely monitors a set of target regions. Finally, some subcrawlers require more resources than others (e.g., IP addresses and avatar identities), and are therefore executed for shorter periods of time.

We monitored the evolution of all regions and objects in SL during 28 days with a crawling frequency of 24 hours. We used three instances of the Region subcrawler and five instances of the Object subcrawler. Traces were collected between March 29, 2008 and April 25, 2008, except for April 4 and 5 when the SL service was down [84].

We ran 60 concurrent instances of the Statistics subcrawler, as this yields the highest crawling performance (Figure 3.3). With this configuration, the Statistics subcrawler can crawl about 11,000 regions in one hour. On March 29, 2008, the Region subcrawler identified 12,765 public regions, so we set the crawling frequency of the Statistics subcrawler to 90 minutes to be able to monitor all public regions with a safe time margin. Traces were collected for 6 days between March 29, 2008, and April 4, 2008.

We monitored the SL map with 40 instances of the Map subcrawler and a crawling frequency of 15 minutes. Traces were collected between April 18, 2008 and April 21, 2008. The traces refer to the total 17,526 regions identified by the Region subcrawler on April 18, 2008.

Table 3.1 summarizes each subcrawler configuration, trace length, and crawling frequency.

subcrawler	Instances	IP@s	Regions	Frequency	Days
Region	3	1	-	1/24 hrs	28
Object	5	1	-	1/24 hrs	28
Statistics	60	1	12,765	1/90 min	6
Map	40	1	17,526	1/15 min	3

Table 3.1: Second Life crawling summary.

3.4.2 Regions

Table 3.2 summarizes the total number of regions discovered by the Region subcrawler, as well as the official number reported by the SL website.

We observe that the Region subcrawler discovered a larger number of regions compared to official figures. These additional regions are not reachable and were discovered as adjacent of active ones. Therefore, they are probably a fraction of the virtual world reserved for future customers, and thus do not count in the official statistics.

	March 29	April 18	April 25
Public regions (RS)	12,765	13,220	13,261
Total regions (RS)	17,280	17,526	17,573
Total regions (SLW)	13,693	N/A	14,150

Table 3.2: Number of regions in SL (RS=Region subcrawler, SLW=Second Life website).

The 6-day trace collected by the Statistics subcrawler shows that many regions experienced periods of unavailability. There are two possible causes for this: (i) the region server was down, or (ii) the maximum number of avatars per server has been reached.

We compute the region *availability* as the probability that a server accepts a connection from the Statistics subcrawler, i.e., the number of times it accepts a connection divided by the total number of connection attempts during 6 days. Figure 3.5 shows the Cumulative Distribution Function (CDF) of region availability. We observe that 90% of the regions have an availability of 0.9 or more, but only 1% show a high availability of 0.99 or higher. This is probably due to short maintenance operations or failures. The bottom 1% of the regions have an availability of 0.7 or lower. Our traces show that these highly unstable regions are not among the most popular ones

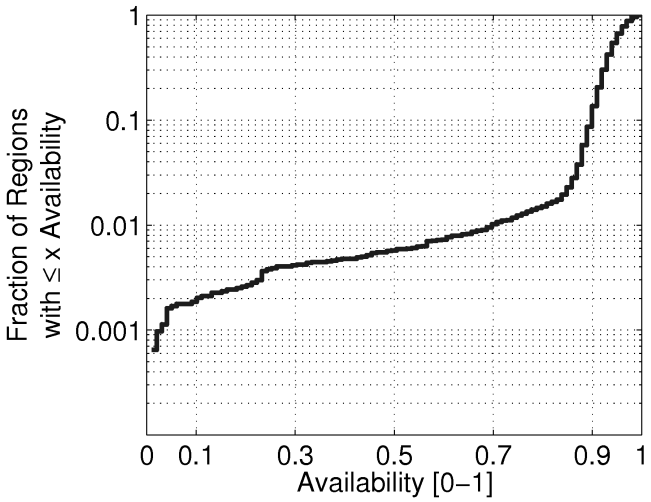


Figure 3.5: CDF of region availability [Statistics subcrawler].

(Section 3.4.6), suggesting that their unavailability is not due to server overload, but it is more likely due to server failures.

3.4.3 Users

Figure 3.6 shows the evolution over time of the number of on-line users, as measured by the Map subcrawler and reported by the Login Server (this data is obtained by monitoring the Login Server [85]). Both curves exhibit the same daily cycle. However, the Login Server reports 10,000-20,000 more users than the Map subcrawler. Moreover, during a major SL outage on Friday at 14:00, the Login Server reported a drop of 10,000 avatars, while our Map subcrawler observed a decrease by 20,000. This suggests that the values provided by the Login Server may be inaccurate, and probably averaged over long time periods.

3.4.4 Server Traffic

The Statistics subcrawler collects information about the rate of outgoing server packets as reported by the SL servers. The curve of the aggregate traffic generated by all servers shows a daily cycle ranging from 1.7 to 3.2 million packets per second. Moreover, the traffic's daily cycle closely follows that of Figure 3.6, which is to be expected for a Client/Server architecture. The correlation coefficient between the number of avatars in a region and the traffic volume is 0.8.

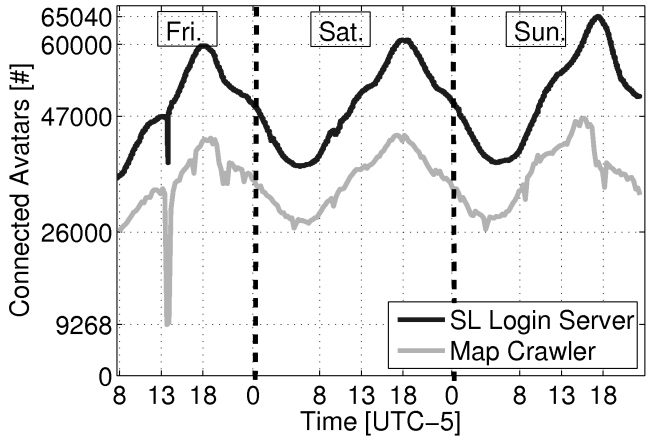


Figure 3.6: Active population over time [Map subcrawler ; SL Login Server ; Coordinated Universal Time - 5].

Knowing that the mean packet size in SL is 500 bytes [92], the aggregate traffic generated by all servers at peak time can be estimated to be around 13 $Gbps$. The peak number of users is $47,000$ measured on Sunday at 18:00, which yields an average bandwidth consumption of 280 $kbps$ per client. This confirms the results reported in [92] and shows the high bandwidth consumption of the SL service.

3.4.5 Object Distribution and Dynamics

We analyze the 28-day trace collected by the Object subcrawler with a crawling frequency of 24 hours in order to understand SL object characteristics.

Object Distribution

We identified about 7 million unique user-generated objects across all public regions. Figure 3.7 shows the CCDF of the number of objects. Since the distribution did not significantly change over 4 weeks, we only plot the data for the first day (March 29, 2008) and for the last day (April 25, 2008). 30% of the regions are almost empty, containing less than 100 objects. Around 65% of the regions contain a relatively low object count, between 100 to $1,000$, while only 5% of the regions have $1,000$ objects or more. The richest region contains nearly $13,000$ objects. We would like to recall that the Object subcrawler cannot collect information about object sizes as this would take too much time and make the crawl operation very slow. Therefore, it is possible that some regions with a low object count actually have a more complex environment (e.g., bigger objects) compared to other regions with a larger object count.

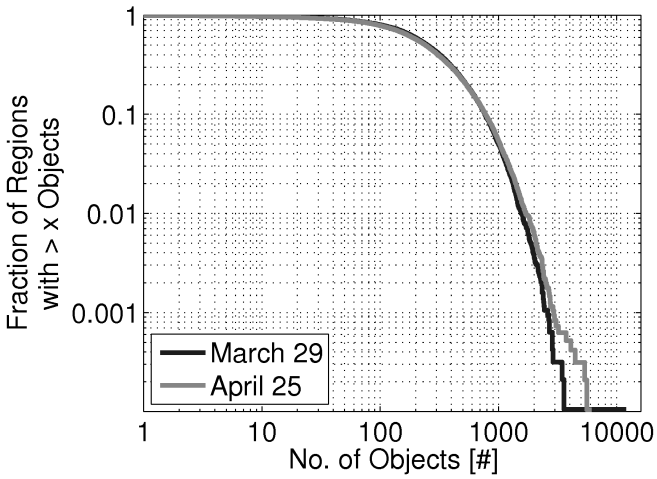


Figure 3.7: CCDF of the object distribution across regions ; [Object subcrawler].

Object Dynamics

We now analyze the evolution of the number of objects over time. For each region, we compute the difference between the number of objects it contains at day i, and its initial object count observed at day 1, i.e., the first day of the monitoring (March 29, 2008). Figure 3.8 shows some significant percentiles of the distribution of these differences measured for all regions (the gap between days 6 and 9 is due to a SL outage). We observe that 50% of the regions (between the 25th and 75th percentiles) are almost completely static, showing a small variation between ± 50 objects after 28 days. The 10th and 90th percentiles remain between ± 250 objects, showing modest object variation rates in most regions. The median value is nearly zero, and the percentiles are almost symmetrical, indicating a similar object creation and destruction rate. In fact, our traces show that the total number of objects in SL remains approximately constant over time. Notice the presence of two drops between days 20-25 and 25-27. During these days, the SL website reported that their servers were being updated. Thus, we believe that these drops correspond to objects being lost during server updates and then slowly recovered. Finally, although not shown in Figure 3.8, we observed minimum and maximum variations close to ± 4000 objects. This implies that the regions at the bottom and top 10% of the distribution show a highly unstable behavior, with a large number of objects being continuously created and destroyed. These are mostly regions where users test their objects (e.g., sandbox regions), and which typically erase objects soon after they are created.

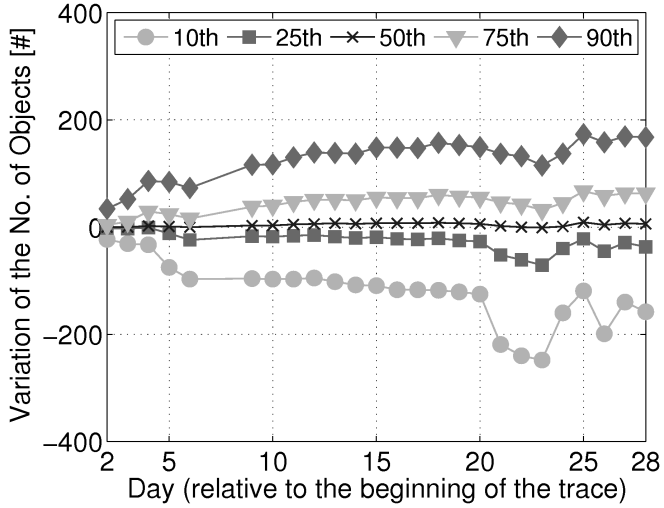


Figure 3.8: Percentiles of the variation of the no. of objects per region ; [Object subcrawler].

3.4.6 Region Popularity

We use the 6-day trace collected by the Statistics subcrawler in order to analyze the popularity of regions in terms of the number of avatars that visit them.

As the number of avatars in a given region is highly dynamic, we study for each region the evolution of the population with time. For each region we calculate the *population CDF*, i.e., the Cumulative Distribution Function (CDF) of the number of avatars observed during the 6-day period. Since we cannot plot the population CDFs for the $12,765$ monitored regions, we will take a few significative percentiles and plot their distribution among all regions.

Figure 3.9 shows the distribution among regions of the 0th, 25th, 50th, 75th, and 100th percentiles of the population CDFs. Note that the 100th and the 0th percentile correspond respectively to the maximum and minimum population observed for a given region. Similarly, the 75th percentile may be interpreted as a peak population, the 50th percentile as a median or typical population, and the 25th percentile as a residual population. Accordingly, Figure 3.9 shows that 30% of the regions are empty all the time, while around 45% of the regions have always less than 5 avatars. The 75th percentile curve overlaps with the 100th percentile one between 0 and 4 avatars. As a consequence, regions whose population is small most of the time have a small population all the time with no exception. The 0th percentile curve indicates that about 30% of the regions are never completely empty. However, the curve rapidly goes to zero, showing that regions with continuous activity are rare, e.g., less than 1% of the regions have a minimum population of 10 avatars. Focusing now on

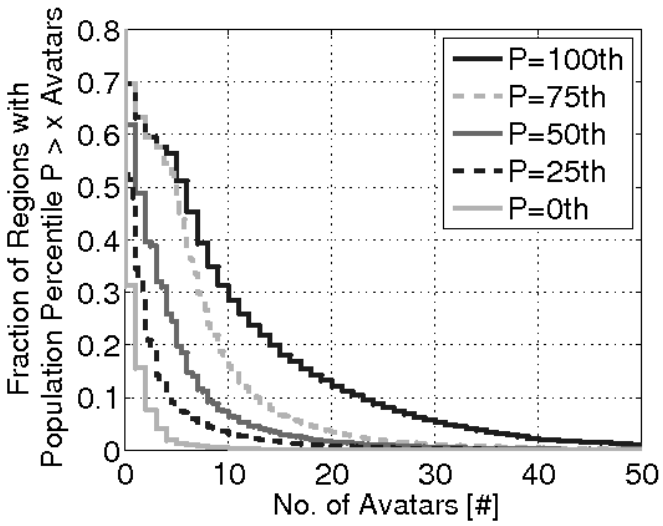


Figure 3.9: CCDF of several percentiles of the population CDFs ; [Statistics subcrawler].

larger populations, we observe that around 5% of regions have at least 30 avatars as a maximum population, but that this number drops to 18 avatars for a peak population (75th percentile) and to 12 avatars for a typical population (50th percentile). Hence, although a non-negligible number of regions are occasionally densely populated, they usually contain few avatars. These results indicate that different SL regions have different population characteristics, and may necessitate different resource provisioning according to their popularity profile.

3.4.7 Virtual Groups

We are interested in determining to what degree avatars concentrate in *groups*, i.e., aggregation of avatars where each avatar is within visibility range from each other (35 meters as defined by SL). The rationale is that avatars located within such *virtual groups* are highly likely to interact with each other. The results we present in this Section are obtained from the analysis of the 3-day trace collected by the Map subcrawler with a crawling frequency of 15 minutes.

We estimate the number of virtual groups in a region by using the *k-means* clustering algorithm [5] to partition avatars in circles of radius $r \leq 35$ meters. We proceed as follows. Let n be the number of avatars in a region. The algorithm takes the avatar coordinates $a_i = (x_i, y_i)$ with $1 \leq i \leq n$, and a number of target partitions k. It then clusters the avatars into k circular areas with center coordinates $c_j = (x_j, y_j)$ and radius r_j, where $1 \leq j \leq k$. We run the algorithm iteratively for increasing values of

k until all circles have a radius $r_j \leq 35$ meters. The final value of k gives the number of virtual groups in the region, and c_j the coordinates of the group center.

We use the k-means clustering algorithm since it minimizes the distance of avatars from the center of the virtual group. Note that this algorithm does not track groups that move across the region. However, since avatars in SL tend to have a static behavior, this limitation has only a minor impact on our clustering scheme.

Virtual Group Sizes

Figure 3.10 shows the CDF of virtual group sizes across all regions. We observe that 50% of the avatars don't form groups. Surprisingly, 45% of the virtual groups are made of only 2-10 avatars. This could be explained by the *budget of attention* theory [64], which suggests that human beings can only focus their attention to a maximum of 5-9 entities at the same time. Finally, we observe a negligible number of virtual groups with more than 20 avatars. Note that large avatar crowds which extend beyond 35 meters may be split by our algorithm into smaller groups.

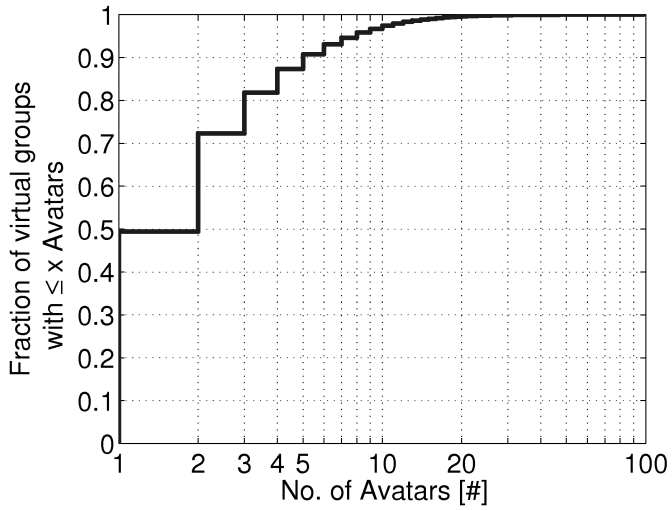


Figure 3.10: CDF of virtual group sizes [Map subcrawler].

Points of Interest (POIs)

Regions usually contain Points-of-Interest (POIs), i.e., spots that attract several avatars. In order to detect the presence of POIs, we look for virtual groups that are stable with respect to time and location. Therefore, we use a metric that we call the group's *spot lifetime*, which is defined as follows. For every new group, we record

its initial center coordinates. Since the group may move or dissolve, we compute its spot lifetime as the time elapsed from its creation until we observe no virtual groups centered within 35 meters (the avatar visibility area) from its initial center coordinates. If the center of a group moves more than 35 meters from its original position, we consider that a new group has formed at the new center coordinates.

Figure 3.11 shows the CDF of spot lifetimes for all groups and for different ranges of S, the average group size. We notice that groups with large sizes tend to have a larger lifetime. This suggests the presence of POIs near the center of groups with high spot lifetimes. We also observe that 50% of the large virtual groups ($S > 10$) have a rather short spot lifetime. These groups can be event-driven groups, i.e., located near short-lived POIs.[1] Conversely, the remaining 50% have a very long spot lifetime. Intuitively, the area around popular POIs is unlikely to become empty, especially in popular regions, resulting in very long spot lifetimes.

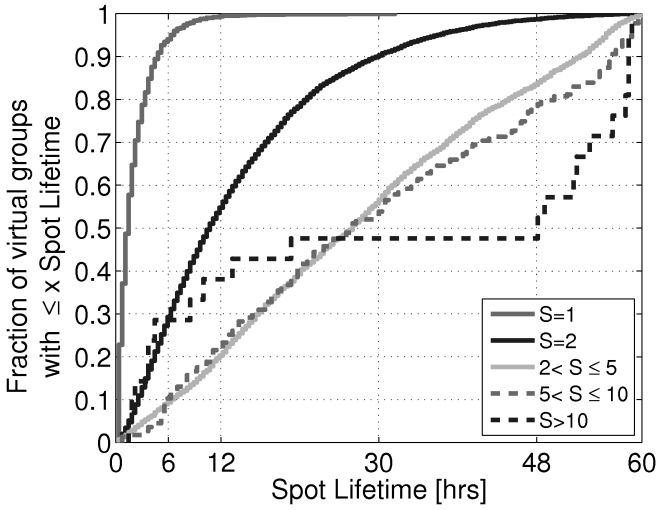


Figure 3.11: CDF of spot lifetimes for different average group size S [Map subcrawler].

Figure 3.11 also provides some interesting insight about isolated avatars. Around 40% of these avatars have a spot lifetime of a few minutes. These avatars are most likely exploring a region. Thus, the area traversed by these avatars are unlikely to be POIs. However, 10% have a lifetime between 5 and 32 hours, i.e., they stay at the same spot for a very long time without interacting with any other avatar. It is unlikely that this behavior is coming from human beings, so we suspect that these avatars are computer controlled, i.e., bots. Given that 50% of the virtual groups are composed by a single avatar (see Figure 3.10), we conjecture that at least 5% of the entire SL population consists of bots.

[1] We visually inspected some of the regions where we found these short-lived POIs, and we verified the presence of concerts and shows.

3.5 In-Depth Analysis of the Most Popular Regions

The analysis of SL at the global scale has left many open questions, such as "how do avatars behave and socialize?", "how do SL servers perform in presence of different avatar populations?". To answer these questions, we need to do a fine-grain monitoring of SL, e.g., collect avatar identities and server statistics with a very high resolution. We now describe the fine-grain crawl of five very popular regions and an in-depth analysis of these regions.

3.5.1 Data Collection

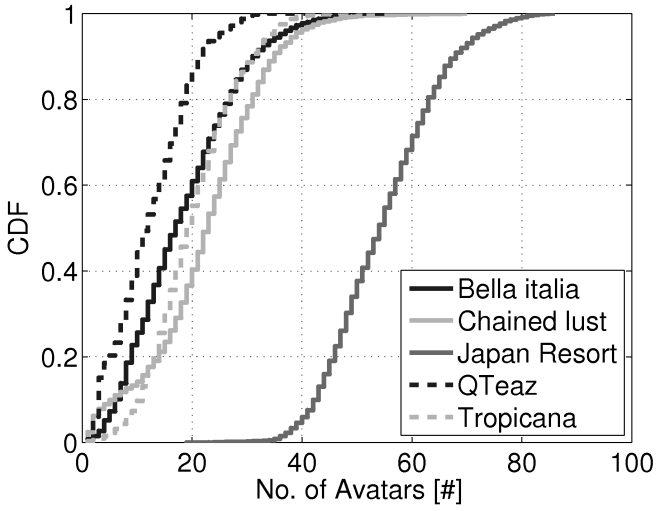
We combined in a single application the functionalities of the Statistics, Map and Avatar subcrawler (we refer to it as Stat/Map/Av subcrawler). Given the limitations of the Avatar subcrawler (Section 3.3.3), we selected 5 highly popular regions where the Avatar subcrawler achieves a 100% accuracy. We used 5 Planetlab [74] machines and a crawling frequency of 30 seconds. Traces were collected during 3 days between May 1 and May 4, 2008. We now give a brief description of the five regions we crawled.

- *Bella Italia* is a meeting place made of a central square with some trees and benches.
- *Chained Lust* is a shop with adult content. The land contains a single building with lots of objects. Avatars teleporting to this region automatically enter this building.
- *Japan Resort* is an island with few trees and thatched huts.

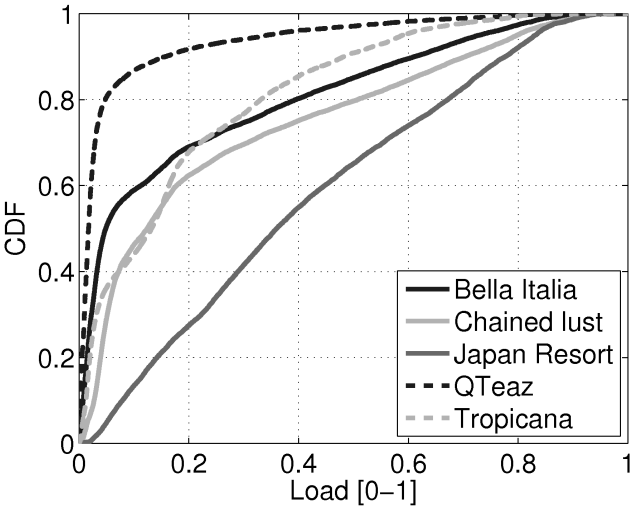
  QTeaz is a region dedicated to leisure. It consists of few small buildings with games and other activities.
- *Tropicana* is a resort. There is a beach, a small lake and several vacation facilities. There are advertisements about music events where avatars can dance and meet.

3.5.2 Region Popularity and Sever Load

Figure 3.12(a) shows the population CDF (see Section 5.3) for the five regions. QTeaz and Japan Resort are respectively the least and most popular regions, while the other



(a) Population



(b) Server load

Figure 3.12: Population and server load ; [Stat/Map/Av subcrawler].

three regions have a comparable popularity. Notice that all regions are almost never empty. In addition, Japan Resort has never less than 20 avatars. The active population per region rarely exceeds 40 concurrent avatars, except for Japan Resort, whose peak population is 84 avatars.

In SL, servers "slow down" the virtual time as a way to cope with high loads. This is called **time dilation** (*td*), and is defined as follows: $td = 1$ means that the server is running at full speed, whereas $td = 0.5$ means that it is running at half-speed. We

consider $L = (1 - td)$ as a measure of load on a server, e.g., $td = 0$ means maximum load (e.g., $L = 1$) and $td = 1$ means minimum load (e.g., $L = 0$). We measure time dilation and so load in the regions by simply interrogating the servers (Section 3.3.1). Figure 3.12(b) plots the CDF of the load per region. As expected, the more a region is popular, the more its server is loaded. We can see that half of the time, Japan Resort has a load larger than 0.35. Interestingly, we observe that both Japan Resort and Chained Lust exhibit very high load values (e.g., larger than 0.8) despite the significant difference in their population CDFs.

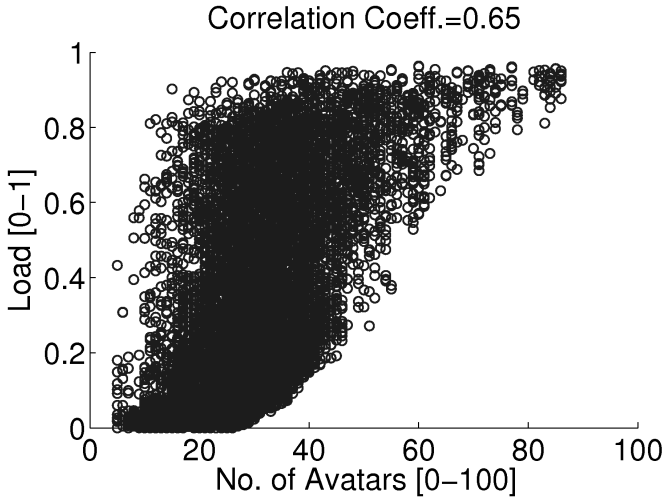
We now analyze the impact of avatar population on the server load. Figure 3.13 shows a scatter-plot of the number of avatars and the server load. We only plot the data for Chained Lust and Japan Resort since they are the most representative.

Figure 3.13(a) shows a weak positive correlation of 0.6, and the presence of two different trends. When the population in the region is lower than 40-50 avatars, the server load is very variable, probably impacted by avatar behaviors. However, when the population grows over 50 avatars the server is always very loaded, as we never observe load values smaller than 0.6. The trends we highlighted in Figure 3.13(a) are representative of all regions, with the exception of Japan Resort (see Figure 3.13(b)). This region shows a lower correlation coefficient of 0.34. In addition, we observe a general trend similar to Chained Lust, but starting at around twice the population size. This result may suggest that Japan Resort is assigned more server resources than other regions.

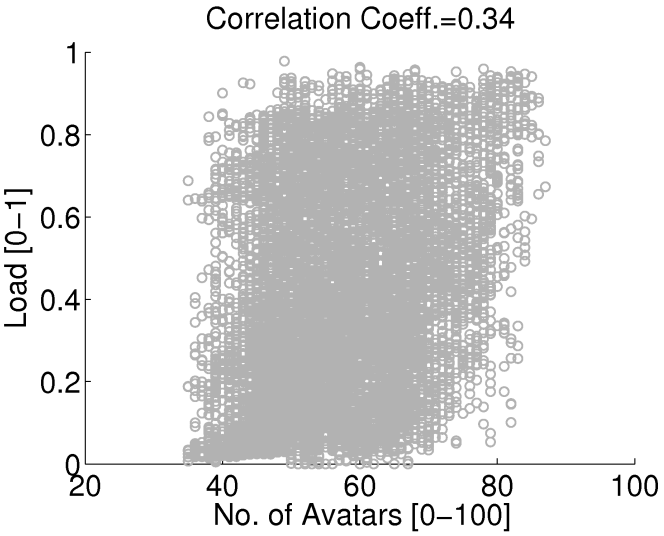
3.5.3 Avatar Behavior

Avatars join and leave a region multiple times. We use the term *session* to denote the time an avatar spends in a region. Note that a session does not span multiple regions, as we cannot detect whether avatars leaving a region are moving to another region or leaving SL. We also assume that each user is associated with a unique avatar. Figure 3.14 shows the CDF of user session times for each region. Despite the regions different popularities, users spend roughly the same time in each region. 50% of the users stay connected less than 10 minutes per session while 15% stay connected 100 minutes or more. Finally, 5% stay connected more than 10 hours. An analysis of the movement of these 5% shows that 98% of the time they do not change their position at all, suggesting that they are bots.

Figure 3.15 illustrates an analysis of avatar movement patterns. We distinguish between standing, walking, running, flying and teleporting according to the avatar's



(a) Chained Lust



(b) Japan Resort

Figure 3.13: No. of avatars versus server load ; [Stat/Map/Av subcrawler].

speed. Surprisingly, avatars stand on the same point more than 80% of the time. The remaining time they mostly teleport or walk (flying and running only account for negligible values). This highly static behavior is probably due to two factors. First, in popular regions avatars spend most of their time chatting with nearby avatars. Second, these regions experience high server load (see Figure 3.13), which introduces a lag and thus makes avatar movements more difficult.

To understand if avatars tend to go back to previously visited places in SL, we analyze the interaction between avatars and the virtual groups. Figure 3.16 shows the

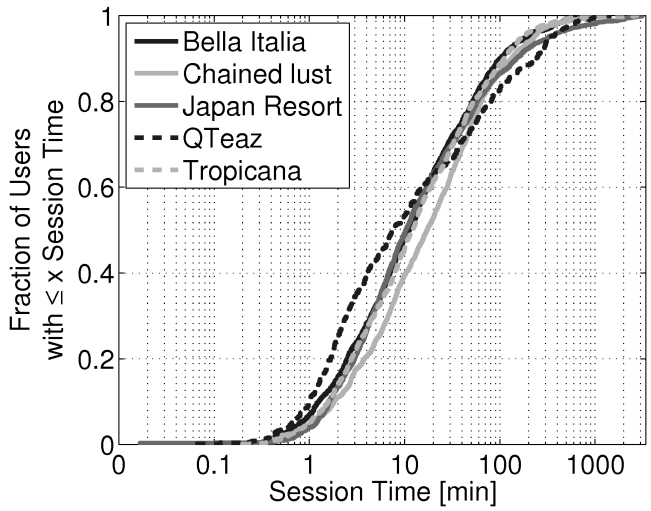


Figure 3.14: CDF of user session times ; [Stat/Map/Av subcrawler].

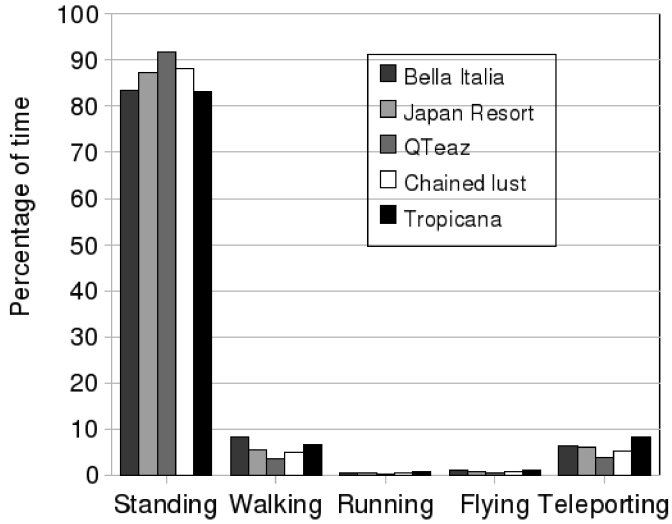


Figure 3.15: Avatar movement patterns ; [Stat/Map/Av subcrawler].

CDF of avatar visits to the same virtual group during 3 days. About 50% of the avatars come back at least once in 3 days to a previously visited group, while 30% revisit the same group at least once per day. Consequently, there exists a high level of predictability in avatar behavior, which can be explained by the social nature of SL: avatars are attracted to places they like, or where they can meet avatars they already know or who share similar interests.

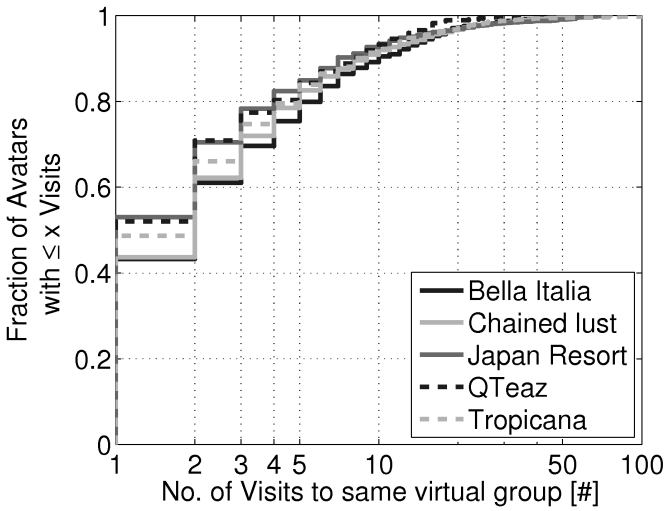


Figure 3.16: CDF of avatar visits to the same virtual group ; [Stat/Map/Av subcrawler].

3.6 Second Life as a Social Network Service

Researchers have shown that avatars in virtual worlds tend to interact similarly to human beings in real life [59]. They meet, spend time together and make friends. This behavior suggests that avatars construct an *on-line social network*, an Internet-based network that represents the social relationships existing among human beings. Popular examples of on-line social networks are the ones created on social sites (e.g., Facebook [31] and Cyworld [23]), content sites (e.g., Flickr [33]), or game sites (e.g., World of Warcraft [115]).

In this Section, we first describe our methodology for constructing a social graph among SL avatars based upon a trace of SL avatar interactions. Then, we analyze the characteristics of the SL social graph.

3.6.1 Contact and Social Graph

We define contacts in SL based upon proximity in the virtual world. An intuitive result in human communication is that "closer together" means "more likely to converse" [28]. Recent work has shown that avatars share behavior similar to human beings [59], e.g., they gather in popular places to meet friends. Hence, we assume that the distance between avatars plays an important role in avatar communication.

We assume that two avatars are interacting, i.e., there exists a *contact* between them, when their Euclidean distance is less than an **interaction range** R. We recognize

that this assumption may identify contacts where avatars are not directly interacting (e.g., "strangers" avatars passing each other), but it still intuitively captures avatar contacts and the possibility of interaction. We define **contact time** as the time interval during which two avatars have an Euclidean distance smaller than R. Finally, we define **session time** as the continuous on-line time of an avatar.

We now introduce the **contact graph** similar to that previously described in [67]. The contact graph $G_t = (V_t, E_t)$ is the collection V_t of avatars connected to a SL region as well as the edges E_t connecting the avatars at a time t. G_t is an *unweighted* graph, i.e., edges are not assigned any a-priori weight. An edge $< i, j >_t$ connecting nodes i and j in $V(G_t)$ equals 0 when the Euclidean distance d between the coordinates of avatars i and j at time t is greater than R, whereas it equals 1 when $d < R$. G_t is also an *undirected* graph, i.e., $< i, j >_t = < j, i >_t$ for any i and j.

Similar to the activity network studied in [20] and the interaction network studied in [112], we now introduce the **social graph** as the network of friendships that users construct with their behaviors. More formally, the social graph $G = (V, E)$ is the collection of avatars V visiting SL as well as the edges E connecting these avatars. G is a *weighted* and *directed* graph, i.e., each edge $< i, j >$ connecting two avatars i and j in $V(G)$ has two associated weights $w_{i,j}$ and $w_{j,i}$. We compute $w_{i,j}$ and $w_{j,i}$ as the ratio of the sum of all contact times between avatars i and j and the sum of the session times of respectively avatar i and avatar j. Intuitively, $w_{i,j}$ is the fraction of "virtual" time avatar i spends being close to j. Therefore, the pair of weights $w_{i,j}$ and $w_{j,i}$ captures the "importance" of the social connection between avatars i and j, e.g., acquaintances, friends or relatives.

3.6.2 Data Collection and Limitations

In order to construct an instance of the SL social graph, we need to overcome the limitation of the Stat/Map/Av subcrawler that can only monitor avatars for 3 days due to IP blacklisting from Linden Lab (Section 3.3.3). In fact, such a short time period is not sufficient to capture social relationships among avatars.

We proceed as follows. First, we crawl a single region. Second, we modify the crawler to toggle its Internet connection by interacting with an home gateway router. This procedure triggers the Internet Service Provider to assign a new public IP address to the connection, circumventing the IP blacklisting mechanism adopted from the SL provider.

Traces	α (in)	α (out)	$\frac{C}{C_{er}}$	$\frac{L}{L_{er}}$
Flickr [65]	1.74	1.78	47,200	-
Orkut[65]	1.5	1.5	7,240	-
Facebook [112]	1.5	1.5	21,866	-
Film Actors [109][10]	3	3	2,925	1.22
Power grid [109][4]	-	-	16	1.50
C.elegans [109][48]	2.2	2.2	5.6	1.17
Second Life	2.2	2.2	70	1.1

Table 3.3: Comparison of the SL social network with several real life networks and on-line social networks.

We crawl the popular "Japan Resort" region for 10 days between July 22, 2008 and August 2, 2008, and we monitor $3,291$ unique avatars. According to SL statistics [86], these avatars account for about 1% of the unique avatars logged in during this time. The traces contain a large gap of about 20 hours due to a major outage of the region, and other minor gaps of a few minutes likely due to server updates. These holes do not represent a loss of information since no avatar activity was possible.

In order to choose a value for the interaction range R, we compute the minimum distance observed between any pair of avatars at each crawling snapshot. We found that 99% of the minimum distance values are smaller than 5 meters. Therefore, we set R to 5 meters.

3.6.3 A "Real" On-line Social Network

We now characterize the features of the SL social graph defined by the traces of avatar contacts in the "Japan Resort" region. When discussing these features, we also compare and contrast them with the characteristics of both on-line social networks and other networks, e.g., power grid and film actors. Table 3.3 summarizes the SL social graph features and how they compare to the characteristics of other networks as determined by previous studies.

We focus on the analysis of the *complete* graph G, i.e., two nodes i and j in G are connected if $w_{i,j}$ and consequently $w_{j,i}$ are non-zero. Thus, we do not discuss the embedded social graphs formed by filtering weak social connections among avatars. This feature of the social graph will be used in Chapter 5.

Degree

The *degree* of a node $v \in V(G)$ is the number of edges incident to v, and represents the number of unique avatars encountered by avatar v during the crawl period. We start with a quantitative analysis of the node degree distribution computed over G as defined at the end of the crawl period. We observe that 90% of the values in the node degree distribution are between 1–30. This range implies that avatars are either very isolated, e.g., they explore the region without meeting any other avatar, or they interact with a restricted set of avatars, e.g., they meet their friends. The highest percentiles of the node degree distribution, however, approach 300. Intuitively, avatars with such high node degrees are either extremely social or controlled by automated scripts. A further analysis of the traces shows that these avatars have the longest session durations as well, e.g., they are consecutively connected for about 90% of the crawl duration. This persistence suggests that their node degree simply represents the number of avatars they *meet*, rather than the number of avatars they *interact* with.

Given that social networks are often characterized by a power law distribution of their node degrees [65][108], we examine how well a power law fits the node degree distribution of G. As in other studies, we use the maximum likelihood method to calculate the best power law coefficient α as well as the lower bound x_{min} from which the law holds. Moreover, we use the Kolmogorow-Smirnow goodness-of-fit to evaluate the fit quality (D) [21].

Figure 3.17 plots the estimation of α, x_{min} and D for the node degree distribution computed on G every 15 minutes for the duration of the trace. During the first 24 hours, the estimations of α, x_{min} and D change rapidly. This phenomenon is due to the fact that we incrementally construct the social graph using the interactions that occur among avatars over time. Therefore, the initial hours simply represent a transient phase in the definition of the social graph. Subsequently, α slowly decreases to a value of 2.2 and D varies around 0.05, indicating that the power law well approximates the node degree distribution. The estimation of x_{min} oscillates around a degree value of 20–25, i.e., the power law fit is verified for about 25% of the avatars. Note that [20][65] verify the power law fit for about 10% of users. For comparison, observations from other networks indicate a value of α between 2 and 3 [4][10][48], whereas a value of α smaller than 2 is observed in most on-line social networks [112][65].

Figure 3.17 shows two high peaks in the estimation of both α and x_{min} at $t = 96$ hrs and $t = 108$ hrs, respectively. These times correspond with two short interruptions of the region service likely due to a server update. At those times the tail of the

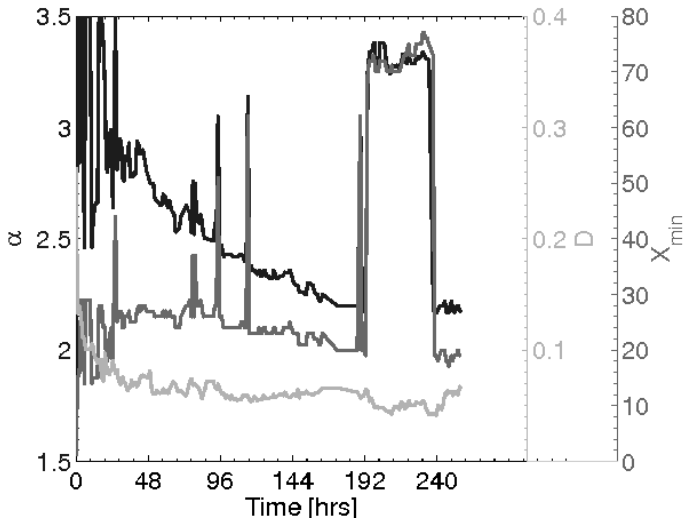


Figure 3.17: Power law analysis of the node degree over time for α, x_{min}, D.

node degree distribution becomes more skewed, causing a shift of the power law fit, i.e., higher x_{min} and α. This shift indicates that the avatars responsible for the very high values in the node degree distribution are the most responsive in re-connecting to the region when the service becomes available. The same phenomenon is even more visible at $t = 192$ hrs when the SL service returns after an outage of 20 hrs (Section 3.6.2). In this case, the node degree distribution is significantly impacted and α and x_{min} return to the values measured before the outage only after 48 hrs. This means that while the majority of users return to a region with some delay in presence of server failures, the high-degree nodes are much more aggressive. We conjecture that these high-degree nodes correspond to *bots* that reconnect to the region as soon as the service is available using automated probing.

Clustering Coefficient

The *clustering coefficient* is often used to characterize the extent to which nodes in social graphs form a small-world network. The clustering coefficient for a node $v \in V(G)$ is the ratio of the number of edges between the nodes within v's neighborhood[2] and the total number of edges that could possibly exist between them [108]. Figure 3.18 shows a scatter plot of a node's degree and its clustering coefficient for all nodes in the SL social graph at steady state (i.e., the graph constructed at time $t = 256$ hrs). We observe no clear relationship between a node's degree and its clustering coefficient for nodes with a degree less than 30 (about 90% of the nodes in G).

[2]The set of nodes in G sharing an edge with v.

Conversely, the clustering coefficient of nodes with a degree greater than 30 quickly decreases since the node degree value grows so high. Looking at the neighborhoods of these nodes in more detail reveals that, for 99% of them, the median value of the weight of their edges is smaller than 0.1, i.e., they spend less than 1% of their virtual time in close proximity to the avatars they contact. This results suggests that avatars associated with nodes with very high degree establish very fragile social relationships, again suggesting that these avatars are likely bots.

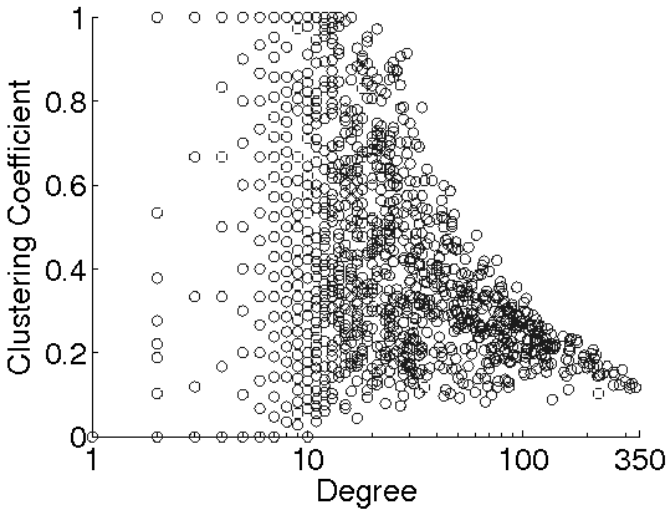


Figure 3.18: Node degree (log scale) vs. clustering coefficient ; $t = 256$ hrs.

Betweenness Centrality

The *betweenness centrality* of a node reflects its relative importance in a graph, indicating, for instance, how important a person is within a social network. Formally, the betweenness centrality for a node v in $V(G)$ is the number of times node v occurs in a shortest path between any two other nodes in $V(G)$ divided by all existing shortest paths in G [108]; the *shortest path length* for a node v in $V(G)$ is the minimum number of edges connecting v to all other nodes in $V(G)$.

Figure 3.19 shows several percentiles of the distribution of the betweenness centrality over time (the gaps again correspond to SL service outages). Figure 3.19 shows that nearly 90% of the nodes appear in less than 1% of all shortest paths. Interestingly, this result is similar to what Mtibaa et al. [67] observed when analyzing centrality in the social network constructed by human beings via a mobile social application. However, in the SL social network we also observe that about 5% of the nodes are very central and intersect with up to 10% of all the existing shortest paths. These very central nodes, though, also have high degrees and again most likely correspond to

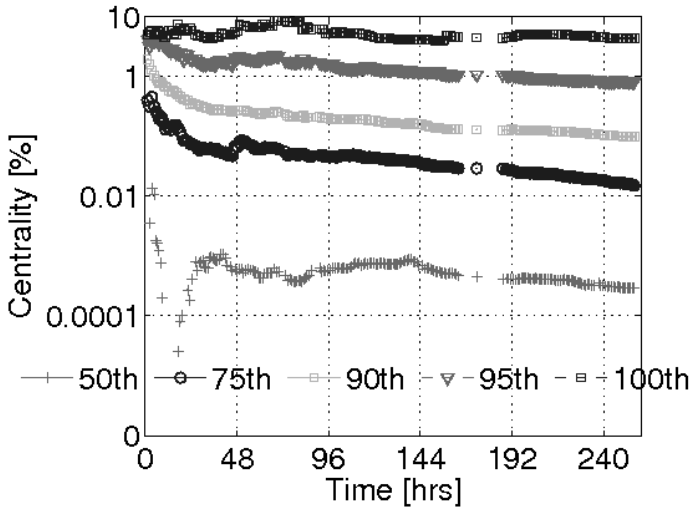


Figure 3.19: Percentiles of the betweenness centrality distribution over time.

bots. This results indicates that the presence of bots may have a significant impact on the structure of the social network.

3.7 Playing Second Life

So far, we have passively measured SL characteristics. This methodology does not allow us to measure the Quality of Experience (QoE) perceived by SL users, e.g., how fast and correctly users obtain information about avatars and objects located in their avatar surroundings. In fact, passive measurement misses two fundamental pieces of information to capture user QoE: (1) avatar locations at any point in time constructed according to real user inputs, and (2) the local view of each avatar, i.e., the evolution over time of the avatar and object updates each user receives from the server. We now adopt an *active* measurement strategy in order to collect this information. In the reminder of this Section, we describe how the measurement is performed, and we introduce the metrics to evaluate user QoE.

3.7.1 Experimental Setting

The key idea of our QoE measurement strategy is to replay avatar traces collected in a popular SL region over unused regions that we use as test-beds. We use our player in order to automate the behavior of an avatar within a test-bed region while collecting information about avatars and objects that intersect its AoI. We launch the

player from multiple Internet end-points and we teleport our controlled avatars into a target SL region. In this way, we capture the local view of each avatar through the traces collected by each player (Section 3.3.2). Moreover, we can easily build the "ground truth" at any point in time since we control all avatars that interact in the region. Note that we require that no external avatars, i.e., real SL users, interfere during our experiments to correctly evaluate user QoE. This is why we use unused regions as test-bed regions (remember from Section 3.4.6 that 30% of SL regions are unused).

We execute the SL player on several Planetlab [74] machines located worldwide in order to emulate realistic network conditions. We select stable Planetlab machines in terms of CPU load, free memory and network activity. We obtain real avatar mobility traces from the avatar traces in the Japan Resort region. We use the one hour period where we observe the maximum number of avatars, i.e., 84 concurrent avatars for a total of 207 different avatars during one hour, and for each avatar we generate its mobility trace. Due to a crawling frequency of about 30 seconds, we only have very coarse sampling of avatar movements. In order to reproduce fluid avatar movements in the region, whenever an avatar changes coordinates between two successive crawling snapshots we compute its speed and we interpolate its trajectory.

We run our experiments on three regions empty of avatars, and that represent the diversity in object composition observed in SL (Figure 3.7). We select respectively a *low density* (6 objects), a *normal density* (130 objects) and a *high density* (541 objects) region according to the number of objects they contain. During the experiments, we also continuously check that no external user connects to the region and interfere with our measurements.

Note that the presence of objects in these test-bed regions may cause avatars to be blocked in their movements, e.g., when they encounter a big building. We solve this issue by making our avatars deviate their mobility pattern and get around objects. While this strategy avoids having avatars stuck in the region, it also biases the avatar mobility pattern we are injecting in the regions. We analyze the impact of these modifications in avatar mobility when comparing user QoE in the three regions.

3.7.2 Metric Definition

Currently, game providers characterize user QoE by looking at the *cancellation rate*, i.e., the number of user accounts canceled during a given period of time, and/or

Mean Opinion Score[3] which is based on user feedback. Given that we cannot use these two techniques, we choose to compute user perception instead. Precisely, we compute three metrics that we formally define next: the *inconsistency*, the *inconsistency duration* and the *discovery latency*.

Inconsistency

In order to have meaningful interactions among avatars, each avatar needs to have correct information about the avatars contained in its AoI. Temporarily missing information as well as incorrect information make an avatar AoI *inconsistent*.

We call $\mathcal{A}(t)$ the set of avatars connected to a region at time t. For an avatar a, we denote with $ID(a)$ its identity and with $P(a,t)$ its coordinates in the region at time t. Remember that we know the exact location of each avatar at any point in time. Therefore, we can determine $AoI(A,t)$, i.e., the set of avatars that should be included in the AoI of an avatar A at time t, for any A and t. We define $AoI(A,t)$ as follows:

$$AoI(A,t) = \{a \in \mathcal{A}(t)\ s.t.\ dist(a,A) \leq 35\ meters\} \tag{3.1}$$

Each player continuously logs the information about avatars in its surrounding as informed by the region server. We use this data to compute $SAoI(A,t)$, i.e., the set of avatars that intersect A's AoI at time t, for $\forall A$ and $\forall t$, given the information transmitted by the region server.

The information contained in the AoI of an avatar A at time t is correct if the perception of A's neighbor avatars given the data received by the server (i.e., $SAoI(A,t)$) is congruent with the ground truth given by the mobility traces (i.e., $AoI(A,t)$). This happens if (i) the two sets $SAoI(A,t)$ and $AoI(A,t)$ are identical, and (ii) each avatar that intersects A's AoI has the correct identifier and coordinates given the ground truth defined by $AoI(A,t)$. More formally:

$$\begin{aligned} &(i)\ |AoI(A,t)| = |SAoI(A,t)| \\ &(ii)\ \forall a \in SAoI(A,t)\ \exists a^{'} \in AoI(A,t) \\ &s.t.\ ID(a) = ID(a^{'}) \wedge P(a,t) = P(a^{'},t) \end{aligned} \tag{3.2}$$

[3] http://en.wikipedia.org/wiki/Mean_opinion_score

We say that an avatar AoI is *inconsistent* when Condition 3.2 is violated. Violation occurs if (i) $AoI(A,t)$ and $SAoI(A,t)$ differ, i.e., when an avatar is in one set but not in the other one, or if (ii) an avatar is in both sets but not at the same position. Formally, we define the number of errors $Nerr(A,t)$ as:

$$\begin{aligned} &Nerr(A,t) =\mid \{(AoI(A,t) \cup SAoI(A,t))\ s.t. \\ &(a \in AoI(A,t) \wedge a \notin SAoI(A,t)) \vee \\ &(a \notin AoI(A,t) \wedge a \in SAoI(A,t)) \vee \\ &(a \in AoI(A,t) \wedge a^{'} \in SAoI(A,t) \wedge ID(a) = ID(a^{'}) \wedge \\ &P(a,t) \neq P(a^{'},t))\} \mid \end{aligned} \tag{3.3}$$

Then, the *inconsistency* is computed as:

$$\frac{Nerr(A,t)}{|(AoI(A,t) \quad \cup \quad SAoI(A,t)|} \tag{3.4}$$

The *inconsistency* as defined in 3.4 takes values between 0 and 1 where 0 means that all the information contained in an avatar AoI is correct, and 1 means that all the information contained in an avatar AoI is wrong.

Inconsistency Duration

User experience in virtual worlds is positive when avatars perceive quickly enough changes in the nearby avatar states [22]. This guarantees *interactivity* among avatars. We introduce the *inconsistency duration* as the time an avatar needs to achieve a consistent view of the avatars in its AoI. We compute the inconsistency duration by starting a timer whenever an inconsistency is detected in $SAoI(A,t)$ and stopping it as soon as $SAoI(A,t)$ becomes consistent again, i.e., it equals $AoI(A,t)$.

Discovery Latency

When an avatar moves and thus its AoI changes, the avatar may have to retrieve the information about the virtual objects located in its surroundings from the SL server. The *discovery latency* is the time an avatar needs to retrieve all virtual objects contained in its AoI. Therefore, the discovery latency holds similarities with the in-

consistency duration. The difference between the two metrics is that the discovery latency refers to objects, while the inconsistency duration refers to avatars.

We measure the discovery latency by parsing the traffic received by a player in order to isolate the packets that refer to the virtual objects. When a *region packet* (Section 3.2.4) is received, we detect that the server is currently updating the avatar AoI and we start a timer. We stop this timer when no more region packets are received, indicating that all objects that intersect the avatar AoI have been correctly received.[4] In case the avatar moves or leaves the region before it has correctly received a description of all virtual objects in its AoI, we simply stop and re-start the timer as soon as its position is stable again.

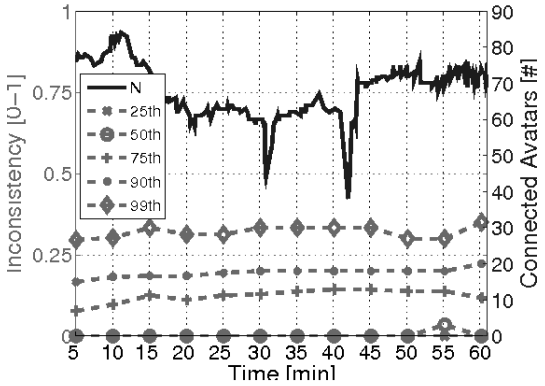
3.7.3 User QoE in Second Life

Inconsistency

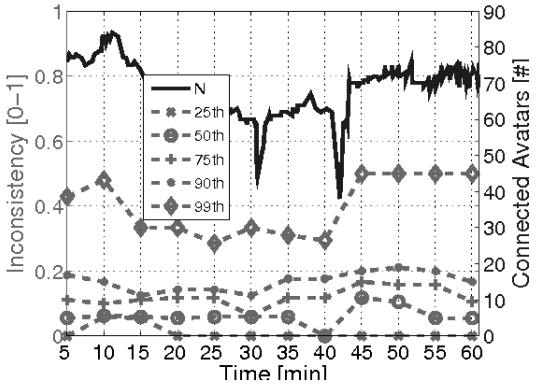
We evaluate the inconsistency for each avatar AoI every $200\ ms$ or anytime a modification of the AoI occurs. Figure 3.20 plots the evolution over time of the 25th, 50th, 75th, 90th and 99th percentiles of the inconsistency distribution among avatars. Each plot also shows the evolution over time of the number of avatars connected to the region.

Figure 3.20 shows that the inconsistency values measured in the low density region are very stable over time: 99% of the time avatars have less than 30% of the information contained in their AoIs that is not correct. This means that the low density region server is nearly not impacted by the variation in the number of concurrent users, and guarantees to its users a satisfactory level of correctness. Conversely, in both the normal and high density regions the inconsistency curves oscillate over time. SL users that interact in these two regions perceive a less correct experience compared to avatars in the low density region, e.g., the median value of inconsistency grows from 0.1 in the low density region to 0.4-0.5 in the high density region. This means that, as expected, the number of objects contained within a region has a significant impact on the user experience. In fact, due to the effort required for a SL server to manage virtual objects [58], the server has only few free resources to perform the visibility computation (Section 3.2.2) for each avatar. As a consequence, the server cannot guarantee to its users a satisfactory level of consistency.

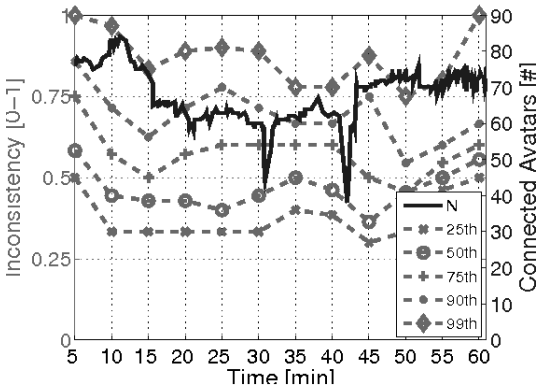
[4] Packets lost or corrupted trigger retransmissions at the SL servers.



(a) low density



(b) normal density



(c) high density

Figure 3.20: Percentiles of the avatar inconsistency over time.

The increase of inconsistency we measure for the high density region could also be due to the high density of objects that prevent an avatar to move exactly as indicated by the mobility traces (Section 3.7.1). For example, larger avatar groups may be created, causing additional load at the server and consequently an unfair comparison with the other regions. We compare the number of avatars intersecting each avatar AoI during the experiments. We find out that in the high density region avatars have on average less than 10% more avatars intersecting their AoIs than in respectively the low and normal density region. We believe that such a small variation of the avatar mobility does not justify the increase in avatar inconsistency we measured in the normal and high density regions.

We now want to understand how frequently inconsistency events affect an avatar during its journey in SL. Figure 3.21 plots the CDFs of the ratio between the sum of the durations of an avatar inconsistency periods and the total time the avatar stays in a region. We observe again that in the high density region, avatars are more inconsistent than in the other two regions. For example, in the high density region

more than 90% of the avatars suffer from inconsistency, whereas this number goes down to only 30% in both the low and normal density region. Interestingly, Figure 3.21 shows that avatars with an almost completely inconsistent SL experience are equally likely in the three regions, e.g., about 8% of the avatars have an inconsistent AoI during about 80-90% of their SL journey. The reason beyond this phenomenon is that the SL server spends a lot of time to correctly perform avatar login/logout as we will investigate next. Subsequently, avatars with very short session times have an inconsistent AoI most of the time.

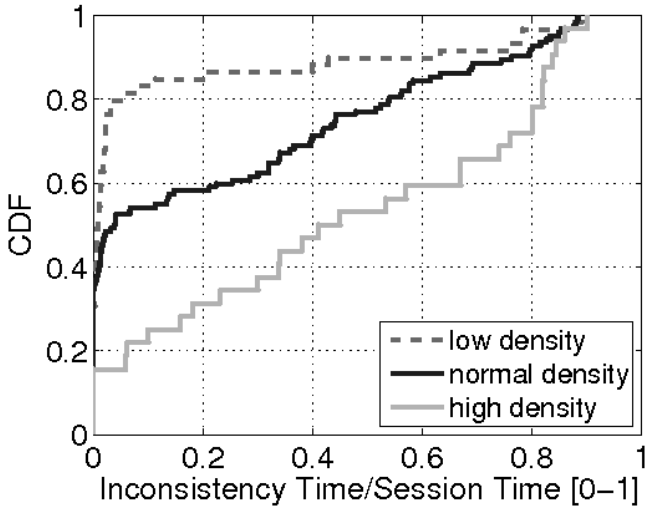


Figure 3.21: CDFs of the fraction of time an avatar is inconsistent.

Interactivity

We now analyze the inconsistency duration measured in the three regions in order to understand how fast SL servers react to avatar inconsistencies. Figure 3.22 shows the CDFs of the inconsistency duration values as measured in the three regions. We notice that avatar inconsistency lasts for more than one second in 40%-50% of the cases. Even worse, 5% to 10% of the inconsistencies last for more than 5 seconds. These inconsistency duration values are very long if we consider that acceptable latency values in virtual worlds vary between 300 ms and 1 sec [22], and indicate that SL servers provide poor interactivity to their avatars. Figure 3.22 also shows that the inconsistency duration measured in the three regions can reach a maximum value of about 20 seconds. These extremely high values are measured in presence of *churn*, i.e., avatar login and logout operations, and they are due to the fact that SL servers spend several seconds to accomplish avatar login/logout. Surprisingly, Figure 3.22 shows that the number of objects contained within a region does not impact the inconsistency duration, e.g., 50% of the inconsistency values measured

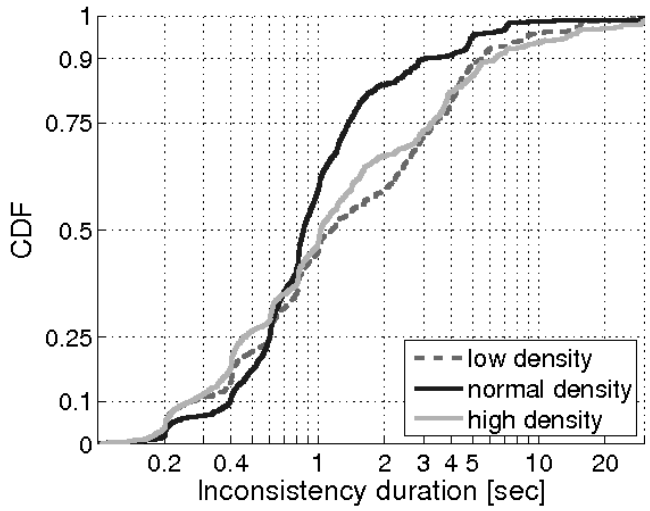


Figure 3.22: CDF of the duration of inconsistencies.

in the normal density region are lower than the values measured in the low density region. In the absence of official information form Linden Labs, we conjecture that SL servers choose, when over-loaded, to increase the number of inconsistencies but to reduce their durations.

Discovery Latency

Finally, we analyze how fast avatars retrieve the objects located in their surroundings. Figure 3.23 shows the CDFs of the discovery latency measured in the three regions. Not surprisingly, the higher the density of content in a region, the longer it takes for an avatar to reconstruct the virtual world in its AoI. The median discovery latency grows from about 4 seconds in the low and normal density region to about 30 seconds in the high density region. In our experiments, we verify that the user's network connection is not a bottleneck. Therefore, this result indicates that the longer discovery latency measured in the high density region is due to the fact that the server limits its outgoing traffic rate. However, Figure 3.23 shows also that the curves for the low and normal density region overlap for latency values larger than 20 seconds, i.e., high discovery latency values are more likely in the low density region than in the normal density region, which is counter-intuitive. This phenomenon is due to the fact that object density is not uniform in a region. Therefore, these high values of discovery latency measured in the low density region happen in portions of the region where the local density is much higher than the average object density of the normal density region.

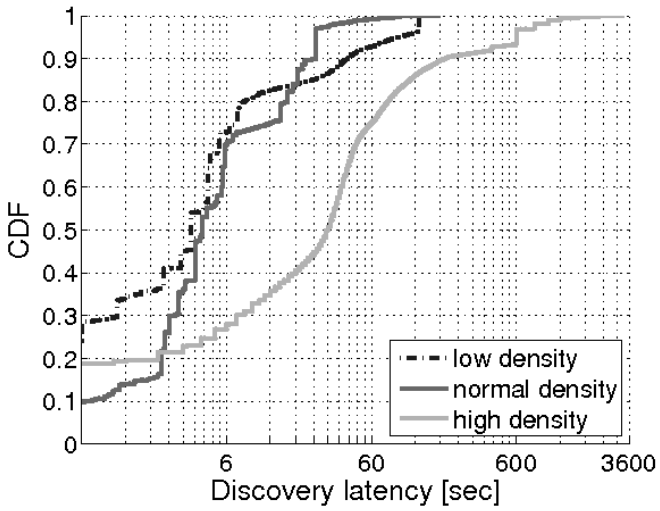


Figure 3.23: CDFs of the discovery latency.

Finally, Figure 3.23 shows that the maximum discovery latency measured in both the low and normal density region can reach a couple of minutes. Conversely, about 15% of the inconsistency values measured in the high density region are larger than one minute, and they can grow up to one hour, i.e., the entire length of the experiment. This result indicates that some avatars in the high density region are never able to correctly render the virtual world in their surrounding during their entire SL journey.

3.8 Conclusions

We have carried out a detailed evaluation of a large portion of Second Life (SL) and made some interesting observations. Almost 30% of the regions do not attract any visitors, and only few regions are quite popular. Moreover, the number of concurrent participants barely reaches 50,000. In comparison, World of Warcraft, a popular multi-player on-line game, reaches peaks of one million concurrent players. So one is tempted to paraphrase the famous American comedian W. C. Fields saying "I went to Second Life and it was closed". We also find that avatars exhibit a behavior that very much resembles that of humans: they get together in popular places, where they frequently meet their friends. Consequently, the SL social network is more similar to real life networks than to popular on-line social networks. From a systems perspective, we observe that SL shows poor scalability.

Our results also indicate that the quality of user experience in SL is generally poor: most of the time avatars have an inconsistent view of their surrounding, missing information about some avatars or visualizing them in an incorrect position. Moreover,

this inconsistency takes generally more than one second to be resolved. Finally, in a region crowded with virtual objects, avatars can spend tens of seconds to completely retrieve the description of the virtual world they are interested in.

Based on these observations, the next Chapter discusses the design and deployment of a *distributed object management* for NVEs. Our design is inspired by some characteristics of the SL virtual world. Moreover, we leverage some of the SL traces described in this Chapter in order to perform a realistic evaluation.

CHAPTER 4

Distributed Object Management

4.1 Introduction

In user-generated Networked Virtual Environments (NVEs) such as Second Life (SL), avatars can participate in the development of the virtual world by creating *objects* such as cars, trees, and buildings. Consequently, the persistent information about the virtual world, e.g., objects and land appearance, cannot be pre-located at the clients via CD distribution or BitTorrent like download [115] as it happens for the majority of the NVEs. Instead, data need to be transmitted to NVE users on demand according to their avatar locations in the virtual world.

The *object management* consists of maintaining a *persistent* copy of all virtual objects created in a NVE, while ensuring that NVE users have *consistent* information about the objects located within their avatar surroundings. Virtual objects need to be quickly transfered to the users in order not to affect NVE *responsiveness*. Nevertheless, the object management should be *scalable*, i.e., not affected by the number of objects and avatars that animate the NVE.

In a Client/Server NVE, the object management is a simple task. Servers store a copy of all virtual objects created on their portion of the virtual world, while clients interrogate the servers about the objects located within their avatar surroundings. This operation is a *range query*, i.e., a request for content whose attributes fall in a given range of the attribute spaces. The advantage of a centralized object management for NVEs is simplicity, while the limitations are scalability [58], high networking cost and slow responsiveness (cf. Chapter 3).

A distributed object management can be a scalable and cheap design for user-generated NVEs. However, maintaining object *consistency* and *persistency* in a Peer-to-Peer (P2P) network with unpredictable peer behaviors is a very challenging task. In addition, an efficient data storage and retrieval strategy must be designed in order to ensure good NVE *responsiveness*. Nevertheless, *security* might also be a issue.

This Chapter designs and evaluates a distributed object management for NVEs. Initially, we introduce a simple approach that can be easily integrated over any standard structured P2P network (Section 4.3). Then, we perform large-scale experiments over the Internet (Section 4.4.5) that allow us to understand how the underlying structured P2P network should be designed. We use this knowledge to design *Walkad* (Section 4.4), a structured P2P network based on the Kademlia protocol [63] that ensures fast *responsiveness* in user-generated NVEs. The avatar management as well as security issues are beyond the scope of this Chapter. Part of this work has been published in [100][105][103][104].

The distributed object management we design consists of dynamically partitioning the virtual world into *cells*, and assigning responsibility for those cells to peers organized in a structured Peer-to-Peer (P2P) network (Section 4.3), e.g., Distributed Hash Tables (DHTs) [63][81]. We integrate this distributed object management on the top of KAD, the DHT formed by eMule users [29]. Thanks to KAD, we perform realistic experiments on the Internet using the resources provided by eMule users. We emulate Second Life over KAD using avatar and object traces collected in a SL region by a crawler application (cf. Chapter 3).

We show (Section 4.3.7) that it is possible to construct a *consistent*, *persistent*, and *scalable* object management for NVEs on top of a structured P2P network already deployed over the Internet. However, in case of large number of objects, avatars can experience a long latency to recover from an inconsistent view of the virtual world. Moreover, search inefficiency in KAD introduces additional latency. Based on these observations, we design *Walkad* (Section 4.4).

The Walkad design derives from the following observation. Range queries in NVEs can be divided into *local* and *non-local*. A local query consists in a request for objects located in the avatar surroundings, e.g., avatars generate local queries when they walk, run or fly, in order to constantly update their visibility area. A non-local query is a request for virtual objects that are located far away from the avatar coordinates, e.g., avatars generate non-local queries when they suddenly cover a very large distance via the teleport operation. Intuitively, local queries are the most popular in NVEs [59]. For this reason, Walkad aims at assigning "close" cells of the virtual world to "close" peers in the Walkad network.

We design Walkad as an extension of the Kademlia DHT [63]. Walkad organizes the Kademlia key-space in a *reverse binary trie*, i.e., a tree-based data structure where nodes of each level of the tree are labeled using the Gray Code [36]. In this way, Walkad maps closeby cells to keys that are closeby in the Kademlia key-space. In other words, a Walkad peer responsible for a cell maintains routing information towards peers responsible for closeby cells. Local queries, originated by avatars moving from a cell to its neighbor cells, are quickly answered by "walking" across the Kademlia routing tables.

We evaluate Walkad via network emulation [98]. We build a Walkad network with up to 1024 peers and we use object traces from Second Life (cf. Chapter 3) to emulate a realistic virtual world. In addition, synthetic traces of avatar movements are used to study different types of range queries. Our results show that in a virtual world made of five Second Life regions indexed in a Walkad network of 1024 peers, local queries are answered in less than $150\ ms$ (on average), whereas non-local queries require about $200\ ms$. In addition, Walkad equally distributes the object load to peers as the network and virtual world sizes increase.

4.2 Background

Since we use KAD as a test-bed to evaluate a distributed object management for NVEs, we now briefly describe KAD and Kademlia, the routing protocol KAD is based on.

4.2.1 Kademlia

Kademlia [63] is a structured P2P network, where peers and content are identified by a random 160-bit identifier. Given two identifiers, a and b, Kademlia defines the *distance* between them as their bitwise exclusive or (XOR), i.e., $d(a;b) = a \oplus b$. This distance is calculated bitwise on the identifiers of the two nodes, e.g., the distance between $a = 1011$ and $b = 0111$ is $d(a,b) = 1011 \oplus 0111 = 1100$, and the distance between $a = 1011$ and $c = 1100$ is $d(a,c) = 0111$. Thus, a is closer to c than b, since $d(a,c) < d(a,b)$. The XOR distance metric is symmetric, i.e., if a is close to b, then b is also close to a. This means that peers in Kademlia learn useful routing information from the queries they receive.

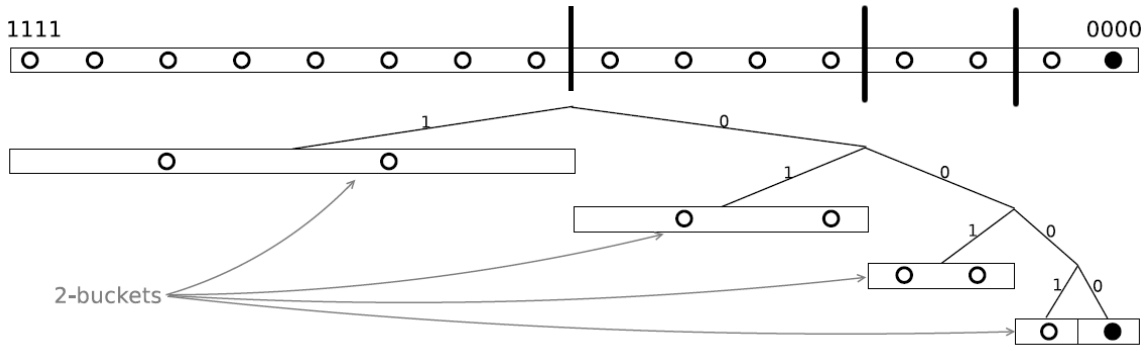


Figure 4.1: Kademlia k-bucket organization ; k=2.

A Kademlia peer keeps for each $0 \leq i < 160$ bit of its identifier a list of peers whose identifiers have XOR distance $2^i \leq d < 2^{(i+1)}$ from its identifier. These lists are called *k-buckets*, where k defines the maximum number of entries per bucket. The entries in the n^{th} k-bucket have a different n^{th} bit from the peer identifier. Each k-bucket is kept sorted by a most-recently-seen metric.

Figure 4.1 shows an example of the k-bucket organization for a peer P in a Kademlia network with $k = 2$ and identifiers composed by $b = 4$ bits. The top of Figure 4.1 shows the space of possible XOR distances from P's identifier. Accordingly, a distance of 0000 means that a peer's identifier has all four bits equal to P's identifier, i.e., the black dot in Figure 4.1 represents P's identifier, whereas a distance of 1111 means that a peer's identifier has all four bits that differ from P's identifier. Since peer identifiers are composed by four bits, peer P allocates four k-buckets. The routing information towards the $2^{(p-1)}$ peers whose identifiers differ from P's identifier in the first bit (e.g., 8 peers in the example of Figure 4.1) are stored in the first k-bucket. Since the k-buckets have size $k = 2$, only the information about two out of these $2^{(p-1)}$ peers can be kept. The other k-buckets are filled similarly. Figure 4.1 clearly shows that peer P has much more detailed information about peers whose identifiers are XOR close to P's identifier.

Routing in Kademlia is done iteratively and is based on prefix matching. A node a forwards a query, destined to a node b, to the node c in its k-bucket that has the smallest XOR-distance to b. Successively, node c performs the same operation and sends back to a the routing information towards b's closest node it knows. Each step allows to locate nodes that are closer to b until either b is found or no more closer nodes to b are available. This strategy allows Kademlia, like many other DHTs, to contact for routing only $O(log(n))$ nodes out of a total of n nodes in the network.

In order to *store* a <*key,value*> pair in Kademlia, a peer locates the k closest nodes to the key and sends them the value to be stored. Each peer re-publishes the

<*key,value*> pairs it manages every hour in order to ensure their *persistency*. In addition, the original publishers of a <*key,value*> pair re-publishes it every 24 hours. Otherwise, all <*key,value*> pairs expire 24 hours after the original publishing in order to limit stale information in the P2P network. Finally, in order to maintain consistency in the publishing-searching life-cycle of a <*key,value*> pair, whenever a node w observes a new node u that is closer to some of w's <*key,value*> pairs, w replicates these pairs on u.

In order to *find* a <*key,value*> pair, a peer performs a Kademlia lookup to locate the k nodes with IDs closest to the target key. The procedure ends as soon as any node returns the target value.

4.2.2 Kad

KAD is a Kademlia-based [63] DHT routing protocol that is implemented by several P2P applications such as Overnet [71], eMule [29], aMule [1], and Azureus [8]. In the following, we describe the main differences between KAD and the original Kademlia protocol.

In KAD, each peer is identified by a 128-bit KAD ID and routing is based on prefix matching, i.e., the smallest XOR-distance. Empirical measurements conducted over KAD [89] show that at most four routing hops are enough to reach any target node. This means that, since the KAD network has more than one million concurrent users [91], the k-buckets are very detailed and contain on average about 1,000 contacts.

KAD distinguishes between two different keys:

- A **source key** that identifies the content of a file and is computed by hashing the *content* of a file.
- A **keyword key** that classifies the content of a file and is computed by hashing a single token from the *name* of a file. A *metadata* is attached to a keyword key that contains information about a file such as its length, name and type.

The publication scheme in KAD differs from Kademlia, since the XOR minimum distance is not applied [88]. A key is published on 10 peers whose KAD ID agree at least in the first 8-bits with the key: this fraction of the key-space is called the *tolerance zone*. Since the KAD network is currently composed by millions of users, at any point in time there are about 10,000 nodes that fall in the tolerance zone of a given key [88].

As for the publication scheme, the republication scheme in KAD differs from the original Kademlia. Keys are simply periodically republished by their owners, source keys every 5 hours and keyword keys every 24 hours. Moreover, a node responsible for a given key does not republish the key to a new node identified as closest to this key.

4.3 A Simple Approach

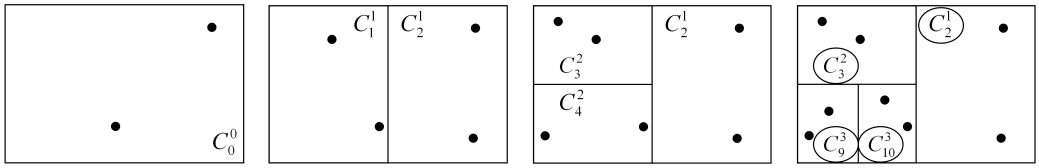
In this Section, we design and evaluate a simple distributed object management for NVEs. First, we describe the mechanisms we use to share the virtual world responsibilities among NVE participants organized as a structured P2P network. Then, we explain how we adapt our architecture to work on KAD as there is no possibility to modify the KAD routing algorithm. Finally, we perform a realistic experimental evaluation over KAD.

Note that we do not claim that this P2P communication infrastructure is completely innovative. It has been designed to allow realistic experiments on the Internet and to help us understand how to design an efficient P2P architecture for next generation NVEs. The innovation lies in the way we adapt KAD to handle the storage and retrieval of virtual objects of a P2P NVE.

4.3.1 Adaptive Cell-Based Management

We introduce here an *adaptive cell-based* partition of the virtual world. This simple mechanism is inspired from the scheme used by CAN [80] to dynamically assign the identifiers to its peers. In the context of NVEs, we use a similar approach to dynamically adapt the virtual world division to the object distribution.

We call a *cell* a portion of the virtual world. We denote a cell by C_i^l where l indicates that the cell is originated by the l-th split of the world, and i is a simple incremental index. We call $N_o(C_i^l, t)$ the number of objects contained within C_i^l at time t. Initially, the virtual space is identified by a single cell C_0^0. Then, C_0^0 is successively divided into multiple cells such that $C_0^0 = \bigcup_{\forall(i,l)} \{C_i^l\}$. Cells are split when the number of objects they contain exceeds a threshold D_{max}. Therefore, D_{max} is the maximum number of objects that can be contained within a cell. We call D_{min} the minimum number of objects contained within two adjacent cells.



(a) Initial Cell (b) First Split (c) Second Split (d) Third Split

Figure 4.2: Adaptive cell-based partition ; $D_{max} = 3$.

Whenever a user notices that $N_o(C_i^l, t) \geq D_{max}$, it performs a *split* operation that creates cells C_{2i+1}^{l+1} and C_{2i+2}^{l+1}. In the same way, when $(N_o(C_{2i+1}^{l+1}, t) + N_o(C_{2i+2}^{l+1}, t)) \leq D_{min}$, a *merge* operation is performed originating cell C_i^l. Split and merge operations are accomplished according to a well defined order. Assuming a two dimensional space, a cell is first split in the vertical dimension, then in the horizontal, and so on.

Figure 4.2 shows an example of the evolution of a cell-based two dimensional NVE with $D_{max} = 3$. In Figure 4.2(a), two objects are created in the original cell. Then, in Figure 4.2(b) two other objects appear. Given $D_{max} = 3$, the first split operation is performed. Figures 4.2(c) and 4.2(d) show how the adaptive cell-based management incrementally partitions the virtual world according to the distribution of objects.

4.3.2 Virtual Space and Distributed key-space

In structured P2P networks peers and content are each identified by a key from the same identifier space named the *key-space*. Each cell is assigned a unique identifier in the key-space called *cell-ID*. We now describe how cell-IDs are associated to cells of the virtual world. The mechanism we describe is inspired by PHT [79], a distributed data structure that enables range queries over DHTs.

We call k_i^l the cell-ID associated with cell C_i^l. The cell-IDs are organized into a l-level binary tree with $0 \leq l \leq (log(N_c) - 1)$ where N_c is the number of cell/cell-IDs in the NVE. Whenever a virtual cell C_i^l is split, two active cell-IDs k_{2i+1}^{l+1} and k_{2i+2}^{l+1} are derived as a function of k_i^l and associated with cells C_{2i+1}^{l+1} and C_{2i+2}^{l+1}. We associate k_{2i+1}^{l+1} to west (vertical) or north (horizontal) originated cells and k_{2i+2}^{l+1} to east (vertical) or south (horizontal) originated cells.

Given a cell-ID k_i^l, we derive k_{2i+1}^{l+1} and k_{2i+2}^{l+1} as follows. Let $\mathcal{H}$ be some hash function. For convention, $k_{2i+1}^{l+1} = \mathcal{H}(k_i^l)$ and $k_{2i+2}^{l+1} = \mathcal{H}(NOT(k_i^l))$. All the peers agree on a unique root for the tree, e.g., k_0^0, and on the hash function, e.g., $\mathcal{H}$=MD4 [17]. The hash function distributes cell-IDs uniformly in the key-space independently from

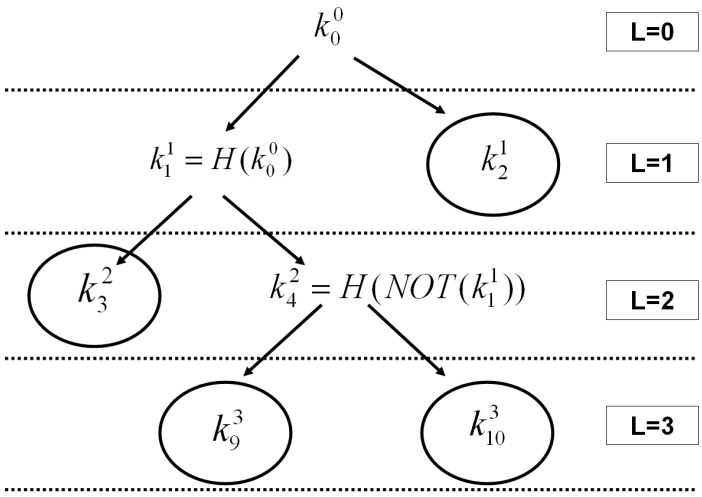


Figure 4.3: Cell-IDs organization.

the distribution of objects and cells in the NVE. Thus, load balancing is achieved among peers.

In order to follow the evolution of the NVE cell division, the cell-IDs are divided into *active* and *control* cell-IDs. We call an active cell-ID the identifier of a cell that represents an active portion of the world, i.e., a leaf in the tree. We call a control cell-ID, the identifier used to retrieve information about the cells organization of the virtual world, i.e., an inner node of the tree. Figure 4.3 shows the tree organization of the cell-IDs associated to the evolution of the NVE described in Figure 4.2.

4.3.3 Operations

Structured P2P networks such as Kademlia have three fundamental functionalities [24]:

- STORE – it allows to store a value in the P2P network.
- FIND_NODE – it allows to find the peer associated with a given key in the P2P network.
- FIND_VALUE – it allows to retrieve a value (if present) in the P2P network.

The participants of our P2P-NVE use these functionalities to perform additional operations related to the construction and management of the virtual world:

COORDINATOR_SELECTION - We call *coordinators* the peers that are responsible for the (cell/cell-ID) pairs of the NVE. NVE users select these peers using the

FIND_NODE function. For each (cell/cell-ID) pair of the NVE, R coordinators are selected. The rationale of this replication factor is to maintain consistency and persistency under churn.

INITIALIZATION - A bootstrap node identifies the R coordinators of the initial cell C_0^0 by performing a COORDINATOR_SELECTION for cell-ID k_0^0. At this time, these R peers are responsible for the entire virtual world. They store all objects created in C_0^0 and answer all range queries.

PUBLICATION - When an avatar creates an object in the NVE, we say that the object is *published* in the P2P network. An object located at coordinates (x, y) in the virtual world is published under the key k_i^l associated to cell C_i^l such that $(x, y) \in C_i^l$. The publication is done using the STORE function, i.e., by informing the R coordinators for cell C_i^l of the new created object. This means that each object in a cell is replicated R times at R different peers.

SPLIT - When a cell C_i^l is split into cells C_{2i+1}^{l+1} and C_{2i+2}^{l+1}, a special object called *pointer* is published in cell C_i^l through a PUBLICATION operation. In this way, an user that joins the NVE at any time is notified that k_i^l is a control cell-ID, i.e., that a split operation was performed.

RANGE_QUERY - An avatar in the virtual world needs to be constantly informed about the objects located in its *Area of Interest* (AoI). This operation is a range query with range equal to the AoI's size. A range query is performed via the FIND_VALUE operation requested to the coordinators for the (cell/cell-ID) pairs that intersect the avatar AoI. In case one of the cells that intersects an avatar AoI is a control cell-ID, it is used as the entry point in the tree. Then, the set of active cell-IDs is retrieved by going through the tree. We call this operation DISCOVERY.

DISCOVERY - This operation is performed when an avatar joins the NVE and every time it moves to a cell that has been previously partitioned. In each hop of the discovery, the cell coordinators must be found. Generally, in a structured P2P network this operation is logarithmic with the number of users N. In the worst case, a newcomer has to go through the entire tree, e.g., $log(N_c)$ hops are required where N_c is the number of cell/cell-IDs in the NVE. Therefore, the complexity of a DISCOVERY operation is $O(log(N) * log(N_c))$.

4.3.4 Implementation over Kad

We use the KAD API [17] in order to integrate our distributed object management with KAD. Note that we cannot modify the behavior of peers that participate to KAD. These peers are used as dumb nodes for an experimental purpose. However, we can modify the KAD API at the peers that we control. In the following, we explain: (1) how we integrate our distributed object management with KAD, and (2) how and why we modify the KAD API at our peers.

Control and active cell-IDs are coded into 120-bit KAD *keywords*. Object descriptions (e.g., coordinates and name) and pointer information (e.g., timestamp) are stored within the fields of a keyword metadata. In addition, a unique identifier is associated to objects and pointers of the NVE in order to distinguish them from the content inserted into KAD by eMule users.

The publication scheme implemented by KAD differs from the original Kademlia protocol (Section 4.2.2). Thus, the performance of our NVE could be reduced by KAD design choice. For this reason, we modify the KAD STORE function at our clients in order to use the XOR minimum distance rule. Precisely, whenever a user subscribes to a cell C_i^l it initially derives the set of peers in the tolerance zone of k_i^l as in the common KAD STORE function. Then, the user extracts the R coordinators for cell C_i^l by selecting the peers whose identifiers are the closest to k_i^l according to the XOR distance.

As for the publication scheme, the republication scheme in KAD differs from the original Kademlia. In order to ensure NVE consistency, we must ensure that at any point in time the R closest peers to a given cell-ID have the same information about the objects located in the cell. For this purpose, we modify the KAD republication scheme. We introduce a generalization of the Kademlia solution that we call the **delta publication**: whenever a peer p realizes that a peer Q is one of the R closest peers to a cell-ID k_i^l, p republishes on Q any objects $O_i(x, y) \in C_i^l$ it notices Q is missing. This means that when a peer p receives from a coordinator P the set of objects contained within cell C_i^l, it computes $\Delta(t) = (V_p(t) - (V_p(t) \cap V_P(t)))$ where $V_i(t)$ is the set of objects contained within the AoI of peer i at time t. Then, if $\Delta(t) \neq \emptyset$ p publishes onto P the set of objects $O_i(x, y)$ contained in $\Delta(t)$ (Algorithm 1).

Algorithm 1: The Delta Publication

```
1 foreach Coordinator P in C_i^l do
2     Δ(t) = (V_p(t) − (V_p(t) ∩ V_P(t))) ;
3     if (Δ(t) ≠ ∅) then
4         republish(P,Δ(t));
5     end
6     V_p(t) = V_p(t) ∪ (V_P(t) − (V_p(t) ∩ V_P(t))) ;
7 end
```

4.3.5 Experimental Methodology

We perform a realistic experimental evaluation of our simple distributed object management for NVEs in two steps. Initially, we design a client prototype that implements the functionalities described in the previous Section, i.e., KAD routing, cell management, avatar movement across the virtual world, and object insertion and lookup. Then, we use object and avatar traces collected in SL to emulate the behavior of a P2P version of SL. We reduce the SL objects to simple (name,coordinates) pairs, i.e., we do not consider any additional object attribute such as textures, in order to store them within the fields of the metadata associated to a KAD keyword.

We use our Second Life crawler (cf. Chapter 3) to collect avatar and object traces in the *Money Tree Island* region for a period of 10 hours. In our traces, we observe about 500 different avatars, with at most 90 concurrent avatars and at least 20 avatars in this region. 90% of the avatars barely move and visit less than 13% of the region. The most adventurous avatar traverses 65% of the region. We found a constant number of 586 objects in the region. While collecting the traces, we also measured that these conditions force the SL region server to slow down the simulation in about 50% of the times.

We emulate the activity of the Money Tree Island region by replaying its traces using our P2P client. We run several instances of the P2P client on a cluster of machines that participate to the KAD network. In this way, objects are stored on KAD nodes and searched through the Internet. A snapshot of each avatar AoI is copied to disk every $200\ ms$ or when a modification of its content occurs. We also introduce a generic user acting as a sniffer that we refer to as the *monitor*. The monitor can see the entire region. The task of the monitor is to constantly measure the availability of objects in the NVE, without interfering with the experiments.

One issue with this methodology is that we do not know the history of objects creation in the Money Tree Island prior to the monitoring period. Therefore, we create an *initialization phase* of duration T_i where avatars randomly insert new objects. Ob-

ject coordinates are extracted from the traces in order to be real. The initialization phase lasts $20\ min$. Once the initialization phase is completed, there is no object creation or deletion in the region. Therefore, all split operations are performed during the initialization phase and they do not impact our performance evaluation. This strategy is representative of the evolution of the object composition of most SL regions (cf. Chapter 3). The maximum number of objects per cell (D_{max}) is set to 20. The number of coordinators per cell varies from 5 to 20 (default value is 20). We approximate the AoI of each avatar as a circle centered on the avatar coordinates, and we vary its radius AoI_r between 5 and 35 units (default value is 35). With these parameters, we distribute the 586 objects of the Money Tree Island region across 44 cells and a tree of 9 levels.

The numbers above are (1) derived from observations in multiple SL regions (cf. Chapter 3), and (2) chosen to make the experimental analysis tractable.

4.3.6 Metric Definition

We now formally define the three performance metrics that we evaluate. For simplicity, we consider a two dimensional region, i.e., a Cartesian space $\Omega = [0, X_{max}] \times [0, Y_{max}]$, where X_{max}, Y_{max} are the maximum extension of the region along the x,y dimension. However, extension to three dimensional virtual worlds is straightforward. An *object* is a piece of content identified by its name and coordinates. We denote it with $O_i(x, y)$. We call $\mathcal{O}(t)$ the set of objects created within the region before time t. We call A some finite portion of the region. We call *state*, i.e., $S(t, A)$, the set of objects contained in an area A of the region at time t. We define $S(t, A)$ as follows:

$$S(t, A) = \{O_i(x, y) \in \mathcal{O}(t)\ s.t.\ (x, y) \in A\} \tag{4.1}$$

We call $AoI_i(t)$ the AoI of an avatar i at time t. We call $V_i(t, A)$ the set of objects seen by an avatar i at time t within an area A. Note that $V_i(t, AoI_i(t)) \subseteq S(t, AoI_i(t))$, i.e., an avatar may not see all the objects in its AoI due to inconsistency in the NVE.

Consistency: A NVE is consistent if at time t all the active avatars $\mathcal{N}(t)$ see the same set of objects (whether they exist or not). In order to define the consistency, we call

$SAoI_i(t)$, or shared $AoI_i(t)$, the portion of $AoI_i(t)$ contained in at least another $AoI_j(t)$ at time t. We define $SAoI_i(t)$ as follows:

$$SAoI_i(t) = \bigcup_{j \neq i} \{AoI_i(t) \cap AoI_j(t)\} \tag{4.2}$$

For a generic user i at time t we define the consistency of a NVE as follows:

$$K_i(t) = \frac{\mid (V_i(t, SAoI_i(t)) \cap \bigcup_{j \neq i} \{V_j(t, SAoI_i(t))\} \mid}{\mid \bigcup_{j \neq i} \{V_j(t, SAoI_i(t))\} \mid} \tag{4.3}$$

The *consistency* as defined in 4.3 takes values between 0 and 1. Consistency equals 0 for an avatar a when the objects contained within a's AoI are completely different from the set of objects seen by all the other avatars with same AoI as a. Consistency equals 1 for an avatar a when all objects contained within a's AoI are equal to the objects seen by all the other avatars with same AoI as a. In case that $SAoI_i(t) = \emptyset$, we consider that $K_i(t) = 1$. Note that the consistency is a user-dependent value.

Persistency: A NVE is persistent if no object gets lost during the evolution of the virtual world. If we assume that no object is removed from the NVE, the persistency is defined by the following property:

$$S(t, \Omega) \subseteq S(t^{'}, \Omega) \quad t^{'} > t \tag{4.4}$$

Scalability: the consistency and persistency of a NVE must not be affected by the number of concurrent users and objects. Therefore, we have identified two approaches to establish the scalability of a NVE. First, prove that the P2P layer is itself scalable. Second, analyze how increasing and reducing object density impacts the consistency and the persistency of the NVE. Given we use KAD, a well know P2P network that supports millions of users, we decided to focus the scalability analysis on the second issue.

4.3.7 Performance Analysis

We now present the performance results. Please note that it was not possible to compare the performance of P2P SL to the real SL as we cannot perform similar measurements on SL. The goal of this work is to show the feasibility of a distributed object management for NVEs, and to give us insights on how to design a dedicated P2P infrastructure.

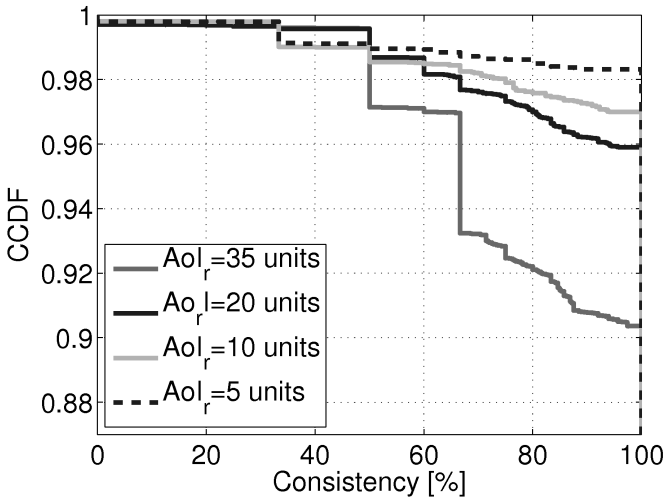
Region Consistency

Figure 4.4(a) shows the Complementary Cumulative Distribution Function (CCDF) of the consistency experienced by the avatars for different values of the AoI radius. For AoI radius up to 20 units, the region is perfectly consistent in 96% of the cases. For a more realistic value of 35 units, each avatar has a consistent view of the region 90% of the time. Intuitively, a larger AoI radius increases the discovery time and the likelihood of inconsistency.

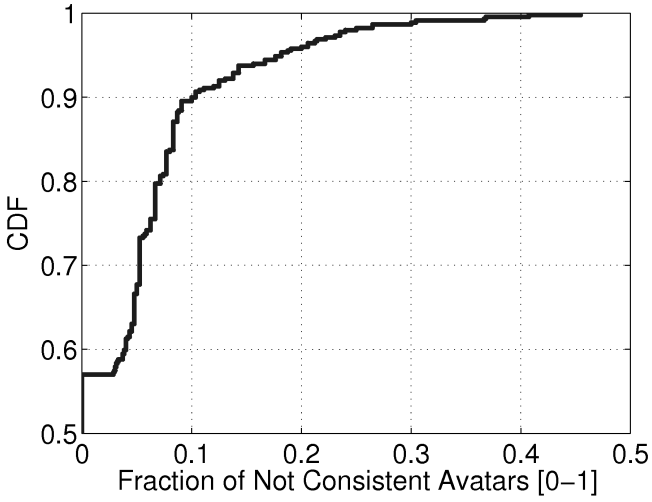
We identify three origins of avatars inconsistency: (1) avatar movements among cells, (2) avatars joining the region, (3) P2P hazards, i.e., Internet latency, peer churn and failures. Among the inconsistency values, avatar movements are cause of inconsistency in 45% of the cases, and they are responsible of consistency reduction from 99 to 65 percent (Figure 4.4(a)). In the same range of consistency values, accessing data in a real and large P2P network is a cause of inconsistency in 35% of the cases. Finally, the most dramatic inconsistency values (between 0% and 65%) occur when an avatar joins the region. These join operations cause 20% of the inconsistency values. In all cases, inconsistency is temporary and all avatars always succeed to reach 100% consistency after a while. The time it takes to come back to a consistent view of the region is discussed later in this Section.

Figure 4.4(a) shows three vertical steps respectively at 35, 50 and 65% of consistency. These steps indicate popular levels of inconsistency. When an avatar joins the region, it performs a DISCOVERY operation (Section 4.3.3). During this period, it only has a partial view of the region. Given that avatars tend to join a region always at the same locations (cf. Chapter 3), they miss more or less the same objects. A low level of consistency during the DISCOVERY operation is not really a problem as the effective join of a NVE user can be delayed until its view is consistent.

Inconsistency can either affect isolated avatars, or sets of avatars. We now measure the probability that a fraction of the avatars is inconsistent at the same time (Figure 4.4(b)). We notice that all avatars are perfectly consistent in about 55% of the cases,



(a) CCDF of region consistency ; $AoI_r = [35; 20; 10; 5]\ units$.



(b) CDF of the fraction of not consistent avatars ; $AoI_r = 35\ units$.

Figure 4.4: Region consistency analysis ; $R = 20$

and that only 35% of the time a maximum of 10% of the avatars is simultaneously inconsistent. Finally, the number of inconsistent avatars is always lower than half of the avatars. These results demonstrate that inconsistency is never related to the P2P architecture, as it only affects a small subset of the region population.

Replica Consistency

We now analyze the impact of the number of coordinators R on the consistency. By default each user selects 20 coordinators per cell, i.e., each object is replicated 20 times. In order to simulate several values of R during a single emulation, we perform the emulation with $R = 20$ and then reconstruct the view of the region for each avatar using different subsets of the number of coordinators.

Figure 4.5 shows an evaluation of the impact of R on region consistency. We notice that we achieve comparable consistency levels for 10 to 20 replicas per object, and 5 replicas is clearly not enough to maintain the region consistency. In order to investigate deeper the impact of the number of replicas on consistency, we compute the probability that avatars access different coordinator sets for a given cell. We observe that avatars contact the same set of coordinators only 50% of the time. This is due to churn in KAD, obsolete information in some nodes routing tables and avatars joining a cell (i.e., selecting the set of coordinators) at different times. In addition, we observe that for $R = 5$, there is a non negligible probability that all avatars in a cell get the objects from a totally disjoint set of coordinators. This explains the reduction of consistency observed in Figure 4.5. These results are clearly impacted by the KAD users session characteristics and could be different with P2P nodes being associated to SL users.

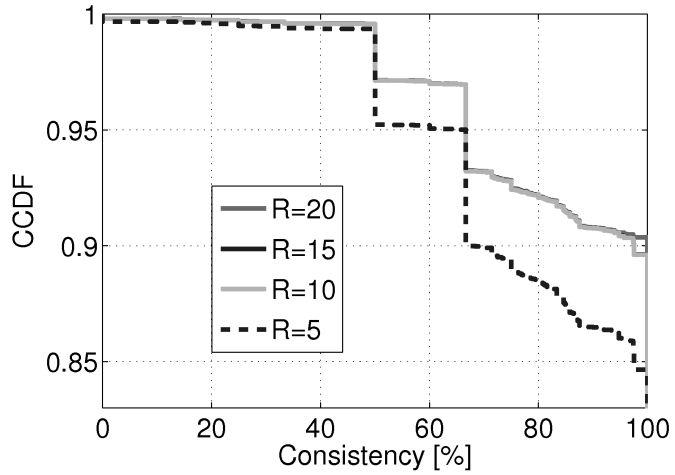


Figure 4.5: CCDF of region consistency ; $R = [20; 15; 10; 5]$; $AoI_r = 35\ units$.

Recovery Latency

We now investigate the time avatars spend in an inconsistent state in order to have a measure of the NVE *responsiveness*. Figure 4.6 plots the cumulative distribution

function (CDF) of the time it takes to an avatar to recover from an inconsistent view of the region. In the following, we refer to this time as the **recovery latency**. The recovery latency is smaller than 1 second around 35% of the times. 40% of the times, it takes to an avatar between 4 and 16 seconds to re-establish a consistent view of the region. About 20% of the times this latency can be longer of 16 seconds and reach couple of minutes. We recall that this latency is only related to the discovery of objects in a region, and it does not refer to avatar interactions.

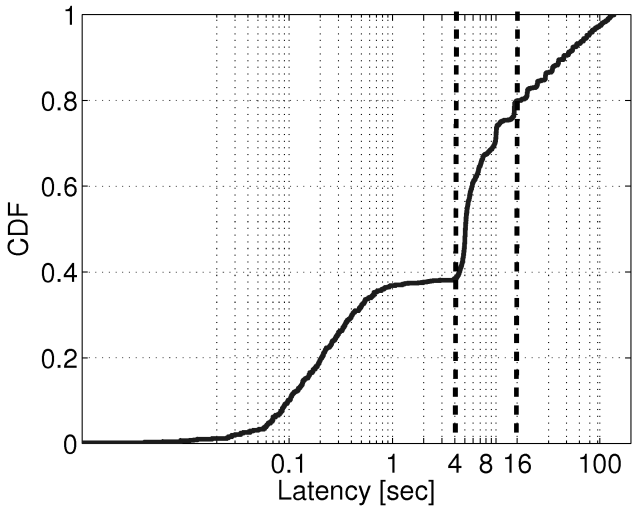


Figure 4.6: CDF of the recovery latency ; $R = 20$; $AoI_r = 35\ units$.

In Table 4.1, we assign the recovery latency to each cause of inconsistency identified earlier. Latency values lower than one second correspond to normal network conditions we observe in KAD. Latency values between 4 and 16 seconds are experienced by avatars that move as they often have to perform a lookup operation in KAD. We observe that this lookup time in KAD is generally close to 4 seconds, as also observed by Steiner et al. [89]. During the lookup, the objects contained in the new cell that falls in the avatar AoI are not yet visible to the avatar. In addition, in case the new intersecting cell is a control cell, a DISCOVERY operation has to be initiated. Therefore, multiple lookups in KAD are performed, and the recovery latency can reach 16 seconds. Finally, values between 16 and 100 seconds are observed by avatars who join the region. Ideally, we should never observe delays larger than 40 seconds as the cells tree organization in the emulation has a maximum depth equal to 9. However, in some cases, avatars join the region and immediately start to move across cells, e.g., they travel the region to reach some friends. This behavior causes avatars to change cell even before they obtain a consistent view of the current cell, with a dramatic impact on the recovery latency.

Cause	Percentage	Recovery Latency (sec)
Moving	45	4-16
P2P Hazards	35	≤ 1
Joining	20	16-100

Table 4.1: Causes of inconsistency.

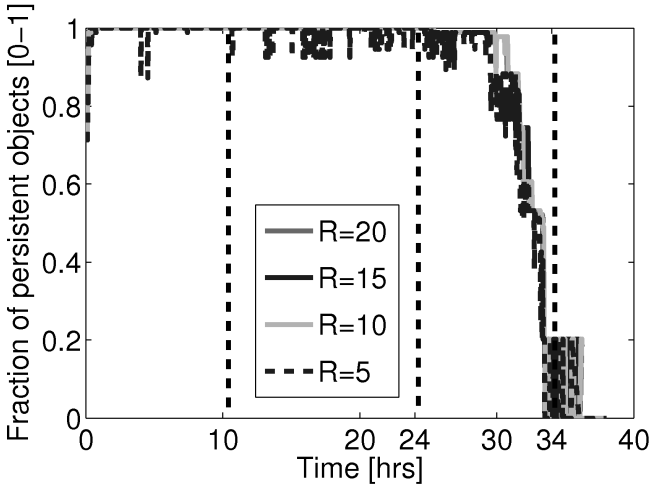
Persistency

Persistency characterizes the ability of our P2P architecture not to loose objects over time. Remember that after the initialization period where all objects are created, no more object appear or is removed. Moreover, a keyword in KAD is removed after 24 hours if not republished. Since we exploit KAD keywords to store SL objects, all objects should be discoverable by any avatar 24 hours after the last time they are re-published.

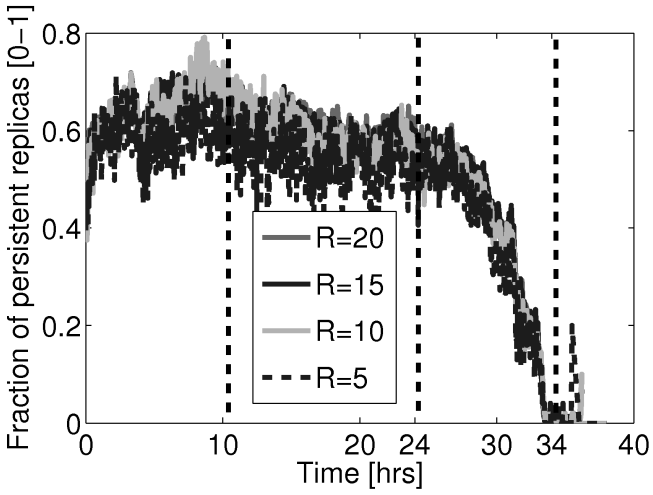
We measure region persistency through four monitors that systematically access all cells in the region and report statistics on the objects they contain. In order to evaluate the impact of a different number of coordinators R, each monitor contacts respectively 20,15,10 and 5 coordinators per cell.

In Figure 4.7, we plot the time evolution of respectively the fraction of persistent objects (Figure 4.7(a)) and the average fraction of persistent object replicas (Figure 4.7(b)). The vertical dotted lines at $t = [10; 24; 34]\ hrs$ in Figure 4.7 indicate time periods associated to interesting events that we explain in the following. Note that an object is persistent when at least one of its replica is available in KAD. The first 20 minutes correspond to the initialization phase. During this period, the cell organization changes frequently as new objects are created. This explains why the persistency grows from 0.7 to 1 in Figure 4.7(a). Once the initialization phase is completed, SL is perfectly persistent during the entire emulation (i.e., 10 hours) for $R = [20; 15; 10]$. For $R = 5$, we notice two "glitches" (i.e., persistency decreases) in the curves of both Figure 4.7(a) and Figure 4.7(b) at $t = 5\ hrs$ and $t = 6\ hrs$. At these times, the monitors could not find some objects in KAD as all their replicas had disappeared from the P2P network. The cause of this phenomenon is the difficulty to constantly maintain a set of consistent coordinators under the presence of churn, failures, etc. Anyway, in both cases the persistency goes back to 1 at the next measure performed by the monitors. This is because, as soon as an avatar observes that some objects have disappeared, it immediately performs a (delta) publication and persistency is recovered.

The emulation ends after $10\ hrs$, which means that objects are not republished anymore for $t > 10\ hrs$. Figure 4.7(a) shows that for $t > 10\ hrs$ the glitches in the curve



(a) Time evolution of the fraction of persistent objects



(b) Time evolution of the average fraction of persistent object replicas

Figure 4.7: Persistency analysis ; $R = [20; 15; 10; 5]$.

with $R = 5$ happen more frequently than in the first 10 hours. In addition, we observe a continuous decrease of the average fraction of persistent object replicas in Figure 4.7(b). This is due to the absence of the delta publication, which goal is to actively maintain a minimum number of replicas in the network. However, until $t = 24\ hrs$ the region remains perfectly persistent for $R > 5$. In addition, even for $R = 5$ SL persistency goes back to 1 from time to time. This behavior is explained by the presence of churn in KAD: even if objects are no more constantly republished, old coordinators that temporarily disconnected from KAD eventually come back restoring the persistency of the region.

The delta publication algorithm seems to be too conservative when the number of object replicas in the P2P network is larger than $R = 5$. In order to study how our P2P SL would behave without the delta publication, we study persistency in the $[10, 34]$ hours period where the delta publication is not active anymore. For this reason, we now analyze the behavior of the KAD nodes selected at least once to be coordinators in the $[0, 10]$ hours period. We compute respectively their continuous on-line time and availability to serve SL objects in the time interval $[10, 34]$ hours, i.e., the likelihood to be selected as coordinators. We observe that 80% of the coordinators have a continuous on-line time smaller than $3\ hrs$, and only 1% of the coordinators is on line during the entire time interval $[10, 34]$ hours. On one hand, this is due to the KAD user session characteristics [90]. On the other hand, it depends by the high level of churn in KAD [90] that causes frequent changes in the sets of cell coordinators. Precisely, 50% of the coordinators serve SL objects only for 10% of the time in the interval $[10, 34]$ hours. However, 15% of the coordinators serve objects for more than 60% of the time. These nodes guarantee persistency despite the presence of many unstable nodes.

Finally, we analyze the time interval $[24, 34]$ hours. Since objects in KAD are removed after 24 hours if not republished, in this time period we expect all objects to be deleted and the persistency of P2P SL to go to zero. As expected, the average number of objects replicas decreases quickly (Figure 4.7(b)). However, for $t \geq 34\ hrs$, 20% of the objects created are surprisingly still present in KAD. The reason is that some third-party eMule clients increase the lifetime of KAD keywords in order to reduce the volume of publishing/republishing operations [29].

The persistency results we have presented are impacted by the behavior of peers in the KAD network. We expect to observe a different behavior of peers in a structured P2P network maintained among avatars of a NVE. However, these results underline two strong features of the distributed object management. First, it can support a very high churn in the P2P network. Second, it is enough to have a low number of stable peers to maintain excellent persistency.

Scalability

We need to discuss two aspects of scalability. First, we must demonstrate that our P2P SL scales with an increase in the number of users. This is easy as our system inherits its scalability from KAD. Therefore, we decided not to perform any emulation with large number of participants as the scalability of KAD and Kademlia has been shown already [88][90].

objects	active cell-IDs	max tree depth
586	44	9
112	15	5
82	6	4
29	3	2

Table 4.2: Cell configuration.

Second, we evaluate the impact of the number of objects (which in turn impacts the number of cells) on the scalability of our P2P architecture. To do so, we perform four emulations varying the number of objects in the region, namely $N = [29; 82; 112; 586]$. Object locations are randomly chosen from the objects population of the Money Tree Island. The different configurations of cells and objects for the emulations are summarized in Table 4.2.

We analyze first the impact of the number of objects on the consistency perceived by the avatars. Figure 4.8 plots the CCDF of the consistency for different values of N. We observe that for $N = 29$, the consistency is nearly always equal to 100%. This is due to the fact that the limited depth of the cell tree organization reduces the impact of the join operation and consequently of the DISCOVERY operation. Moreover, avatars move less frequently among cells as cells have large extensions. Finally, we see that increasing the number of objects has a sub-linear impact on the consistency; increasing by one order of magnitude the number of objects causes a reduction of consistency by less than 10%.

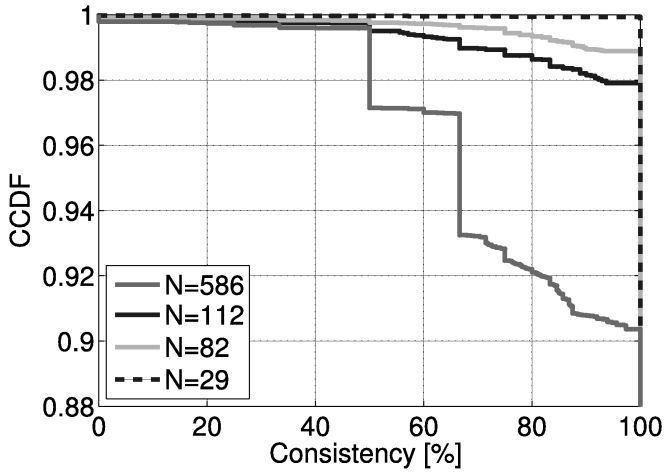


Figure 4.8: CCDF of region consistency ; $R = 20$; $AoI_r = 35\ units$; $N = [586; 112; 82; 29]$.

In Figure 4.9 we plot the CDF of the recovery latency for different values of N. There is not clear impact of the cell organization on latency values smaller than 1 second. In fact, these values only depend on the KAD nodes involved in the emulations and

on the network conditions. All curves meet at 4 seconds, which corresponds to the discovery latency of one cell as we have seen in Figure 4.6. For recovery latency values larger than 4 seconds, we observe a clear impact of the different complexity of the cell organization in the four emulations. For $N = 29$, 90% of the recovery latency values are smaller than 8 sec. Since the maximum depth of the tree is two, 8 seconds correspond to the time duration of a DISCOVERY operation of length two in the tree. Then, the depth of the tree increases with N. This in turn increases the probability to experience a larger recovery latency. In particular, we notice that when the recovery latency is larger than 4 seconds, increasing the number of objects by one order of magnitude causes a latency increase of about 30%. This result shows that the duration of lookup operations in KAD limits the scalability of our P2P SL.

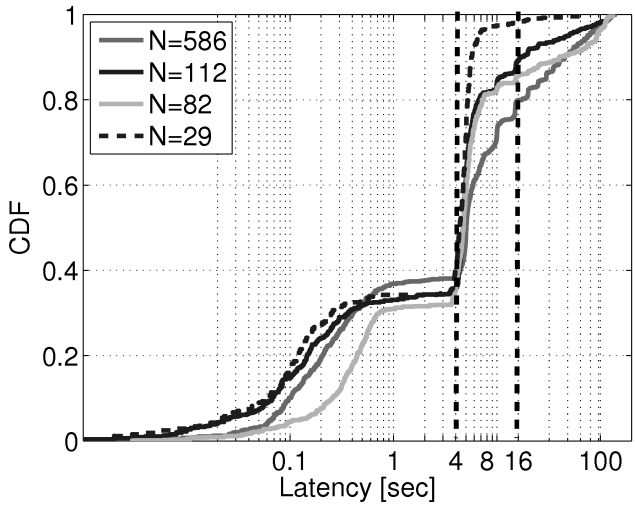


Figure 4.9: CDF of the recovery latency ; $R = 20$; $AoI_r = 35$ $units$; $N = [586; 112; 82; 29]$.

4.4 Walkad

The long latency we experience running Second Life over KAD is also due to the fact that KAD like all other DHTs, use a hash function to distribute content fairly among peers. This design is very efficient to build P2P lookup systems where content is only addressed punctually. Conversely, this design is not efficient for P2P-NVEs where users address content within a range, e.g., they need to discover all the objects located in their AoIs. Based on this observation, this Section introduces *Walkad*, a structured P2P network specifically designed to store and retrieve virtual objects in NVEs.

4.4.1 The Gray Code

The Walkad key indexing algorithm leverages the *Gray Code* [36]. This is a binary numeral system where two successive values differ in only one digit. The $(n+1)$-bit Gray Code is constructed as follows: (1) *reflect* the 2^n values of the n-bit Gray Code, i.e., list them in the reverse order, (2) prefix the original values with a bit set to 0, and the reflected values with a bit set to 1, (3) concatenate the reverse list to the original list.

Figure 4.10 shows the construction of the *3*-bit Gray Code. On the left portion of Figure 4.10, we can see the *1*-bit Gray Code, i.e., the most basic Gray Code, $G = \{0, 1\}$. In order to construct the *2*-bit Gray Code, we reflect $G = \{0, 1\}$ obtaining $G' = \{1, 0\}$. Then, we prefix the original values (G) with a 0, and the new values (G') with a 1, obtaining the *2*-bit Gray Code, $G = \{00, 01, 11, 10\}$. The same procedure is applied to construct the *3*-bit Gray Code.

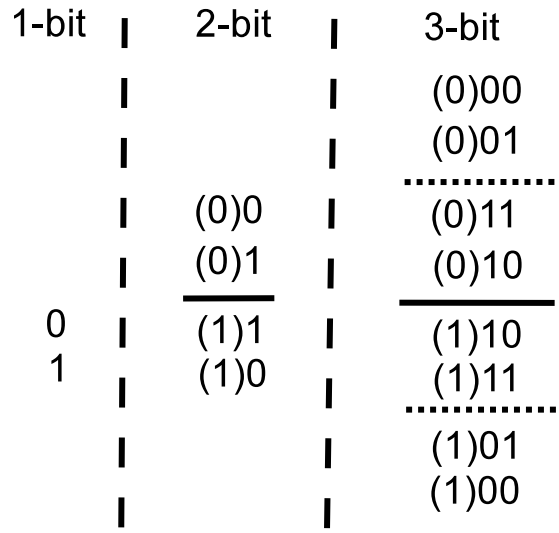


Figure 4.10: Example of the Gray Code construction.

4.4.2 Key Indexing

We first define the notion of "locality" in a cell-based virtual world as described in Section 4.3.1. We say that two cells are **neighbor** cells if: (1) they are adjacent, i.e., they have a side in common, or (2) they are symmetric according to the axis used in previous split operations. We say that two cell-IDs are **neighbors** when their cell-IDs have a *Hamming* distance of one, i.e., when they differ only by one bit. By definition, a cell-ID with l significant bits has l neighbor cell-IDs. To illustrate this, we consider

an example of a one-dimensional virtual world (Figure 4.11). We denote the *i*-th cell/cell-ID generated by l splits respectively as C_i^l and k_i^l.

The top portion of Figure 4.11 shows the initial cell organization. At this stage, there is only one cell, C_0^0, that covers the whole virtual world. The middle part of Figure 4.11 shows the virtual world configuration after the first split, where two new cells are created, C_1^1 and C_2^1. These two cells are obviously neighbors. The bottom part of Figure 4.11 shows the result of splitting again both cells, obtaining cells C_3^2, C_4^2, C_5^2 and C_6^2. Let's consider cell C_3^2. Its neighbor cells are cell C_4^2 which is adjacent and cell C_6^2 which is symmetric to C_3^2 according to the long dashed line crossing the middle of the line (indicating a previous split). A generic cell C_i^l generated after l split operations has l neighbor cells in the virtual world.

Walkad organizes the cell-IDs in a *reverse binary trie* in order to associate neighbor cell-IDs to neighbor cells. A binary trie is a tree-based data structure that uses prefix bits to direct branching in a tree. Conventionally, a 0 represents a left branch and a 1 represents a right branch. Each node in the trie is associated to a label composed of the set of bits indicating the path in the trie to reach that node. In a reverse binary trie, the nodes of each level of the trie are labeled with the Gray Code [36] (Section 4.4.1).

We now explain how we organize the cell-IDs in a reverse binary trie considering the example of Figure 4.11. For convention, cell C_0^0 is assigned cell-ID $k_0^0 = 0*$. When C_0^0 splits, two neighbor sub-cells are created. We generate the corresponding cell-IDs by taking $k_0^0 = 0*$, and setting the least significant bit respectively to 0 and 1. We thus obtain two cell-IDs, $k_1^1 = 0*$ and $k_2^1 = 1*$, which have a Hamming distance of one. The intuition is that to build a reverse binary trie in some cases we need to reverse the added bits (i.e., add a 1 to the left cell-ID). Figure 4.11 shows that splitting cell

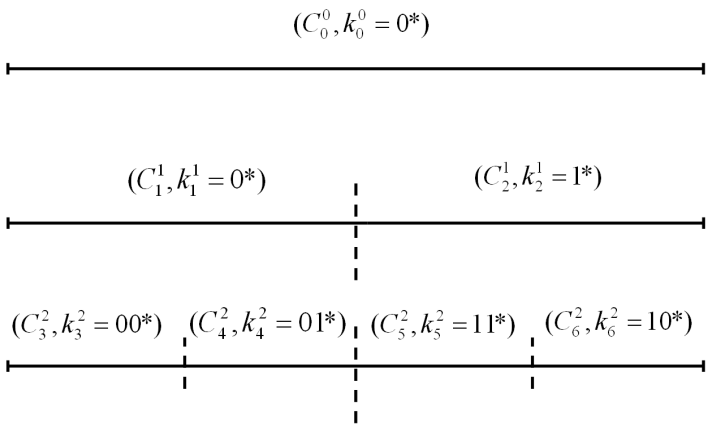


Figure 4.11: 1-dimensional virtual world indexed by Walkad.

C_2^1 required setting the least significant bit to 0 for $k_6^2 = 10*$ and to 1 for $k_5^2 = 11*$ in order to guarantee that cells C_3^2 and C_6^2, which are neighbors, are assigned neighbor cell-IDs as well. We see that the code generated at level 2 of the trie is the 2-bit Gray Code.

The generalization to the multi-dimensional case is straightforward, as the cell split mechanism is applied independently to each dimension of the virtual world.

The Walkad indexing algorithm as described above generates a distribution of cell-IDs within the key-space that follows the shape of the trie. Therefore, an unbalanced trie will result in an unbalanced distribution of cell-IDs and so of load among peers. In order to restore the uniform distribution of the cell-IDs, we divide the world into regions (as in Second Life), and we allocate to each region a *region-ID*. Then, we perform a XOR operation between the cell-IDs and the region-ID. In this way, the Hamming distance property defined among cell-IDs of the same region is maintained and load balancing is achieved among cell-IDs of different regions.

4.4.3 Operations

Walkad leverages Kademlia without requiring any changes to its routing algorithm. However, similarly to Section 4.3.3 some additional operations are required to construct and manage the virtual world. Here below, we re-define some of these operations and we define few new ones that are specific to Walkad. Note that the INITIALIZATION and PUBLICATION operations (Section 4.3.3) remain unchanged and so we do not discuss them.

COORDINATOR_SELECTION The coordinator for a cell C_i^l indexed by cell-ID k_i^l is selected using the Kademlia FIND_NODE operation. Accordingly, a coordinator for a cell C_i^l is the XOR closest peer to k_i^l. For each (cell/cell-ID) pair there are R coordinators, i.e., each object and consequently cell is replicated at R peers.

SPLIT - When a cell C_i^l is split in cells C_{2i+1}^{l+1} and C_{2i+2}^{l+1}, its coordinators do the following operations: (1) select the coordinators for C_{2i+1}^{l+1} and C_{2i+2}^{l+1} by performing a COORDINATOR_SELECTION for cell-IDs k_{2i+1}^{l+1} and k_{2i+2}^{l+1}, (2) transfer to the coordinators of C_{2i+1}^{l+1} and C_{2i+2}^{l+1} the list of C_i^l neighbor cell/cell-IDs , (3) distribute the virtual objects currently located in C_i^l to the coordinators of cell C_{2i+1}^{l+1} and C_{2i+2}^{l+1} according to the object coordinates. Note that a merge operation is done similarly.

COORDINATOR_TASKS - A peer selected to be a coordinator for a cell C_i^l with cell-ID k_i^l does the following operations: (1) derive the l cell-IDs with Hamming distance

equal to 1 from k_i^l, (2) compare each of these cell-IDs with the list of cell-IDs received during the SPLIT in order to identify the existing neighbor cell-IDs, (3) perform a COORDINATOR_SELECTION operation for each existing neighbor cell-ID to populate its k-buckets with the routing information towards the neighbor coordinators, (4) inform the coordinators of the neighbor cells that a new cell was created.

RANGE_QUERY - We suppose that a peer P that generates a range query already knows the R coordinators of the cell where its own avatar is located. P sends the range query to one of these coordinators. The coordinator answers the query or a portion of it according to the information it has about the neighbor cells. Then, it sends back to P the information it may know, i.e., routing information towards the coordinators for the cells that intersect with the query's range. In case a coordinator has not a complete view of the entire range, it forwards the query to the coordinators it knows that manage the closest cells to this range. Intuitively, these coordinators have a more detailed view of this portion of the virtual world. This procedure is done iteratively until the range is completely covered. Finally, P directly contacts the set of coordinators responsible of the query's range in order to retrieve the information about the virtual objects located in this portion of the virtual world.

We now give an example of how a range query is performed in Walkad considering the virtual world described in Figure 4.11. We consider a peer P whose avatar is located in cell C_4^2 and the query's range to be contained in cell C_6^2. Peer P submits its query to a coordinator for cell C_4^2. This coordinator cannot solve the query, so it identifies cell C_5^2 as the closest cell to the query range. Therefore, it searches in its k-buckets the routing information towards the R coordinators for cell C_5^2 and selects a coordinator to whom it forwards the query. The selected coordinator for cell C_5^2 has a more detailed view of this portion of the virtual world. So, it identifies C_6^2 as the cell that contains the query's range. The query is now solved and the coordinator of C_5^2 can send to P the list of coordinators for cell C_6^2. Peer P can now directly contact the coordinators of cell C_6^2 and download the description of the objects located within the query's range.

4.4.4 Cost Analysis for Range Queries

We now analyze the cost of range queries in Walkad in terms of number of routing hops. Let k be the size of a k-bucket and R the number of coordinators per cell. N is the number of active peers and N_c the number of cells composing a region of the virtual world. We assume $N >> (R * N_c)$ such that each cell-ID is stored at R different coordinators.

In case of a uniform distribution of cells within the world, all the leaves of the trie are at the same level l. In this case, every cell has l neighbor cells associated to l different cell-IDs. Therefore, local queries are answered in a single routing hop, while non local queries require $log(N_c)$ routing hops.

Figure 4.12 shows an example of a two-dimensional virtual world composed by a uniform distribution of cells. The arrows indicate neighbor (cell/cell-ID) pairs as well as routing information, e.g., the coordinators for cell-ID $001*$ keeps routing links towards the coordinators for cell-IDs $000*$, $011*$ and $101*$. We clearly see that all local queries require a single routing hop. Let's consider a non-local query, e.g., an avatar P located in the cell identified by the cell-ID $001*$ that wants to teleport to the cell identified by cell-ID $110*$. In this case, the query goes through the coordinators of cell-IDs $101*$, $111*$ and $110*$ to be solved, i.e., it requires $log(N_c)$ routing hops.

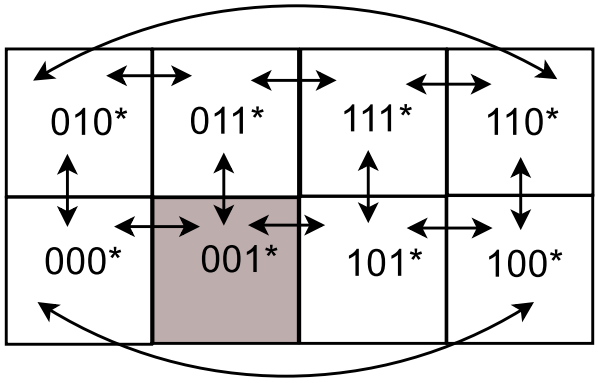
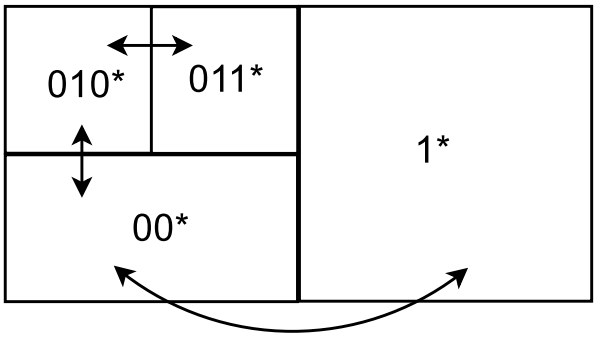


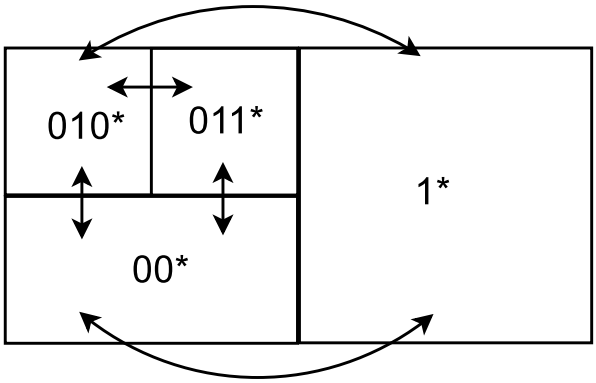
Figure 4.12: Uniform cell division of a 2-dimensional virtual world.

A skewed distribution of cells within the virtual world results in an *unbalanced* trie. In this case, a cell that is close to the root of the trie may have more neighbor cells than neighbor cell-IDs. Let's consider two cell-IDs k_i^l and k_j^f respectively located at level l and f of the trie with $f < l$. In this case, there are $R * 2^{(l-f)}$ coordinators "colliding" on the same k-bucket of the coordinators for cell-ID k_j^f. If $k < R * 2^{(l-f)}$, the coordinators of k_j^f select only a subset of the $2^{(l-f)}$ neighbor cells to maintain a direct route towards their coordinators.

Figure 4.13 shows an example of a two dimensional virtual world composed by a skewed distribution of cells. The arrows indicate routing links among the coordinators, respectively in the case of $k = R$ (Figure 4.13(a)) and $k = 2R$ (Figure 4.13(b)). We can see that the neighbor cells for the cell identified by cell-ID $1*$ are the cells indexed by $011*$ and $00*$, which are adjacent, and $010*$ which is symmetric to the vertical split. However, by definition, cell-ID $1*$ has only a single neighbor cell-ID that is $0*$. Therefore, if $k = R$ the coordinators for cell-ID $1*$ keep a single link to-



(a) k=R



(b) k=2R

Figure 4.13: Skewed cell division of a 2-dimensional virtual world.

wards the coordinators for cell-ID $00*$, whereas if $k = 2R$ these coordinators keep two links towards the coordinators for cell-ID $010*$ and $00*$.

We now derive an expression for the number of routing hops in Walkad in case of an unbalanced trie. Let's consider again two arbitrary cell-IDs k_i^l and k_j^f located at level l and f of the trie. In case k_i^l and k_j^f are neighbor cell-IDs, a query from k_i^l to k_j^f or vice-versa is local and requires $\frac{(|l-f|)}{\lfloor log(\frac{k}{R}) \rfloor}$ routing hops. In case k_i^l and k_j^f are not neighbor cell-IDs, the query is non-local and requires an intermediate step at a cell-ID k_k^c that has routing information towards the destination cell-ID. Therefore, the number of routing hops is equal to $\frac{(|l-c|+|c-f|)}{\lfloor log(\frac{k}{R}) \rfloor}$. Note that in the worst case the number of routing hops is $O(N_c)$ for both local and non-local queries.

In conclusion, the number of routing hops in Walkad varies between $O(1)$ and $O(N_c)$ according to the skewness of the cell distribution and the type of range query. Intuitively, prefix expansion [99] can be used to reduce the number of routing hops in case of a very unbalanced trie. For comparison, the simple approach of Section 4.3

and P-Grid [2] require respectively $log(N) * log(N_c)$ and $log(N)$ routing hops for both local and non-local queries.

4.4.5 Experimental Evaluation

We cannot evaluate Walkad using the KAD network as a test-bed (Section 4.3.5) as this would require to change the behavior of KAD peers. Therefore, we evaluate Walkad using network emulation. We deploy up to 1024 peers (i.e., avatars) on a local cluster, and we use Modelnet [98] to emulate wide-area latencies and bandwidths. We use a synthetic Internet topology generated by Inet [113]. We use a classic Kademlia setup with k-bucket size $k = 20$ and $R = 10$ [63].

We construct a realistic virtual world using object locations from five popular Second Life regions (cf. Chapter 3) that contain respectively a minimum of 70 objects and a maximum of 350 objects. A bootstrap Walkad node computes the virtual world division in cells, and informs all coordinators of their role. We use synthetic traces for avatar movements generated via the Random Waypoint Mobility model [66] with different speeds to simulate avatar *walking* (1 m/s), *running* (3 m/s), *flying* (10 m/s) and *teleporting* (100 m/s). The avatar traces last for one hour. We use synthetic traces for avatar movements and not real avatar traces in order to analyze Walkad performance under controlled avatar behaviors.

In the evaluation, we do not focus on consistency, persistency and scalability as these metrics have been already extensively evaluated over KAD (Section 4.3.5). Conversely, we focus on routing hops and latency to answer range queries, and on load balancing properties. For comparison, we also present some results obtained with the simple approach (Section 4.3) implemented over Kademlia.

Routing Hops and Latency

We analyze first the number of routing hops as a function of D_{max}, the maximum number of objects per cell. By varying D_{max} we simulate different divisions of the virtual world. We consider a Walkad network composed by 1024 peers and a single avatar walking in the virtual world. This means that all range queries are local and the "load" in terms of concurrent number of queries in the network is small.

Figure 4.14 shows that 90% of the local queries in Walkad require only one or two routing hops to be solved. This percentage becomes even larger as we increase D_{max}, e.g., when $D_{max} = 100$ all local queries are answered in a single hop with no excep-

tion. In fact, increasing the number of objects per cell systematically results in a more simple cell organization of the virtual world. However, if we focus on the results obtained with $D_{max} = [30; 10]$ we notice that the number of routing hops is comparable or even smaller for $D_{max} = 10$, which seems contradictory. The cause of this phenomenon is that for $D_{max} = 10$, the condition $N >> (R * N_c)$ is not verified. Therefore, peers are coordinators of multiple cells, and the Walkad indexing algorithm tends to aggregate closeby cells on the same peer. The side effect is a reduction in the number of effective routing hops especially for local queries.

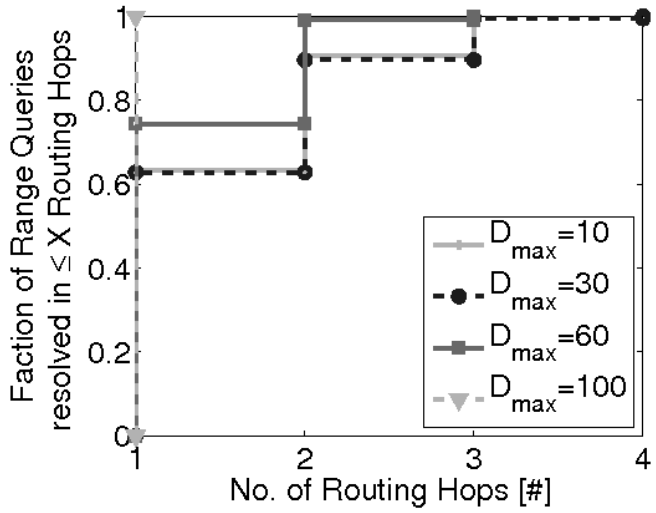


Figure 4.14: $D_{max} = [10; 30; 60; 100]$; $N = 1024$; Avatar=$[Walk]$.

We now evaluate the *latency*, i.e., the time required to answer range queries, as a function of the network size N and type of range query (Figure 4.15). In order to generate different range queries, we consider a single avatar walking, running, flying and teleporting in the virtual world. Figure 4.15 plots also the latency values for the simple approach. For the simple approach, we only consider the case of an avatar walking as the type of range query does not impact the way routing performs.

Figure 4.15 shows that range queries generated by an avatar walking, running or flying are all resolved in about the same time, i.e., $100 - 130\ ms$ in average. In fact, all these movements generate local queries. Conversely, non local queries generated by an avatar teleporting in the virtual world require about twice the time, e.g., up to $200\ ms$. For comparison, the average latency for the simple approach is between $800\ ms$ and $1500\ ms$, i.e., 8 times larger than in Walkad. Figure 4.15 shows also that the overall latency only slightly increases with the size of the network N. In fact, the number of routing hops to solve range queries in Walkad depends on the size of the virtual world rather than on the size of the network (Section 4.4.4). However, when

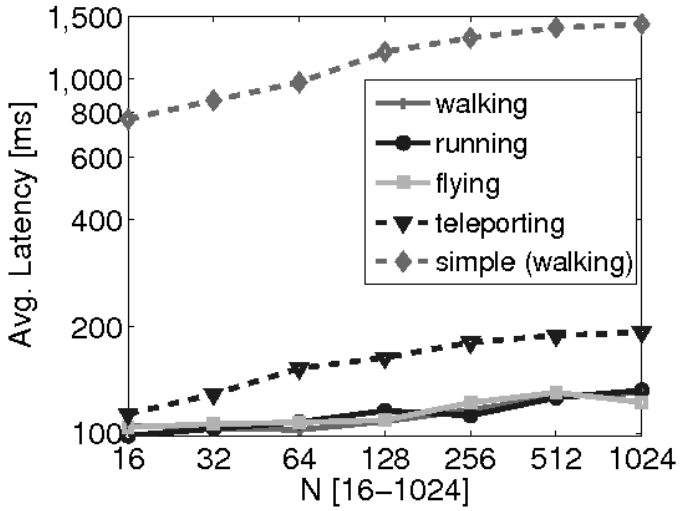


Figure 4.15: $D_{max} = 10$; $N = [16 - 1024]$; Avatar=$[Walk; Run; Fly; Teleport]$.

the network is very small, peers are coordinators of multiple cells, thus reducing the number of routing hops and latency as well.

We now evaluate the impact of *load*, i.e., concurrent number of range queries, on the latency. We set $D_{max} = 10$, $N = 1024$, and we vary N_a, i.e., the fraction of peers whose associated avatars walk, i.e., generate local queries, in the virtual world. Figure 4.16 shows different percentiles of the distribution of latency values as a function of load. We observe that Walkad is very robust to load, as the overall latency is not impacted by a large value of N_a. The fluctuations we observe for each curve depend on the different overlay organizations in the experiments. Figure 4.16 shows another interesting result: 75% of the latency values are smaller than $200\ ms$. This confirms that Walkad solves local queries fast as already observed in Figure 4.14 and 4.15. Only 1% of the latency values are significantly higher, and reach a maximum of $1000\ ms$. These values occur when avatar move across cells located at different levels in the trie (e.g., Figure 4.13). However, even the largest latency value we observe in Walkad is significantly smaller than the average latency value we observe with the simple approach (Figure 4.15).

Load Balancing

We now analyze the load balancing properties of Walkad , i.e., how the responsibility of the virtual world is distributed among peers. We choose $D_{max} = 10$. Figure 4.17 shows several percentiles of the distribution of the fraction of cell-IDs per peer as a function of N. Load balancing is achieved when each peer manages a comparable

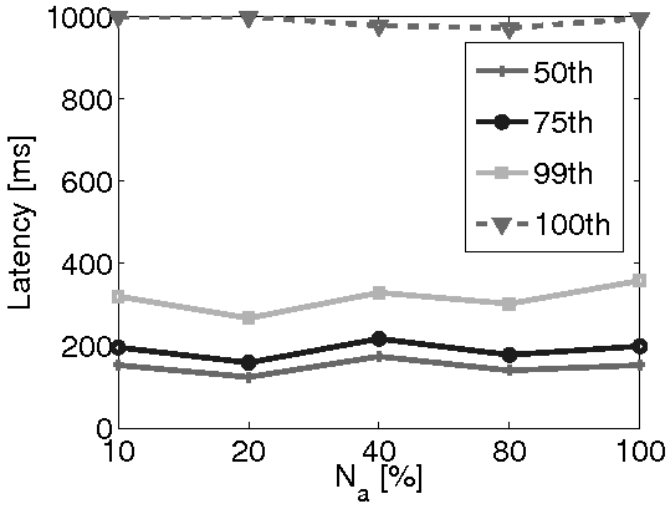


Figure 4.16: $D_{max} = 10$; $N = 1024$; $N_a = [10; 20; 40; 80; 100]\%$; Avatar=$[Walk]$.

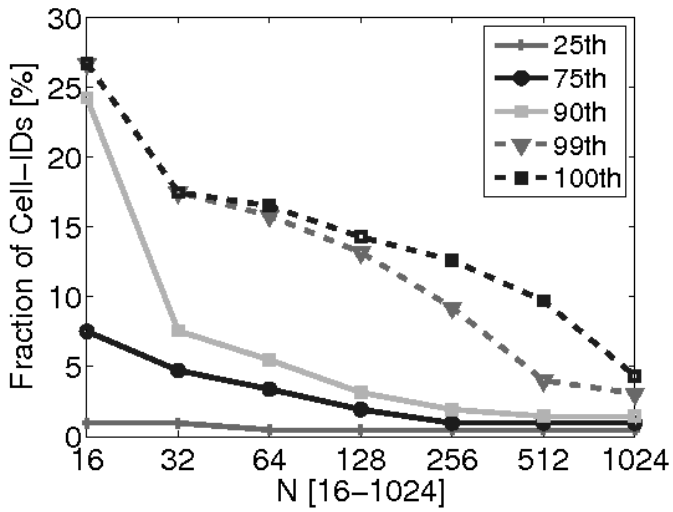


Figure 4.17: Some percentiles of the distribution of the fraction of cell-IDs per peer ; five Second Life regions ; $D_{max} = 10$; $N = [16 - 1024]$.

number of cell-IDs, i.e., when all percentiles of the distribution for a given N assume the same value. Note that this is always the case for the simple approach.

Figure 4.17 shows that as the size of the Walkad network increases the load distribution becomes more uniform among the active peers. For example, when N is larger than 256, for about 90% of the peers the difference in the fraction of cell-IDs they manage is smaller than 1%. The remaining 10% of the peers are responsible of a larger fraction of cell-IDs. This is due to the fact that we are considering a small virtual world composed by only five Second Life regions. Therefore, the global cell-ID organization is still impacted by the specific cell-ID organization within each region.

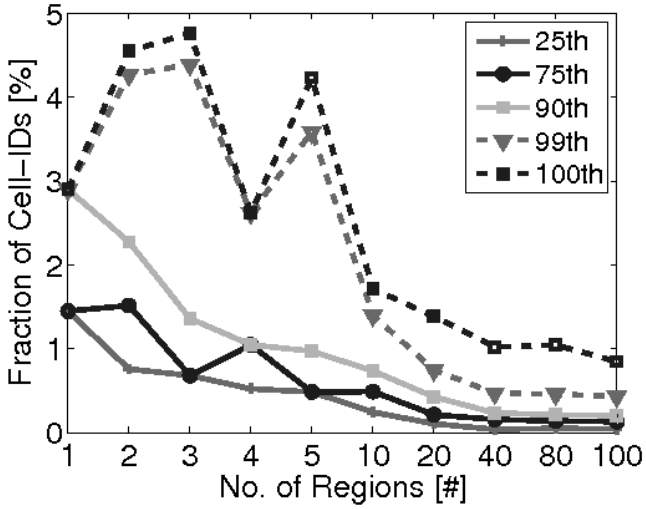


Figure 4.18: Some percentiles of the distribution of the fraction of cell-IDs per peer ; $D_{max} = 10$; $N = 1024$; No. of regions=$[1 - 100]$.

We also evaluate the distribution of the virtual world responsibilities as a function of the virtual world size, i.e., number of Second Life regions indexed with Walkad (Figure 4.18). To do so, we set $D_{max} = 10$, $N = 1024$ and we vary the number of Second Life regions between 1 and 100 using the dataset described in Chapter 3. Figure 4.18 shows that when the virtual world is small, i.e., composed by less than five regions, we cannot identify a general trend for the curves. In fact, the different object compositions strongly impact the general distribution of the cell-IDs. By focusing on a number of regions ≥ 10, we observe that the division of the virtual world responsibilities becomes more and more uniform as the size of the virtual world increases. For example, in a virtual world composed by 100 Second Life regions only 1% of the peers store a larger portion of the virtual world, which consists in worst case of only 1% of the entire virtual world.

4.5 Conclusions

This Chapter presented: (1) an experimental evaluation of a distributed object management for Networked Virtual Environments (NVEs), and (2) *Walkad*, a Kademlia-based Peer-to-Peer (P2P) network designed to manage objects in NVEs.

The distributed object management consists of dynamically partitioning the virtual world into cells, and assigning responsibility for those cells to peers named *coordinators*. This simple distributed object management can be implemented over any structured P2P network. In order to perform realistic experiments, we integrate the

object management over KAD, a widely deployed structured P2P network based on the Kademlia routing protocol. Then, we use traces of user activity (i.e., movement, churn and object creation) in Second Life (SL) to perform a realistic emulation of a P2P NVE on the Internet.

Our evaluation shows that the architecture we have designed achieves acceptable levels of *consistency*, *persistency* and *scalability*. Inconsistency is temporary and limited to avatars that enter new cells or join the P2P NVE. Persistency is excellent for the whole duration of the emulation. Nevertheless, the architecture can scale up to the number of objects contained in a typical SL region. However, we also show that avatars experience long latency to recover from an inconsistent view of the virtual world. The causes of these inconsistencies are mainly: (1) avatars joining the P2P architecture, (2) avatar movements across cells.

The recovery latency due to newcomers joining our P2P NVE can be easily reduced. The tree based organization of the cells could be cached at the clients and re-used across different sessions. During the initial connection to the NVE or in case the cell organization has changed between two sessions, the effective join of an avatar could be easily delayed. Reducing the recovery latency due to avatar movements is a more complex problem to solve. Pre-locating the coordinators for all adjacent cells can help reduce boundary-crossing latencies. However, this solution would come at the expense of increased traffic and load on the coordinators.

In order to reduce the recovery latency in presence of avatar movements, we design *Walkad*. Walkad maps cells that are close in a region of the NVE to peers that are overlay-close in Walkad. This allows a faster navigation in the virtual world with no additional cost on the coordinators. Walkad leverages on the Kademlia routing protocol and on an indexing algorithm based on a reverse binary trie.

We evaluate a prototype of Walkad via network emulation with up to 1024 peers. We simulate a realistic virtual world using object traces from 5 and up to 100 Second Life regions. Our results show that Walkad ensures to NVE users a fast retrieval of the objects located in their avatar surroundings. Moreover, the management load of the virtual world is fairly distributed among peers as both the network and virtual world grow.

The distributed object management described could be generalized to the management of avatars. The state of an avatar could be represented as a "dynamic object" and indexed over Walkad. In the thesis, we do not investigate this approach as it is very inefficient for the following reasons. First, the dynamism of avatars (movements in the virtual world and churn) would cause the world to be continuously split

and merged, i.e., reducing object consistency. Second, avatars are non-persistent entities. That said, the mechanisms described to ensure object persistency are useless and inefficient when handling avatar states. Third, the state of an avatar changes continuously causing high rate of publish and search operations. These issues motivate Chapter 5 where we investigate the design and deployment of a *distributed avatar management*.

CHAPTER 5

Distributed Avatar Management

5.1 Introduction

The *avatar management* in Networked Virtual Environments (NVEs) consists of informing each avatar about the status of its neighbor avatars in real time. In a Client/Server (C/S) NVE this operation is straightforward. The server maintains a copy of all avatar states given the information received by the clients, and simply computes for each avatar the set of avatars that intersect the avatar Area of Interest (AoI). However, even this simple operation can become very difficult in presence of un-predictable number of concurrent users and avatar behavior. For example, Second Life (SL) servers get easily over-loaded with as little as 40 concurrent avatars (cf. Chapter 3).

A scalable approach to avatar management consists in delegating to each user the management of its own avatar state, i.e., adopting a *distributed avatar management*. A Peer-to-Peer (P2P) network is formed among NVE users such that each NVE user is connected to the set of peers whose avatars are located in its avatar AoI. Then, peers continuously exchange information about their avatar states. Previous work shows (by simulation) that the *Delaunay Network* [69][9] allows to efficiently manage avatars in a distributed way. The Delaunay Network is a structured P2P network where peers connectivity is driven by their avatar coordinates in the NVE.

In this Chapter, we first study experimentally the Quality of Experience (QoE) that a distributed avatar management provides to NVE users. To do so, we deploy a **P2P-SL client** that leverages the Delaunay Network in order to manage the dissemination of avatar state updates in SL. Then, we design and evaluate several optimizations to Delaunay-based NVEs: (1) a clustering algorithm to efficiently manage large avatar

groups, and (2) the *Social Delaunay Network*, a secured Delaunay Network obtained by taking into account the friendship relationships that exist among avatars in a NVE. Part of this work has been published in [101][102][106].

We compare P2P versus C/S Second Life by focusing on the QoE perceived by multiple SL users. We execute several instances of the P2P-SL client on Planetlab machines [74] and we populate a SL region with our controlled avatars. SL avatar mobility traces (cf. Chapter 3) are used to reproduce real avatar behaviors. We show that a distributed avatar management for SL always outperforms the current C/S design. In a P2P Second Life, 20% of the times users perceive a more correct view of their neighbor avatars compared to C/S. About 90% of the times inconsistency in P2P is solved in less than one second, i.e., five times faster than in C/S. However, the experimental evaluation of the Delaunay Network shows also that user QoE is reduced in presence of avatar groups, fast movements, and churn.

We evaluate the clustering algorithm for Delaunay-based NVEs with Matlab simulation and using a 30 minute avatar trace. Our results show that the clustering is effective in reducing the volume of maintenance traffic of a Delaunay Network. Most importantly, the clustering achieves an efficient utilization of peer resources that allows to improve the responsiveness of avatar interactions in the NVE.

Finally, we evaluate the Social Delaunay Network using Matlab simulations and the 10 day traces of avatar mobility and social behavior collected in the Japan Resort region (cf. Chapter 3). We show that enforcing security in the Delaunay Network comes at the cost of a server effort to manage the interactions among avatars not socially connected. However, for a reasonable value of acquaintance required among avatars to be directly connected, still 80% of the avatars manage the propagation of about 75% of their avatar state updates without any server help. Interestingly, our results indicate also that the Social Delaunay Network slightly improves the responsiveness of avatar interactions in the NVE compared to the classic Delaunay Network.

5.2 The Delaunay Network

The *Delaunay Network* [69] is an overlay network whose topology is defined by a Delaunay triangulation. In the following, we give a formal definition of the Delaunay triangulation:

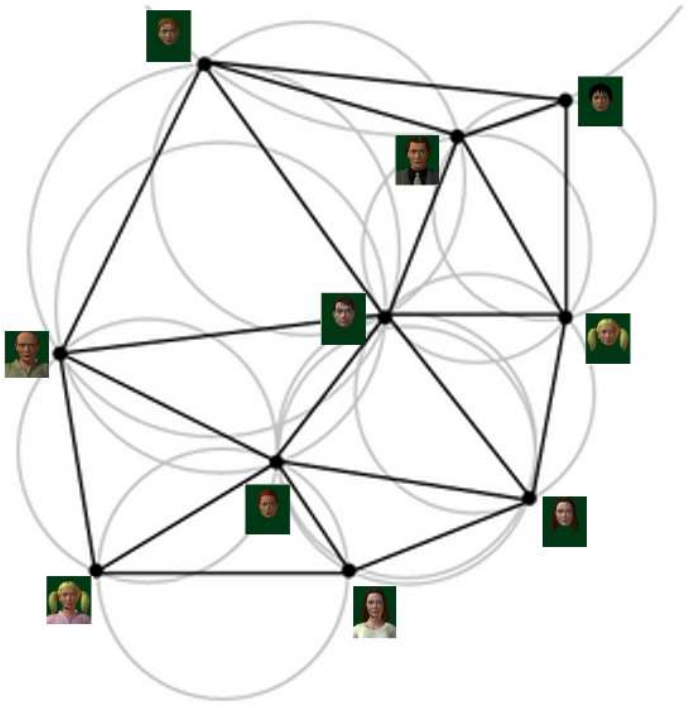


Figure 5.1: Distributed avatar management leveraging the Delaunay Network.

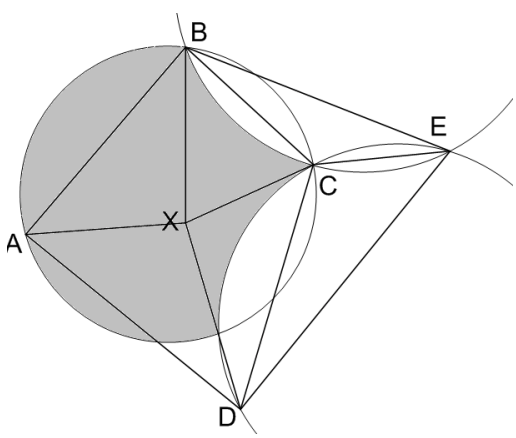
Definition 1 *The Delaunay triangulation of a set of N points in $\Re^2$ is a triangulation of points $DT(N)$ such that no point p lies inside the circumcircle of any triangle in $DT(N)$.*

The coordinates of avatars in the virtual world are used to generate the Delaunay triangulation, and consequently build a Delaunay Network among the respective end-users. In this way, the interactions among avatars on a proximity metric basis are reflected into Internet connections among their end-users. An inner node in a Delaunay triangulation, i.e., a node not on the convex hull[1] of the triangulation, with a sufficiently large population has an average number of neighbors limited to six [16]. This means that each peer in the corresponding Delaunay Network is connected to a finite number of peers independently from the NVE population. Figure 5.1 shows an example of a Delaunay triangulation constructed among avatars in a virtual world.

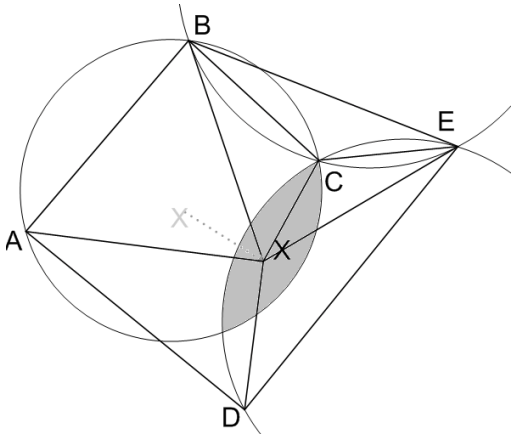
An avatar that participates to a Delaunay Network continually monitors the position of its neighbor avatars in order to maintain a valid triangulation over time. As avatars move or enter/leave the NVE, Delaunay links are added and removed through *flip* operations in order to maintain a valid and consistent triangulation. We explain how a flip operation works with an example (Figure 5.2). We start with a Delaunay triangulation that involves nodes A,B,C,D,E and X (Figure 5.2(a)). The gray

[1]The convex hull for a set of points X in a vector space $V \in \Re^k$ is the minimal convex set containing X.

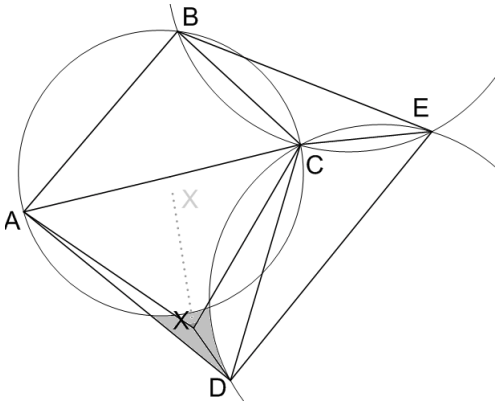
area is the portion of the Delaunay triangulation where node X can move without violating Definition 1. In Figure 5.2(b), node X moves and enters circumcircle C_{cde}. Consequently, triangle T_{cde} is disrupted and triangles T_{xce} and T_{xde} are created. To do so, link CD flips to XE. Figure 5.2(b) shows a more complex scenario. Node X moves out from circumcircle C_{abc} and so triangle T_{abc} can be formed, i.e., link XB flips to AC. This flipping technique is possible because the Delaunay triangulation maximizes the minimum angle of its triangles, i.e., it tends to avoid the creation of skinny triangles [16].



(a) Delaunay triangulation of nodes A,B,X,D,C and E



(b) Node X enters circumcircle C_{cde}, link CD is replaced by link XE



(c) Node X leaves circumcircle C_{abc}, link XB is replaced by link AC

Figure 5.2: Evolution of a Delaunay triangulation via flip operations.

5.3 A Distributed Avatar Management for Second Life

In this Section, we design a P2P-SL client that performs a distributed avatar management. Then, we evaluate the effectiveness of this distributed avatar management through realistic experiments conducted in Second Life.

5.3.1 P2P-SL Client

We use the libsecondlife [61] libraries to deploy a P2P-SL client. The P2P-SL client includes the fundamental features of the official SL client, e.g., login/logout operations and avatar movements, while removing the CPU intensive operations, e.g., the three dimensional rendering of the virtual world.

The P2P-SL client does not require to be human-controlled. Avatar traces, e.g., movement and churn, can be used to automate client operations. The innovative feature of the P2P-SL client is the possibility to directly communicate with other P2P-SL clients without the need of a server.

In order to permit direct communications among SL users, the P2P-SL client implements the Delaunay Network protocol using HyperCast [45]. HyperCast is a set of Java libraries that allows to build several overlays such as the Delaunay Network [16] and Pastry [81]. HyperCast provides to the P2P-SL client a **Neighborhood table** that contains the routing information towards the one-hop Delaunay neighbors of a peer. A complete description of HyperCast can be found in [62].

We build a distributed avatar management for SL on top of the Delaunay Network constructed among P2P-SL clients. To do so, we intercept the avatar state updates generated by the client and transmitted to the SL server and we duplicate them into the Delaunay Network. Note that the avatar state updates sent to the server are now redundant as each avatar already manages its state updates via the Delaunay Network. However, we experienced that suppressing or reducing this traffic causes two main problems: (i) the server continually queries the P2P-SL client about its avatar state, (ii) the server can label our avatars as "misbehaving" and temporarily exclude them from its region. Nevertheless, this strategy is extremely useful to perform a fair comparison between P2P and C/S Second Life (Section 5.3.2).

We use UDP as transport layer protocol for the dissemination of the avatar state updates over the Delaunay Network. The official SL client also uses UDP communication to transport the avatar traffic. Similarly to the SL design choice, we opted

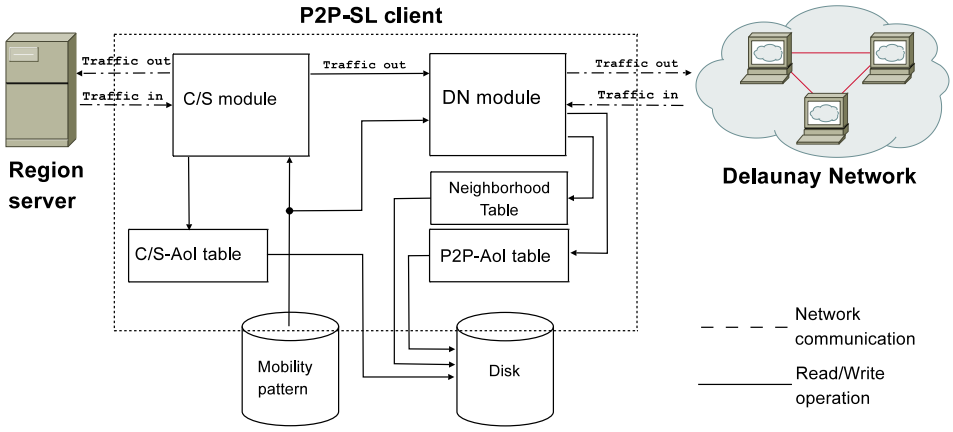


Figure 5.3: The P2P-SL client design.

for UDP since the avatar state updates are transmitted at a constant rate, and do not need delivery guarantees.

We now propose a detailed description of the P2P-SL client (Figure 5.3). In the following, we call **node** the representation of an avatar in the Delaunay triangulation.

- The **C/S module** is the core of the P2P-SL client. It manages the communication between the P2P-SL client and a SL server, i.e., avatar, object and landscape discovery. It receives as input the avatar mobility pattern that it uses to emulate a realistic avatar behavior on a SL region. Most importantly, it duplicates the traffic dedicated to the avatar state management and forwards it to the Delaunay Network module.

- The **C/S-AoI table** is the data structure that contains up-to-date avatar state information for the avatars located within an avatar AoI. This data structure is constantly updated using the avatar traffic received from the SL server. A snapshot of the C/S-AoI table is copied to disk every 200 ms or when a modification of its content occurs.

- The **Neighborhood table** is the data structure maintained by the HyperCast libraries. It contains routing information towards the Delaunay one-hop neighbors of a peer. A snapshot of the Neighborhood table is copied to disk every 200 ms or when a modification of its content occurs.

- The **Delaunay Network module** manages the Delaunay Network, and the transmission/reception of avatar state updates. It receives as input the avatar mobility pattern that it uses to update the coordinates of the corresponding node in the Delaunay triangulation. This information is propagated to the nodes contained in the node's Neighborhood table via heartbeat messages at a

fast or *slow* rate. The fast rate (1 message every 200 ms) is used during the join of a new node and in case of unstable neighborhood, e.g., when a node changes position or a newcomer joins the network. The slow rate (1 message every sec) is used when the neighborhood is stable. Rate values are chosen according to [62]. The Delaunay Network module receives as input the traffic generated by the C/S module for the avatar state updates. This traffic is flooded into the Delaunay Network with *AoI filtering*, i.e., packets are not forwarded farther than an avatar AoI (35 meters as default in SL). Local forwarding decisions at nodes are made using *compass routing*, e.g., among three nodes A, B and C that are all one-hop Delaunay neighbors of a node D, the node that forwards the avatar state update received by a node R is the node that minimizes the angle it forms with R and D [57].

- The **P2P-AoI table** is the data structure that contains up-to-date avatar states information for the avatars located within an avatar AoI. This data structure is constantly updated using the avatar traffic received from the Delaunay Network module. A snapshot of the P2P-AoI table is copied to disk every 200 ms or when a modification of its content occurs.

5.3.2 Evaluation

We compare a P2P versus a C/S architecture for Second Life by focusing on the user Quality of Experience (QoE) perceived by automated SL users. We adopt a methodology similarly to the one used in Section 3.7.1. We launch the P2P-SL player over multiple Planetlab [74] machines and we populate a SL region empty of any objects with our controlled avatars. In this way, objects do not to interfere with avatar mobility patterns. Our automated avatars reproduce real avatar behaviors using a mobility trace collected in the SL Japan Resort region (cf. Chapter 3). We reproduce the behavior of 207 different avatars during one hour; Figure 5.4 shows the evolution over time of the avatar population we reproduce in SL.

P2P vs C/S

As defined in Section 3.7.1, we compute **inconsistency** and the **inconsistency duration** in order to compare the QoE perceived by SL users respectively in a C/S and P2P Second Life.

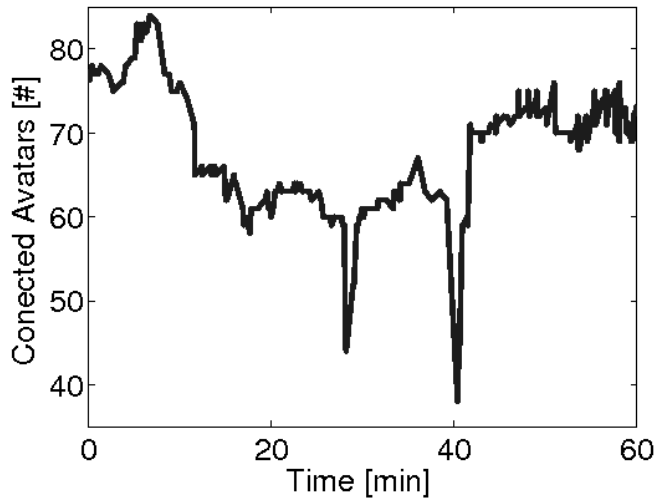


Figure 5.4: Evolution over time of the no. of connected avatars during our experiment.

Inconsistency We start by looking at the probability to have inconsistencies in the avatar AoIs. We evaluate the inconsistency for each avatar AoI every $200\ ms$ and anytime a modification of the AoI occurs. Figure 5.5 shows the Cumulative Distribution Function (CDF) of the inconsistency values computed on both C/S and P2P Second Life. Since we plot the inconsistency values in logarithmic scale (x-axis in Figure 5.5), the two curves are truncated respectively for inconsistency values equal to 0.02 for P2P and 0.05 for C/S, i.e., the smallest non-zero values measured during our experiments.

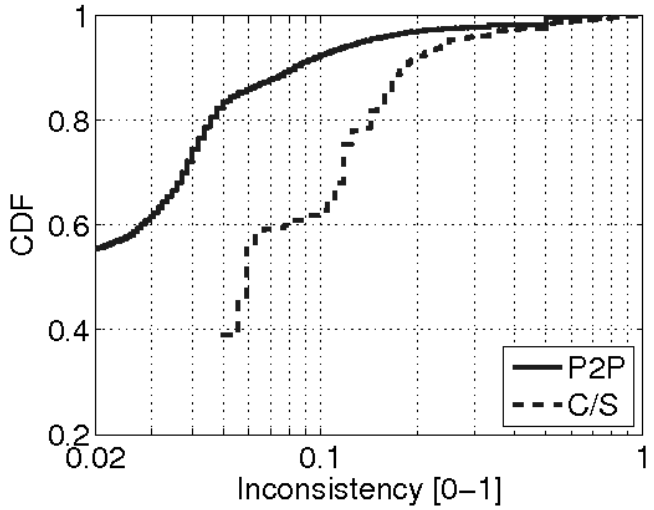


Figure 5.5: CDF of the inconsistency.

Figure 5.5 shows that P2P achieves higher consistency than C/S. Avatars have a perfect view of their AoIs, i.e., inconsistency equals 0, in about 55% of the cases

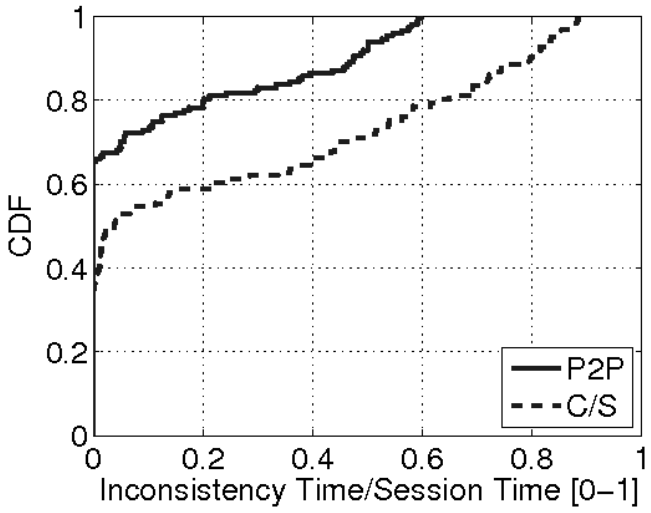


Figure 5.6: CDF of the fraction of time an avatar AoI is inconsistent.

compared to 40% of the cases in C/S. The distance between the two curves is roughly constant for inconsistency values smaller than 0.2, indicating that P2P produces a gain of correctness in the user experience of about 20%. For inconsistency values larger than 0.3-0.4 the two curves nearly overlap. These high inconsistency values happen in presence of churn (i.e., login/logout operations) and avatar groups. While the SL server suffers these events due to an increase on its load (cf. Chapter 3), the P2P overlay suffers due to the difficulty in maintaining a consistent Delaunay triangulation [9].

We now want to understand how frequently inconsistency events affect an avatar during its SL journey. Figure 5.6 plots the CDF of the ratio between the sum of the durations of an avatar inconsistency periods and the total time the avatar stays in a region. We observe again that C/S suffers more from avatar inconsistency than P2P. In C/S, about 35% of the avatars do not see any inconsistency event, whereas this number nearly doubles in P2P. Interestingly, inconsistency in P2P never lasts more than 60% of the time an avatar spends in a region, whereas in C/S, 10% of the avatars have an inconsistent view of their neighbor avatars during about 80%-90% of their SL journey. The reason behind this phenomenon is that the SL server spends a lot of time to correctly accomplish avatar login/logout as we will investigate in Section 5.3.2. Subsequently, avatars with very short session times have an inconsistent AoI most of the time.

Inconsistency Duration We now analyze the inconsistency duration in P2P and C/S in order to understand which architecture solves avatar inconsistencies faster (Figure 5.7). As for the inconsistency results, P2P clearly outperforms the current

C/S design. About 90% of the time, inconsistency in P2P lasts less than 1 second, i.e., P2P is about 5 times faster than C/S. This result is very promising if we consider that acceptable values of interactivity in on-line games vary between 300 ms and 1 sec [22]. Conversely, Figure 5.7 unveils unacceptable inconsistency duration values under the current C/S architecture, e.g., 40% of the inconsistencies last for more than 2 seconds.

Figure 5.7 shows another interesting result. SL avatars can experience a very long inconsistency duration both under a P2P and a C/S architecture, e.g., about 10 seconds in P2P and 20 seconds in C/S. Similarly to what we observed in Figure 5.6, churn (i.e., login/logout operations) is the main cause of these high values of inconsistency duration.

Finally, despite the fact that the Delaunay Network allows a direct communication among SL avatars Figure 5.7 shows that only 20% of the inconsistencies in P2P last less than 150 ms, i.e., a value comparable to common network latencies over the Internet [113]. Since avatars tend to form groups in SL (cf. Chapter 3) the dissemination of avatar state updates require multiple hops in the Delaunay Network to reach all the interested avatars. This operation generates additional latencies that increase the inconsistency duration.

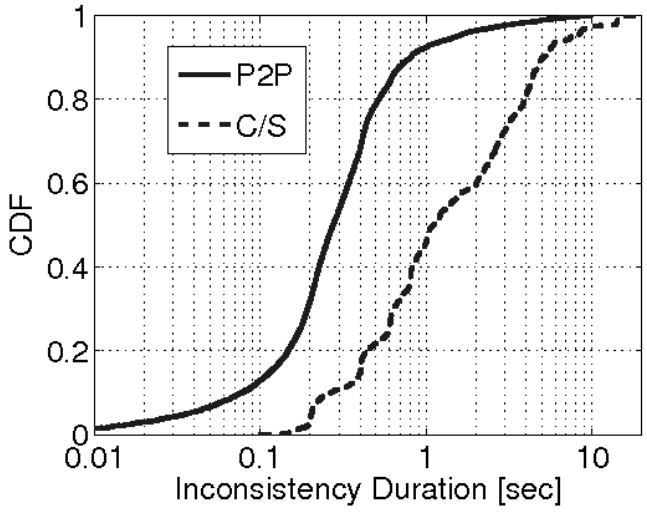


Figure 5.7: CDF of inconsistency duration.

5.4 Dynamic Clustering

The experimental evaluation of the Delaunay Network unveils that the QoE perceived by NVE users is reduced in presence of large avatar groups. In fact, the com-

bination of high avatar density with avatar dynamism generate continuous network rewirings that make the Delaunay triangulation mostly incorrect.

In this Section, we design and evaluate a *clustering* algorithm for the Delaunay Network that addresses the problem of avatar groups in NVEs. The clustering algorithm we design has the following properties: (1) eliminates un-necessary network maintenance operations, (2) increases the responsiveness of avatar interactions in the NVE, and (3) efficiently exploits end-user resources. In the reminder of this Section, we first formalize the problem we want to solve, then we design and evaluate the clustering.

5.4.1 Problem Formalization

We assume a finite population of N peers that never leave the network, i.e., there is no churn. Table 5.1 summarizes the parameters we use in the analysis along with a brief explanation.

Parameter	Definition
T_{ijk}	triangle defined by nodes i, j, and k
C_{ijk}	circumcircle of triangle T_{ijk}
r_{ijk}	radius of circle C_{ijk}
C_m	maintenance cost
$d1$	average distance between a node and its 1-hop neighbors
$d2$	average distance between a node 1-hop neighbors and its 2-hop neighbors
N	number of users
N_1	number of 1-hop neighbors
R_b	rate of keep-alive messages
R_f	rate of flip operations
v	speed of node X
L_{min}	minimum allowed path before flip
L_{max}	maximum allowed path before flip
k	number of packets per flip operation

Table 5.1: Table of parameters.

The maintenance of the Delaunay Network is a task distributed among its peers. The maintenance cost consists of two different components. First, each peer monitors its one-hop neighbors via keep-alive messages at a fixed rate R_b in order to obtain their positions. Second, when the Delaunay Network needs to be rewired peers exchange

control messages in order to set links active or inactive, i.e., perform a flip operation. Accordingly, Equation 5.1 gives a simple expression for the maintenance cost.

$$C_m = N_1 \cdot R_b + k \cdot R_f \tag{5.1}$$

The first term in 5.1 ($N_1 \cdot R_b$) is the rate of keep-alive messages a peer exchanges with its one-hop neighbors. At steady state, it has been shown that $N_1 = 6$ for an inner node of a two dimensional Delaunay triangulation [16], i.e., a node not located on the convex hull of the triangulation. The second term in 5.1 ($k \cdot R_f$) is the rate at which messages are exchanged during the reconstruction of the triangulation. The parameter k indicates the number of messages required per flip operation and depends on the way the distributed computation of the Delaunay triangulation is performed. R_f is the rate at which flip operations are performed, and depends on avatar density and velocity.

We call L_{min} and L_{max} respectively the minimum and maximum path on which a node of the Delaunay Network can move without triggering any flip operations (e.g., the gray area in Figure 5.2). In order to derive a closed expression for L_{min} and L_{max} and consequently R_f, we need to make some assumptions on the geometric shape of the Delaunay triangulation. For this purpose, we introduce parameters $d1$ and $d2$:

- $d1$: average distance between a node and its one-hop neighbors.
- $d2$: average distance between one-hop neighbors of a node X and the two-hop neighbors that are vertices of a triangle whose circumcircle intersects the convex hull defined by one-hop neighbors of node X.

Figure 5.8 shows the Delaunay triangulation generated by the introduction of $d1$ and $d2$. The convex hull defined by the 6 neighbors of node X is a regular hexagon with side length equal to $d1$. All triangles constructed with the two-hop neighbors are isosceles triangles; the two equal sides have length equal to $d2$, the remaining side is equal to $d1$. All vertices of the hexagon are points on the same circle with radius equal to the side of the hexagon, i.e., for construction $r_{abc} = d1$. The free area of movement for node X is the gray zone in Figure 5.8. The maximum path on which node X is allowed to move is bounded by the radius of C_{abc}, i.e., $L_{max} = d1$.

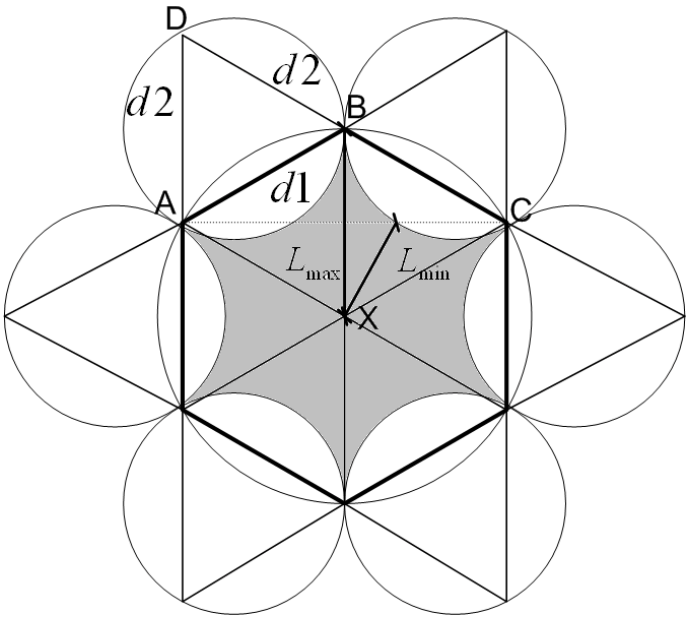


Figure 5.8: Delauany triangulation generated by introducing $d1$ and $d2$.

The simplifications introduced allow us to give a closed formula for $L_{min}(d1, d2)$ (Equation 5.2). To derive it, we consider the distance between node X and the circular segment intercepted by chord BC.

$$L_{min}(d1, d2) = d1 \cdot \left(\frac{\sqrt{3}}{2} - \frac{d1}{2 \cdot (4d2^2 - d1^2)} \right) \tag{5.2}$$

The results above tell us that the minimum path on which a node X can move in a Delaunay triangulation without triggering a flip operation is smaller than $d1$. Each time a flip operation is due, we reconstruct the geometric structure of Figure 5.8 by computing the new values of $d1$ and $d2$. Assuming as hypothesis that $d1$ and $d2$ do not vary too much in the surroundings of node X during a short time T, we can say that $\overline{L_{min}(d1, d2)} \approx L_{min}(d1, d2)$. Then, in order to derive an expression of R_f we compute the time before a flip operation occurs considering node X moving at a constant speed v in the direction of $L_{min}(d1, d2)$. Finally, Equation 5.3 gives a formulation of C_m.

$$C_m = 6 \cdot R_b + k \cdot \frac{v}{L_{min}(d1, d2)} \tag{5.3}$$

The take-home result of this Section is that the maintenance cost as well as the flip rate in a Delaunay Network depend on avatar speed and density. This formalizes the problem highlighted by the experimental evaluation of a Delaunay-based Second Life (Section 5.3.2).

5.4.2 Design

In this Section, we first introduce the rationale of our design and then describe the details of our clustering algorithm for Delaunay-based NVEs. Finally, we perform trace-driven simulations to evaluate our design.

Design Rationale

The main goal of our clustering algorithm for Delaunay-based NVEs is to improve the *responsiveness* of avatar interactions in presence of large avatar groups. Our design is based on the following observation. When a set of avatars get together to form a group, the dissemination of avatar state updates based on the AoI filtering is not efficient. In fact, an avatar that takes part of a group generally needs to distribute its state updates to all avatars within the group, thus several hops over the Delaunay Network are required. This has two main consequences: (1) the maintenance of the Delaunay Network may be not required within a group of avatars , and (2) additional resources may be needed within a cluster to achieve a faster dissemination of avatar state updates, i.e., improve NVE responsiveness.

Our solution is based on a clustering algorithm that identifies avatar groups without requiring global knowledge. Each peer simply monitors its own maintenance cost in order to derive information about avatar groups. Once a group of avatars is identified, we "relax" the Delaunay Network in order to reduce the maintenance cost and improve the dissemination strategy of avatar state updates. Interestingly, identifying avatar groups implicitly highlights where additional network resources may be needed.

Clustering

We consider the P2P-NVE at steady state with N peers organized in a Delaunay Network. A distributed computation of the Delaunay triangulation is used [69]. We

```
1 Compute_C_m();
2 if C_m ≥ B_t then
3     foreach peer i in N_1 do
4         propose_clustering();
5     end
6     select_cluster-head();
7 end
```

Algorithm 2: Cluster initialization

introduce the parameter B_t, the target volume of maintenance traffic per peer within the Delaunay Network.

Each peer monitors its maintenance cost C_m. If peer A notices $C_m \geq B_t$, this suggests that the local density is high and a group of avatars is possibly being created. Peer A reacts by proposing its neighbors the creation of a *cluster*. The neighbors of peer A check their value of C_m and decide whether to take part in the clustering process or not. In case they decide to take part to the cluster, they spread the clustering request to their neighbors by piggybacking it on heartbeat messages.

The peer that first initiates the cluster creation selects the responsible peer for the cluster (Algorithm 2); we refer to this node as the **cluster-head (CH)**. Peers who agree on the creation of a cluster contact the CH. We do not focus on the CH selection, since it is irrelevant for the behavior of the clustering. Note that the CH could be a control server that manages the NVE (e.g., for billing or login verification) as well as an off-loaded peer.

The CH collects all pending requests for clustering. For each cluster c, it computes the area S_c occupied by the cluster and its center (X_c, Y_c). Then, the CH informs all involved peers of the cluster creation (Algorithm 3).

```
1 collect_clustering_requests();
2 foreach cluster c detected do
3     compute S_c, X_c, Y_c;
4     foreach peer i in N_c do
5         send(S_c, X_c, Y_c);
6     end
7 end
8 inform_extra-cluster_nodes();
9 monitor_the_cluster();
```

Algorithm 3: Cluster definition

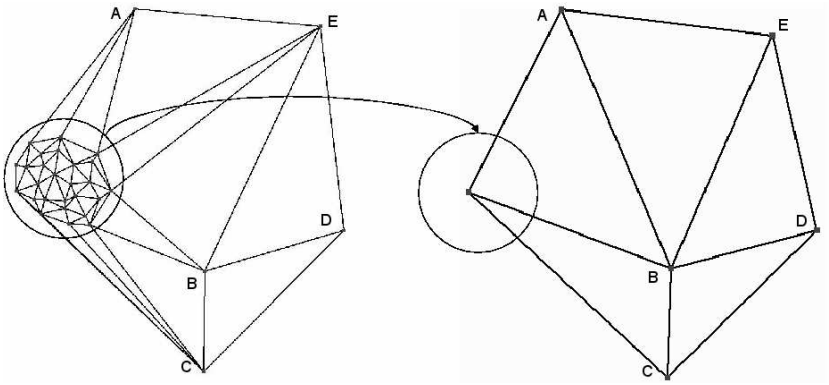


Figure 5.9: Example of clustering in a Delaunay Network.

External nodes refer to the cluster considering it as a **virtual node** at coordinates X_c, Y_c. The CH monitors this virtual node as if it was a normal node in the triangulation, i.e., using heartbeat messages and flip operations (Algorithm 3). In this way the connection between the cluster and the rest of the NVE is maintained. Figure 5.9 shows an example of clustering in a Delaunay Network.

An extra-cluster node joins a cluster c when its avatar AoI intersects the area S_c. In fact, as nodes within a cluster are allowed to move within S_c, there is a non-zero probability that they intercept the AoI of extra-cluster nodes. Similarly, an intra-cluster node leaves c when its AoI does not intersect any more S_c. In this way, we ensure visibility among avatars going in and out of a cluster.

The size of the cluster S_c is computed at the creation of the cluster and is not modified when avatars join and leave. This means that the probability of the cluster to become empty decreases with avatar departures from the cluster. This design choice is driven by the notion we gathered in Second Life that group of avatars tend to meet at some fixed virtual places (cf. Chapter 3). When a cluster is empty, the CH removes it from the Delaunay triangulation. It results as a disconnection event of a node in the Delaunay triangulation.

We call N_c the population of a cluster and $\rho_c = \frac{N_c}{S_c}$ the density within a cluster c. Members of a cluster "expand" their coordinates, i.e., we define $S_c' = E * S_c$ with $E > 1$. The expansion is unique for every cluster member so that the angular relationship among nodes remains the same, i.e., the intra-cluster Delaunay links are not modified. The cluster density becomes $\rho_c' = \frac{\rho_c}{E}$. Cluster creation works in a hierarchical way, i.e., clusters can be created within clusters if the expansion results not to be enough or if N_c grows too much.

The expansion of the avatar coordinates within a cluster increases the distance among nodes. Moreover, the Delaunay triangulation generated does not reflect any-

more the actual positions of avatars in the NVE. Therefore, the dissemination of avatar state updates based on AoI filtering is not more meaningful. However, when the density of avatars is high this dissemination strategy is inefficient. We now discuss how we manage the dissemination of avatar state updates within a cluster.

A simple approach consists of allowing each avatar to broadcast its avatar state updates among the members of a cluster. This approach increases the upload bandwidth consumed by peers within a cluster but implicitly solves the problem of slow NVE responsiveness measured within avatar groups (Section 5.3.2). In fact, long delays are traded for higher upload bandwidth. In order to keep low the upload bandwidth of peers within a cluster, the solution we propose is to take advantage of under-utilized peers in the Delaunay Network.

Specifically, peers within a cluster disseminate their avatar state updates to their Delaunay one-hop neighbors, and to the CH. The CH acts as a relay node and updates all peers whose avatars fall within the AoI of the avatar that originates the update. The additional bandwidth required by a CH for this task is provided by the peers that are responsible for the nodes located on the convex hull of the cluster (e.g., peers A,B and C in Figure 5.9), if they do not already manage a cluster. Since these peers manage avatars that are not part of a group, they may have spare upload resources. This approach is suboptimal since many other under-utilized peers may be available in the Delaunay Network. However, finding these resources require the usage of complex distributed resource selection strategies [39]. Conversely, the solution described is very simple as the CHs already know the peers on the convex hull of the cluster.

5.4.3 Evaluation

We evaluate the Delaunay clustering through a Matlab simulator. The simulator builds and maintains the Delaunay triangulation among avatars and manages the clustering. In the simulator, each flip operation counts as one packet sent to the N_1 neighbors of the involved nodes. The packet size is set to 100 $bytes$. We does not count heartbeat messages, since they represent an additional cost for all peers. We consider one level of clustering only. Avatars move realistically according to avatar traces we collect with our Second Life crawler (cf. Chapter 3) in the Public Help Island region for 30 minutes. The traces contain patterns of movement for 89 unique avatars.

Maintenance Cost

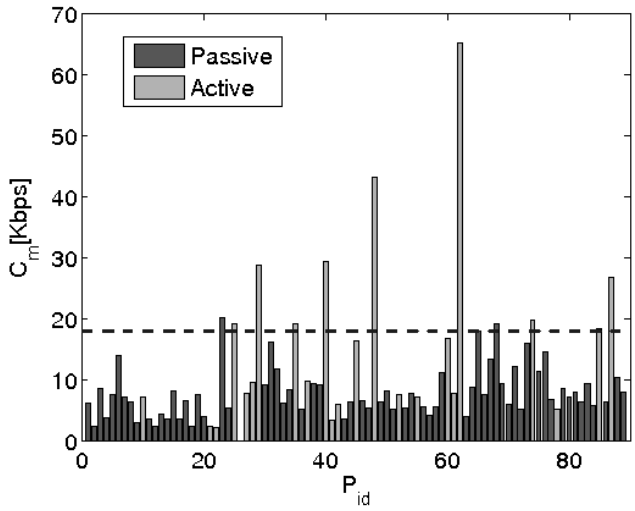
We start by analyzing the effect of the clustering on the maintenance cost C_m of a Delaunay Network. Figure 5.10 plots the median value of C_m computed over 30 minutes for each of the 89 peers in the Delaunay Network respectively before (Figure 5.10(a)) and after clustering (Figure 5.10(b)). Figure 5.10 do not show the overhead introduced by the clustering as in the current implementation a centralized unit is responsible of the clustering management. We distinguish between two categories of peers, *passive* and *active*. Passive/active peers have an average velocity smaller/higher than $1\ unit/sec$. We indicate active peers with a light color and passive peers with a dark one.

Figure 5.10(a) shows that C_m is low for the majority of the peers. These are the passive peers that account for the largest portion of avatars in Second Life (cf. Chapter 3). Conversely, active peers exhibit a very large maintenance cost, up to 6 times larger than the majority of the peers. Avatar dynamism is not the only cause of high C_m: for example peers 23, 64 and 67 experience a considerable value of C_m even though they are passive peers. In this case, churn is the cause of their considerable maintenance cost. A churn event counts as an insertion/deletion of a node in the triangulation, so it affects the maintenance cost C_m of the specific peer.

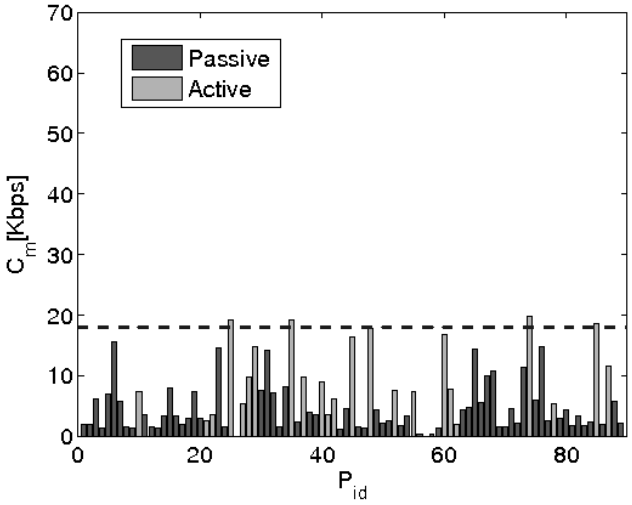
According to what we observe in Figure 5.10(a), we set the clustering threshold B_t at 18 kbps, i.e., we target to cut the very high maintenance cost that affects about 10% of the peers, and we re-run the simulation applying our clustering algorithm (Figure 5.10(b)). As expected, the clustering is effective in keeping the maintenance cost of the Delaunay Network under the desired threshold. Moreover, the maintenance cost of several passive peers is further reduced.

Responsiveness

We now analyze the advantages of the clustering for the propagation of the avatar state updates. We compute the **propagation hops**, i.e., the number of overlay hops required to correctly distribute the avatar state updates. Intuitively, keeping low the number of propagation hops improves the NVE responsiveness. Figure 5.14 plots the CDF of the number of propagation hops computed over the Delaunay Network before and after clustering. We set $AoI_r = 35\ units$ and $B_t = 18\ kbps$. As expected, the clustering does not impact the dissemination of avatar state updates that are accomplished in a single hop, i.e., about 40% of avatar state updates. In fact, the dissemination of avatar state updates to one-hop Delaunay neighbor is the same



(a) Delaunay Network



(b) Clustering in a Delaunay Network

Figure 5.10: Maintenance cost C_m ; Public Help Island (SL) ; $N = 89$; $T = 30\ mins$.

within or outwards a cluster. Conversely, the clustering efficiently reduces about 20% of the propagation hops that are larger than 3 hops in the original Delaunay Network to only 2 propagation hops. Clearly, this is the effect of the CHs acting as relay peers. Finally, the clustering halves the maximum number of propagation hops, i.e., from 6 in the classic Delaunay Network to only 3.

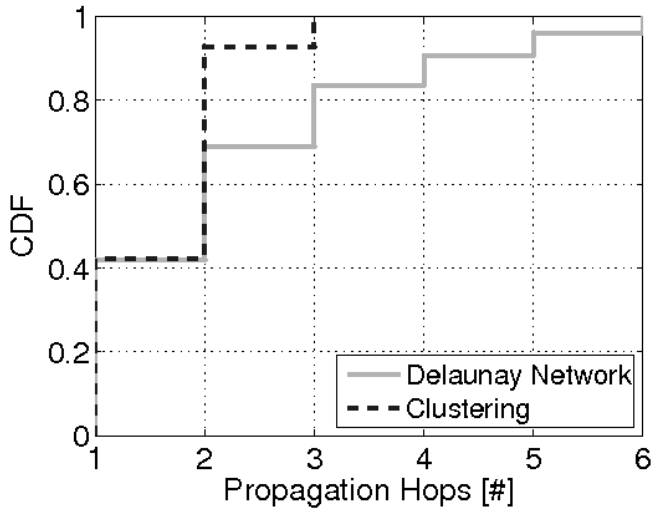


Figure 5.11: CDF of propagation hops ; $B_t = 18\ kbps$; $AoI_r = 35\ units$.

Overhead

The clustering improves NVE responsiveness in presence of avatar groups (Figure 5.11) by exploiting the resources of peers located on the convex hull of the clusters. We now analyze the overhead that this operation introduces. We compute for each peer its **propagation-sets**, i.e., the set of peers a peer sends or forwards the avatar state updates. This is a subset of the **peer-set**, i.e., the set of peers a peer is connected to as indicated by the overlay rules. Note that in case of a two-dimensional Delaunay Network most of the peer-sets are composed by 6 peers [16].

Figure 5.12 plots the evolution over time of several percentiles of the propagation-sets sizes distribution computed every 5 minutes in the Delaunay Network. We start by focusing on the first time slot, i.e., $T = 5\ min$. During this time period, no cluster is created. Figure 5.12 shows that about 50% of the propagation-sets are composed by 4-6 peers. About 25% of the propagation-sets are smaller than 4, indicating some free available resources. Finally, the maximum value of a propagation-set size reaches 10 peers. This happens when the Delaunay Network needs to be rewired in presence of churn and avatar movements. In this scenario, a peer-set can be temporarily expanded making larger propagation-sets possible as well.

We now focus on the second time bean, i.e., $T = 10\ mins$. At this time, two clusters are created. Figure 5.12 shows a neat increase in the peer-set sizes for about 20% of the peers. These are the peers that reside on the convex hull of a cluster, and that provide the additional resources to improve NVE responsiveness within the cluster (Figure 5.14). Figure 5.12 also shows that the highest percentiles of the distribution

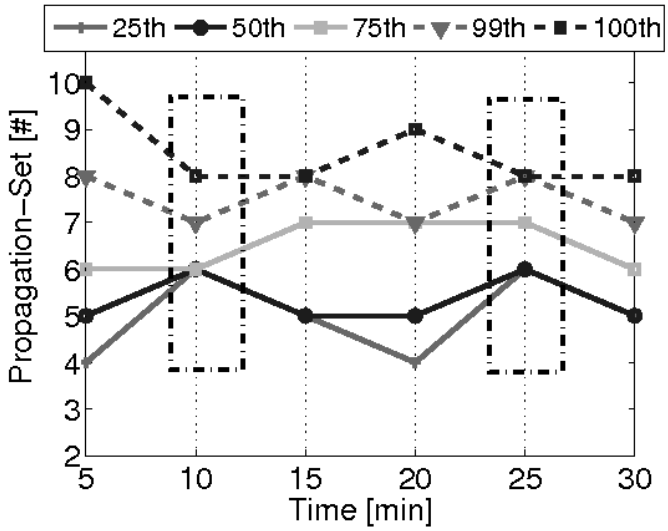


Figure 5.12: Several percentiles of the distribution of the propagation-set sizes over time ; $AoI_r = 35\ units$; $B_t = 18\ kbps$.

are reduced: this happens because the creation of the clusters remove some network rewiring operations (Figure 5.10(b)). These results show the effectiveness of the clustering in obtaining a much uniform usage of user resources in the Delaunay Network. Similar results are observable at $T = 25\ mins$ when another cluster is created.

5.5 Social Delaunay Network

From a technical prospective, a distributed avatar management is an appealing solution for NVEs. However, NVE providers are generally scared by the security threats of a distributed approach. In fact, a malicious user might exploit the direct connections with NVE users to cheat [116], send bogus data or even steal private user information.

In this Section, we design and evaluate a **Social Delaunay Network**. This is an extension to the classic Delaunay Network that enforces security by leveraging the network of confidence avatars implicitly construct by interacting in a virtual world.

5.5.1 Design

Avatars in NVEs construct a network of friendships upon their behaviors, i.e., the time they spend being close. Specifically, we can construct a social graph G where

each edge <i,j> between avatars i and j has two associated weights $w_{i,j}$ and $w_{j,i}$. These weights represent the percentage of avatar i's session times it spends being close to j in the NVE and vice-versa (cf. Chapter 3).

We build the peer-sets of each NVE user upon the friendship network. Specifically, the peer-set of a user i is the set of active users that share an edge with i in G whose both weights $w_{i,j}$ and $w_{j,i}$ are larger than a threshold W, with $0 < W \leq 1$. Therefore, if i is part of j's peer-set, j is part of i's peer-set as well, and so a two-way communication between peer i and j is possible. Requiring both weights to be larger than W implies that both peer i and j need to "agree" in the importance of their social connection to be directly connected.

The Social Delaunay Network is constructed by augmenting and reducing the original Delaunay Network with the information contained in the peer-sets. When two peers A and B enter their respective AoIs, they share a **direct** Delaunay link as in the conventional Delaunay Network if they are contained within their respective peer-sets. In fact, this suggests that A and B are virtual friends and they trust each other. Conversely, when peers A and B are not contained in their respective peer-sets, e.g., the first time they meet, two scenarios are possible. (1) It exists a path that connects nodes A and B in the social graph G entirely composed of edges $< i, j >$ that have both weights $w_{i,j}$ and $w_{j,i}$ greater than W. In this case, peer A and B share an **indirect** Delaunay link composed by several hops in the P2P overlay. The rationale is that reciprocal friends of A and B are used to build a trusted communication link between A and B. Then, if A and B become friends, the indirect link is automatically transformed into a direct link. (2) The path that connects nodes A and B in the social graph G does not exist. In this case, the Delaunay link between A and B is **relayed**. Then, a trusted authority such as a control server is needed to build the communication link between A and B.

5.5.2 Evaluation

We simulate the Social Delaunay Network (SDN) over Matlab. The simulator takes as an input avatar traces and builds the SDN over time.[2] Then, it simulates how the traffic related to the avatar state updates is shared between P2P, i.e., the SDN, and C/S. In the simulations, we vary W, the minimum level of acquaintance required among two peers to be connected. We also variate AoI_r, the radius of the avatar

[2]For comparison, the simulator also builds the classic Delaunay Network.

visibility area. We use the 10 days traces of avatar behaviors collected in the Japan Resort region (cf. Chapter 3).

We start by quantifying the volume of traffic managed by the SDN network, i.e., traffic carried by *direct* and *indirect* Delaunay links, versus the traffic managed by the SL server, i.e., traffic carried by *relayed* Delaunay links. Figure 5.13 plots the Complementary Cumulative Distribution Function (CCDF) of the ratio of P2P traffic and the total traffic as a function of W. Unless otherwise noted, we set $AoI_r = 5\,units$, i.e., the avatar AoI radius equals the avatar interaction range R. Figure 5.13 shows that increasing the level of acquaintance W required by two peers to form a P2P link more server help is required. Intuitively, this happens because the SDN becomes less connected as W increases. However, when $W = 3\%$ (i.e., two avatars are contained in their respective peer-sets if they interact at least 10 minutes each 8 hours) we observe that 80% of the avatars manage about 75% of their avatar traffic. Figure 5.13 shows also that increasing AoI_r from 5 to 35 units (i.e., the default value in SL) the traffic managed by the server increases by about 30%. However, still 50% of the avatars manage about 70% of their avatar traffic via the SDN.

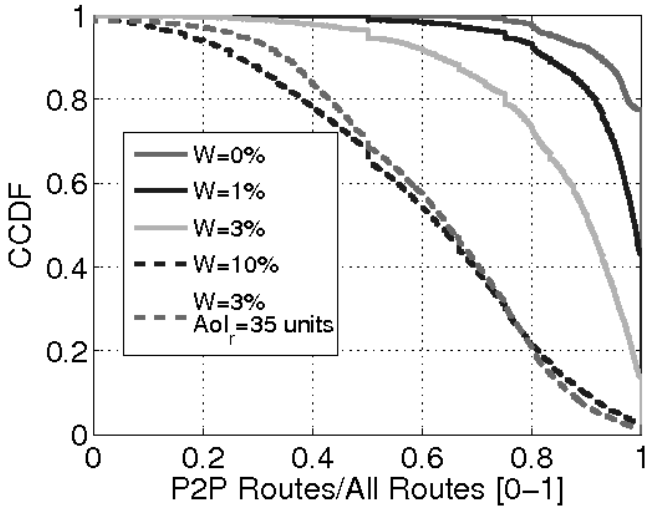


Figure 5.13: Ratio of P2P traffic and total traffic ; $W = [0, 1, 3, 10]\%$; $AoI_r = [5; 35]\ units$.

We now analyze the performance of the SDN in terms of *propagation hops* and *peer-set* sizes. Figure 5.14 plots the CDF of the number of propagation hops computed over the SDN and the Delaunay Network. We set $W = 3\%$ and $AoI_r = 35\ units$ in order to evaluate a realistic scenario. Surprisingly, Figure 5.14 shows that the number of propagation hops is smaller in the SDN than in the Delaunay Network, e.g., 65% of the avatar state updates are delivered in a single hop, while this number is only 40% in the Delaunay Network. Similarly, while 10% of the avatar state updates require more than 5 hops to be propagated, the maximum number of propagation hops is

5 in the SDN. The explanation to this phenomenon is twofold. First, the diameter of the SL social network obtained by filtering all edges with a weight smaller than $W = 3\%$ is small, so large propagation hops are impossible. Second, the additional overlay links introduced in the SDN allow a direct communication among friends as soon as they enter their avatar AoIs independently of their Delaunay distance. These results suggest that the additional overlay links introduced in the SDN are also useful to improve NVE responsiveness.

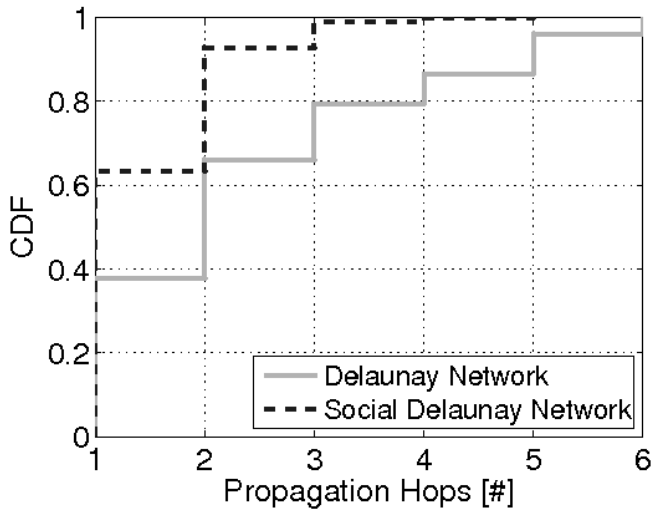


Figure 5.14: CDF of propagation hops ; $W = 3\%$; $AoI_r = 35\ units$.

Figure 5.15 plots the CDF of the peers-set sizes measured in the SDN and in the Delaunay Network. As expected for the Delaunay Network, 80% of the times the peer-set sizes are smaller or equal than 6, a well-known result for peers that do not reside on the convex hull of a two dimensional Delaunay Network [16]. Peer-set sizes larger than 6 in the Delaunay Network are a consequence of network rewirings due to churn and avatar movements. We now focus on the SDN. Figure 5.15 shows that the size of the peer-sets in the SDN is reduced compared to the Delaunay Network, e.g., 90% of the peer-sets have size smaller or equal than 3. This confirms that in the SDN we are removing more overlay links than we are adding. Interestingly, about 50% of the peer-sets size equal zero. These peer-sets are associated to SL users that do not have strong social relationships. As a consequence, their avatar traffic is relayed by the server (Figure 5.13).

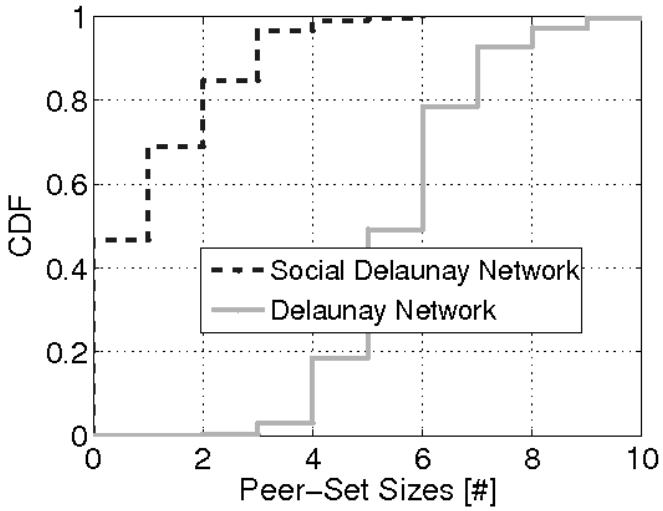


Figure 5.15: CDF of peer-set sizes ;$W = 3\%$; $AoI_r = 35\ units$.

5.6 Conclusions

This Chapter presented: (1) an experimental evaluation of a Delaunay-based avatar management for Networked Virtual Environments (NVEs), (2) a clustering algorithm to handle large avatar groups within a Delaunay-based NVE, and (3) a secured extension to the Delaunay Network that leverages the social component of NVEs.

In order to perform an experimental evaluation of the Delaunay Network, we design a Second Life (SL) client that leverages the Delaunay Network to manage interactions among avatars. We use our client to evaluate P2P versus C/S Second Life. We execute several instances of our client over multiple Planetlab machines and we populate a SL region with our controlled avatars. Avatar mobility traces extracted from SL are used to emulate real avatar behaviors. We show that a distributed avatar management for SL makes the avatar experience more correct and interactive. However, we also unveil that user QoE in a Delaunay-based NVE is reduced in presence of churn, fast avatar movements and avatar groups.

Based on these observation, we design and evaluate a dynamic clustering strategy for the Delaunay Network. The key idea of the clustering is to relax the Delaunay Network in presence of avatar groups, i.e., clusters, in order to improve the dissemination of avatar state updates within a cluster. We show by simulations that the clustering eliminate un-necessary maintenance operations of the Delaunay Network, while improving the responsiveness of avatar interactions in the NVE.

Finally, we design and evaluate the Social Delaunay Network, i.e., a relaxed Delaunay Network obtained by taking into account the information contained in the NVE

social graph. The rationale beyond the Social Delaunay Network is to enforce security in a P2P-based NVE by leveraging the network of confidence users implicitly construct with their virtual behavior. Avatars that do not share any mutual social connection communicate using a control server as a trusted relayed link. Our preliminary results show a clear trade-off between the traffic relayed by the server and the minimum level of acquaintance required to authorize a connection between two peers. Interestingly, we show that the usage of a Social Delaunay Network slightly improves the responsiveness of avatar interactions in the NVE compared to the classic Delaunay Network.

CHAPTER 6

Conclusions

In this Chapter, we first review the work presented in the thesis. Then, we discuss some topics that we could not treat or that could be treated in more detail.

6.1 Summary

A Networked Virtual Environment (NVE) is a synthetic world composed of *objects* where human-controlled *avatars* can interact. NVEs were introduced in the 80s for military simulators. Afterwards, they were successfully applied to on-line games, among which Quake and World of Warcraft have been the most popular. In early 2003, *Second Life* (SL) was launched. SL is a *social virtual world* where avatars can meet, play, trade and even contribute to the development of the NVE. SL rapidly became the most popular NVE, reaching more than 16 million registered users in September 2009. SL differs from previous NVEs in the amount of user-generated content: while in on-line games the virtual world is mostly predefined and static, in SL the virtual world continuously grows and changes through the creation of user-generated objects.

Commercial NVEs leverage a Client/Server architecture where multiple servers maintain the state of the virtual world and distribute it to the users. This architecture is very expensive as large amount of servers need to be deployed, operated and maintained. Moreover, scalability is an issue. These drawbacks motivate alternative designs such as Peer-to-Peer (P2P). This thesis investigates the analysis, design and deployment of a P2P architecture for *user-generated* virtual worlds where avatars can have *fast-paced* interactions. Ideally, a P2P virtual world can scale with the number of its users as each user dedicates some of its resources (storage, CPU, bandwidth)

to the management of the virtual world. Moreover, P2P can dramatically cut server and network cost for the virtual world provider.

Due to the lack of publicly available data about NVEs such as avatar movement patterns or object distribution, we perform an extensive analysis of SL. We deploy a *crawler* and a *player* application and monitor objects, avatars, user Quality of Experience (QoE) and servers performance in the public part of SL over one month. Interestingly, we show that avatars interact similarly to humans in real life, gathering in small groups and visiting the same locations. From a systems perspective, we observe that the SL architecture suffers from a scalability problem and provides poor user QoE.

Based on our observations of SL, we study experimentally how a P2P-based Second Life would perform. We design and build a communication infrastructure that distributes the management of virtual objects among end-users. Then, we integrate this distributed object management on the top of KAD, the P2P network that supports millions of eMule users. We use avatar and object traces extracted from SL in order to perform a realistic emulation of P2P Second Life over the Internet. We show that P2P SL is mostly *consistent*, *persistent* and *scalable*. However, the rendering of the virtual world can be slow in case of a large number of users and objects.

In order to reduce the time avatars spend to find virtual objects located in their surroundings, we introduce *Walkad*, a Distributed Hash Table (DHT) designed to manage user-generated objects in P2P-based NVEs. Walkad leverages the Kademlia DHT and an indexing algorithm based on a *reverse binary trie*. We evaluate Walkad via network emulation, and SL object traces. Our results show that Walkad guarantees a fast discovery of the virtual world, while load balancing the virtual land responsibilities among peers.

Finally, we investigate the feasibility of a distributed avatar management using the *Delaunay triangulation*, a well-known strategy for avatar management in P2P-based NVEs. To start with, we evaluate the performance of the Delaunay triangulation via realistic experiments performed in SL using a modified client we developed. Our results show that P2P greatly outperforms the current C/S design of SL. However, we also show that a Delaunay-based avatar management suffers the presence of avatar groups, which in turn degradates user QoE. In order to address this issue, we design a distributed clustering algorithm that structures the Delaunay triangulation hierarchically when avatar density grows. We show by simulation that the clustering achieves an efficient utilization of peer resources that allows to improve the responsiveness of avatar interactions in the NVE.

Towards the end of the thesis, we discuss to which extent the social component of NVEs can be useful in order to enforce security through trust. The key idea is that, as in real life, friends are willing to store information for us and we generally trust them. As a proof of concept, we design the *Social Delaunay Network*, a P2P network for NVEs based on the Delaunay triangulation and on the information extracted from the *social graph* that avatars construct through their virtual behaviors.

6.2 Outlook

The conclusion of the thesis is that the design of a P2P architecture for NVEs is possible and can guarantee better performance than C/S at a much lower cost. However, there are still some topics that we could not treat or that could be treated in more detail. We discuss some of them in the reminder of this Section.

Enhanced Object Management for Second Life Currently, SL leverages a centralized object management. The objects created on a region are stored on the server responsible of the region. When an avatar enters a region, the associated client downloads from the server the description of the objects located in its avatar surroundings. The client stores recently visited objects in a *cache* in order to make them available for future visits and thus save network traffic [58].

Our analysis of SL unveils that avatars tend to form a social network (Section 3.6). This means that there is a high chance that every time an avatar connects to SL it interacts with its *friends*, i.e., a set of avatars it has repeatedly encountered before. This suggests that the caching system can be enhanced by taking into account avatar social behaviors. The idea is to build a *distributed cache* using the information provided by the social network, i.e., that some avatars meet frequently in the virtual world. Avatars could first attempt to download data from the caches of their trusted friends, and only resort to contacting the server when no friends are available. We expect this approach to greatly reduce the server network traffic, while improving the rendering latency at the clients.

A more challenging enhancement to the SL object management consists of building a completely *distributed object management*. Specifically, we could integrate a **Walkad module** with our P2P-SL client and allow SL users to manage the storage and retrieval of virtual objects over Walkad. The benefits of this approach are multiple. (1) Obtain a proof of concept of the benefits Walkad provides to object management in NVEs. (2) Evaluate Walkad with real experiments and compare its performance with

the centralized approach of SL. (3) Study how P2P and C/S may interact together to form an *hybrid* architecture for NVEs.

Further Usages of Walkad Walkad is an architecture designed to address *range queries* over a P2P-based NVE. However, range queries arise in a number of fields:

- *Location-aware computing* [40] Many applications aim at exploiting the information located in the user surroundings to improve application performance and user experience. This operation is essentially a range query based on geographic coordinates.
- *Databases* [44] Peer-to-peer databases need to support SQL-type relational queries in a distributed fashion. Range queries are a key component of SQL.
- *Distributed computing* [56] Many large-scale computing infrastructures comprise heterogeneous hardware and software resources. This raises the need for scalable resource selection services, i.e., range queries to locate resources within certain ranges in a decentralized manner.

Intuitively, Walkad is an interesting design also for location-aware applications due to the similarities they share with NVEs. Conversely, the benefits of using Walkad in the context of databases are not straightforward. In fact, while predicting the query patterns for NVEs and location-aware applications is quite easy, this is much harder in the context of databases. An interesting avenue of future work consists of building a more general indexing scheme Walkad-like that allows to efficiently handle range queries independently by the application that generates them.

The real need of a Delaunay Network An intriguing research topic is understanding to which extent the maintenance of the Delaunay Network is necessary to handle avatar interactions. The final goal is to design a P2P overlay less structured than the Delaunay Network (i.e., lower maintenance cost) that guarantees an efficient dissemination of avatar state updates. A possible design consists of dividing the peer-set of a NVE user into two logical sets: the *interest set* and the *maintenance set*. The interest set contains the peers responsible of the avatars a user is communicating with. The interest-set may be built using the classic Area-of-Interest filtering or using interest-filtering as discussed in [13]. The maintenance set includes the peers a NVE user needs to communicate with in order to ensure a correct discovery of new neighbor avatars. The construction and maintenance of the maintenance set is a challenging research problem.

6.3 What is Missing?

While we tried to address as many issues as possible related to the design of a P2P NVE, there are still some open issues that we do not solve in this thesis.

Security Security is a fundamental aspect in every system, especially when large scale computer networks are involved. This means that building a secure distributed NVE is a task that requires (probably) another PhD thesis. That said, security lies out of the scope of this work. However, we believe that the social component of NVEs is an interesting feature that can be used in order to enforce security through trust. The key idea is that, as in real life, friends are willing to store information for us and we generally trust them. Chapter 5 partially explores this idea when applied to distributed avatar management. However, this is only a preliminary work and several other aspects may need to be analyzed. As discussed above, a similar approach could also be adopted for the distributed object management.

Integration of the two overlays In the thesis, we reduce the design of a P2P NVE into two distinctive problems: avatar and object management. For each task, we design and evaluate a P2P overlay. While running the two overlays together is a trivial problem, the integration of the two overlays is an interesting research problem that we had not time too look at.

Latency Equalization Responsiveness is a very important requirement for NVEs. In order to achieve responsiveness, we focus on the design of P2P overlays that minimize the overlay-distance among peers. A further optimization consists of taking into account the geographical location of peers in the peer selection strategy. However, in addition to end-to-end latency bounds, NVE applications require that the perceived delay difference among multiple users is minimized. This means that latency should be *equalized* among NVE participants.

Application We evaluate our P2P architecture for NVEs using trace-driven simulations and emulations, and real experiments conducted over the Internet. Moreover, we modify a Second Life client and integrate it with a portion of our P2P architecture. We do not design and deploy any application for NVEs (e.g., an on-line game) and integrate it with our P2P architecture. Indeed, our P2P design can be further improved through a better understanding of the application requirements. Never-

theless, porting an on-line game to work with our P2P architecture is of interest to conduct experiments that involve real-users.

Bibliography

[1] A-Mule. http://www.amule.org/.

[2] K. Aberer, P. Cudre-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a Self-Organizing Structured P2P System. *SIGMOD Rec.*, 32(3):29–33, September 2003.

[3] Active Worlds. http://www.activeworlds.com/.

[4] R. Albert, I. Albert, and G. L. Nakarado. *Phys. Rev.*, (69), 2004.

[5] K. Alsabti, S. Ranka, and V. Singh. An Efficient K-Means Clustering Algorithm. 1997.

[6] R. Antonello, S. Fernandes, J. Moreira, P. Cunha, C. Kamienski, and D. Sadok. Traffic Analysis and Synthetic Models of Second Life. *Multimedia Systems*, 15(1):33–47, February 2009.

[7] J. Aronson. Dead Reckoning: Latency Hiding for Networked Games. Gamasutra, Sept. 19, 1997.

[8] Azureus. http://azureus.sourceforge.net/.

[9] H. Backhaus and S. Krause. Voronoi-Based Adaptive Scalable Transfer Revisited: Gain and Loss of a Voronoi-Based Peer-to-Peer Approach for MMOG. In *NETGAMES'07*, Melbourne, Australia, September 2007.

[10] A. L. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, October 1999.

[11] R. Bartle. Mud, Mud, Glorious Mud. Micro Adventurer, 1(11):22–25, Sept. 1984.

[12] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *SIGCOMM'04*, Portland, OR, USA, August 2004.

[13] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games. In *SIGCOMM'08*, Seattle, WA, USA, August 2008.

[14] A. Bharambe, J. Pang, and S. Seshan. Colyseus: A Distributed Architecture For Online Multiplayer Games. In *NSDI'06*, Berkeley, CA, USA, May 2006.

[15] bittorrent. http://www.bittorrent.com/.

[16] A. Bowyer. Computing Dirichlet Tessellations. *Computer Journal*, 24(2):162–166, 1981.

[17] R. Brunner and E. Biersack. A Performance Evaluation of the Kad Protocol. Technical report, Eurecom, 2006.

[18] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen. The SIMNET Virtual World Architecture. In *Annual International Symposium Virtual Reality IEEE*, pages 450–455, Sept. 1993.

[19] L. Chappell and R. Spicer. Is your Network Doomed? NetWare Connection, Jan./Feb. 1996.

[20] H. Chun, H. Kwak, Y.-H. Eom, Y.-Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: a case study of cyworld. In *IMC'08*, Vouliagmeni, Greece, November 2008.

[21] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law Distributions in Empirical Data, June 2007.

[22] M. Claypool and K. Claypool. Latency and Player Actions in Online Games. *Commun. ACM*, 49(11):40–45, 2006.

[23] Cyworld. `http://www.cyworld.com`.

[24] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. *Towards A Common API For Structured Peer-to-Peer Overlays*, pages 33–44. Number Volume 2735/2003 in Lecture Notes in Computer Science. 2003.

[25] DECsystem-10. `http://en.wikipedia.org/wiki/PDP-10`.

[26] Doom. `http://en.wikipedia.org/wiki/Doom_(video_game)`.

[27] J. Douceur, J. Lorch, F. Uyeda, and R. C. Wood. Enhancing Game-Server AI with Distributed Client Computation. In *NOSSDAV'07*, Urbana-Champaign, IL, USA, June 2007.

[28] H. Edward. *The Hidden Dimension*. Doubleday, Garden City, 1966.

[29] eMule. `http://www.emule.com/`.

[30] Everquest. `http://everquest.station.sony.com/`.

[31] Facebook. `http://www.facebook.com`.

[32] Final Fantasy. `http://www.playonline.com/ff11us/index.shtml`.

[33] Flickr. `http://www.flickr.com`.

[34] L. Gautier and C. Diot. Design and Evaluation of MiMaze, a Multi-Player Game on the Internet. In *IEEE Multimedia Systems Conference*, pages 233–236, 1998.

[35] L. Gautier, C. Diot, and J. Kurose. End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet. In *INFOCOM'99*, New York, NY, USA, March 1999.

[36] F. Gray. Pulse Code Communication. U.S. Patent 2,632,05, March 1953.

[37] Half Life. http://en.wikipedia.org/wiki/Half-Life_(video_game).

[38] Halo. http://halo.xbox.com/.

[39] M. Harchol-Balter, T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. In *PODC '99*, Atlanta, Georgia, USA, May 1999.

[40] M. Hazas, J. Scott, and J. Krumm. Location-Aware Computing Comes of Age. *Computer*, 37(2):95–97, 2004.

[41] T. Henderson. *The Effects of Relative Delay in Networked Games*. PhD Thesis, 2003.

[42] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: A Scalable Peer-to-Peer Network for Virtual Environments. *Network, IEEE*, 20(4):22–31, 2006.

[43] R. L. Huard. *Plato's Political Philosophy: the Cave*. Algora, 1999.

[44] R. Huebsch, J. M. Hellerstein, N. Lanham, B. Thau, L. S. Shenker, and I. Stoica. Querying the Internet with Pier. In *VLDB'03*, pages 321–332, 2003.

[45] HyperCast. http://www.hypercast.org.

[46] Institute of Electrical and Electronic Engineers. 1516-2000, IEEE Standard for Modeling and Simulation High Level Architecture – Framework and Rules. IEE, September 2000. New York, NY, USA.

[47] K. James and C. Mark. Traffic Analysis of Avatars in Second Life. In *NOSDAV'08*, Braunschweig, Germany, May 2008.

[48] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A. L. Barabási. The Large-Scale Organization of Metabolic Networks. *Nature*, 407(6804):651–654, October 2000.

[49] J.-R. Jiang, Y.-L. Huang, and S.-Y. Hu. Scalable AOI-Cast for Peer-to-Peer Networked Virtual Environments. In *ICDCSW'08*, Washington, DC, USA, 2008.

[50] K. Joh. What is a Role-Playing Game? http://www.darkshire.net/~jhkim/rpg/whatis/.

[51] Kaneva. http://www.kaneva.com/.

[52] J. Keller and G. Simon. SOLIPSIS: A Massively Multi-Participant Virtual World. In *PDPTA'03*, pages 262–268, 2003.

[53] F. Kevin. A Delay-Tolerant Network Architecture for Challenged Internets. In *SIGCOMM'03*, Karlsruhe, Germany, August 2003.

[54] T. Klingberg and R. Manfredi. *Gnutella 0.6*. Network Working Group, June 2002.

[55] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *INFOCOM'04*, Hong Kong, China, March 2004.

[56] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. SETI@home-Massively Distributed Computing for SETI. *Computing in Science and Engineering*, 3(1):78–83, 2001.

[57] E. Kranakis, H. Singh, and J. Urrutia. Compass Routing on Geometric Networks. In *CCCG*, pages 51–54, Vancouver, BC, CA, August 1999.

[58] S. Kumar, J. Chhugani, C. Kim, D. Kim, A. Nguyen, P. Dubey, C. Bienia, and Y. Kim. Second Life and the New Generation of Virtual Worlds. *IEEE Computer*, 41(9):46–53, 2008.

[59] C.-A. La and P. Michiardi. Characterizing User Mobility in Second Life. In *WOSN'08*, Seattle, WA, USA, August 2008.

[60] H. Liang, R. N. D. Silva, W. T. Ooi, and M. Motani. Avatar Mobility in User-Created Networked Virtual Worlds: Measurements, Analysis, and Implications. *To Appear in Multimedia Tools and Applications*, 2009.

[61] libsecondlife. `http://www.libsecondlife.org`.

[62] J. Liebeherr, M. Nahas, and W. Si. Application-layer Multicasting with Delaunay Triangulation Overlays. *Selected Areas in Communications, IEEE Journal on*, 20(8):1472–1488, October 2002.

[63] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *IPTPS'02*, Cambridge, MA, USA, March 2002.

[64] G. A. Miller. The Magical Number Seven, Plus or Minus two: Some Limits on Our Capacity for Processing Information. *Psychological Review*, 63:81–97, 1956.

[65] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC'07*, San Diego, CA, USA, October 2007.

[66] Mobility Models. `http://ica1www.epfl.ch/RandomTrip/`.

[67] A. Mtibaa, A. Chaintreau, J. LeBrun, E. Oliver, A.-K. Pietilainen, and C. Diot. Are You Moved by Your Social Network Application? In *WOSP'08*, New York, NY, USA, August 2008.

[68] NIST. Secure Hash Standard. Federal Information Processing Standard, FIPS-180-1, April 1995.

[69] M. Ohnishi, R. Nishide, and S. Ueshima. Incremental Construction of Delaunay Overlaid Network for Virtual Collaborative Space. *CCCCC'05*, 0:75–82, January 2005.

[70] Orkut. `http://www.orkut.com`.

[71] Overnet. `http://www.overnet.org/`.

[72] Oxford. *Dictionary of Computing*. Oxford University Press, 1996.

[73] J. Pang, F. Uyeda, and J. R. Lorch. Scaling Peer-to-Peer Games in Low-Bandwidth Environments. In *IPTPS'07*, Bellevue, WA, USA, February 2007.

[74] Planetlab. https://www.planet-lab.org/.

[75] PS3 Home. http://www.homebetatrial.com/.

[76] J. M. Pullen and D. C. Wood. Networking Technology and DIS. *IEEE*, 83(8):1156–1167, August 1995.

[77] Quake. http://quake.com/.

[78] B. Quinn and K. Almeroth. IP Multicast Applications: Challenges and Solutions, 2001.

[79] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief Announcement: Prefix Hash Tree. In *PODC'04*, page 368, 2004.

[80] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *SIGCOMM'01*, San Diego, CA, USA, October 2001.

[81] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware'01*, London, UK, November 2001.

[82] A. Rowstron, A. Kermarrec, M. Castro, and P. Druschel. Scribe: The Design of a Large-Scale Event Notification Infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[83] Second Life (SL). http://www.secondlife.com/.

[84] SL Blog. http://blog.secondlife.com/.

[85] SL Login Server. http://secondlife.com/app/login/.

[86] SL Statistics. http://secondlife.com/statistics/economy-data.php.

[87] M. Steiner and E. W. Biersack. DDC: A Dynamic and Distributed Clustering Algorithm for Networked Virtual Environments based on P2P networks. In *Global Internet Symposium '06*, Barcelona, Spain, April 2006.

[88] M. Steiner, E. W. Biersack, and T. En Najjary. Actively Monitoring Peers in KAD. In *IPTPS'07*, Bellevue, WA, USA, February 2007.

[89] M. Steiner, D. Carra, and E. W. Biersack. Faster Content Access in KAD. In *P2P'08*, Aachen, Germany, September 2008.

[90] M. Steiner, T. En Najjary, and E. W. Biersack. A Global View of KAD. In *IMC'07*, San Diego, CA, USA, October 2007.

[91] M. Steiner, T. En-Najjary, and E. W. Biersack. Long Term Study of Peer Behavior in the KAD DHT. *IEEE/ACM Transactions on Networking*, 17(6), 2009.

[92] F. Stênio, K. Carlos, S. Djamel, M. Josilene, and A. Rafael. Traffic Analysis Beyond This World: the Case of Second Life. In *NOSDAV'07*, Urbana-Champaign, IL, USA, June 2007.

[93] N. Stephenson. *Snow Crash*. Bantam Spectra Book, 1992.

[94] D. Stutzbach and R. Rejaie. Understanding Churn in Peer-to-Peer Networks. In *IMC'06*, Rio de Janeiro, Brazil, October 2006.

[95] The Sims Online. `http://www.thesimsonline.com/`.

[96] There. `http://www.there.com/`.

[97] Ultima Online. `http://www.uoherald.com/news/`.

[98] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *OSDI'02*, Boston, MA, USA, December 2002.

[99] G. Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufman, 2006.

[100] M. Varvello, E. Biersack, and C. Diot. A Networked Virtual Environment over KAD. In *CONEXT'07, Ext. Abstract*, New York, NY, USA, December 2007.

[101] M. Varvello, E. Biersack, and C. Diot. Dynamic Clustering in Delaunay-Based P2P Networked Virtual Environments. In *SIGCOMM'07, Ext. Abstract*, Kyoto, Japan, September 2007.

[102] M. Varvello, E. Biersack, and C. Diot. Dynamic Clustering in Delaunay-Based P2P Networked Virtual Environments. In *NETGAMES'07*, Melbourne, Australia, September 2007.

[103] M. Varvello, C. Diot, and E. Biersack. A Walkable Kademlia Network for Virtual Worlds. In *INFOCOM'09, Ext. Abstract*, Rio De Janeiro, Brazil, April 2009.

[104] M. Varvello, C. Diot, and E. Biersack. A Walkable Kademlia Network for Virtual Worlds. In *IPTPS'09*, Boston, MA, USA, April 2009.

[105] M. Varvello, C. Diot, and E. Biersack. P2P Second Life: experimental validation using Kad. In *INFOCOM'09*, Rio De Janeiro, Brazil, April 2009.

[106] M. Varvello, S. Ferrari, E. Biersack, and C. Diot. Distributed Avatar Management for Second Life. In *NETGAMES'09*, Paris, France, November 2009.

[107] M. Varvello, F. Picconi, C. Diot, and E. Biersack. Is There Life in Second Life? In *CONEXT'08*, Madrid, Spain, December 2008.

[108] D. J. Watts. *Small Worlds: The Dynamics of Networks between Order and Randomness*. Princeton University Press, 1999.

[109] D. J. Watts and S. H. Strogatz. Collective Dynamics of 'small-world' Networks. *Nature*, 393(6684):440–442, June 1998.

[110] B. Wietrzyk and M. Radenkovic. Enabling Rapid and Cost-Effective Creation of Massive Pervasive Games in Very Unstable Environments. In *WONS'07*, Obergurgl, Austria, January 2007.

[111] Wikipedia. `http://en.wikipedia.org/wiki/Real_estate_(Second_Life)`.

[112] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User Interactions in Social Networks and their Implications. In *EuroSys'09*, Nuremberg, Germany, October 2009.

[113] J. Winick and S. Jamin. Inet-3.0: Internet Topology Generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.

[114] Wolfenstein. `http://www.idsoftware.com/`.

[115] World of Warcraft. `http://www.worldofwarcraft.com/`.

[116] J. Yan and B. Randell. A Systematic Classification of Cheating in Online Games. In *NETGAMES'05*, Hawthorne, NY, USA, October 2005.

APPENDICES

ANNEXE A

List of Abbreviations

Table A.1 describes various abbreviations and acronyms used throughout the thesis. The page where each abbreviation is defined or first used is also given. Nonstandard acronyms used in the thesis to abbreviate the names of certain variables and structures are not in this list.

Abbreviation	Meaning	Page
AoI	Area of Interest	13
C/S	Client/Server	5
CCDF	Complementary Cumulative Distribution Function	27
CDF	Cumulative Distribution Function	30
CH	Cluster Head	109
DHT	Distributed Hash Table	9
FPS	First Person Shooter	2
MOG	Multi-player Online Game	2
NVE	Networked Virtual Environment	2
P2P	Peer-to-Peer	6
POI	Point-of-Interest	36
QoE	Quality of Experience	19
RPG	Role Playing Game	2
SDN	Social Delaunay Network	116
SL	Second Life	2
SVW	Social Virtual World	2

TAB. A.1: List of Abbreviations

ANNEXE B

Synthèse en français

B.1 A la Decouverte de Second Life

Au cours des ces dernières années, nous avons observé une croissance rapide des *mondes virtuels sociaux*. Ce sont des Environnements Virtuels en Réseau (NVEs) où les utilisateurs peuvent se rencontrer, jouer, vendre, acheter et même contribuer au développement du NVE. *Second Life* (SL), lancé en 2003 par Linden Lab, est devenu le monde virtuel social le plus populaire, enregistrant jusqu'a 14 millions d'utilisateurs en Juin 2008.

SL est composé d'un terrain virtuel, divisé en régions de tailles fixes, où les utilisateurs interagissent par leurs représentants numériques appelés *avatars*. La principale caractéristique innovatrice de SL est le contenu crée par l'utilisateur : les avatars peuvent participer au développement de l'environnement virtuel en créant des *objets* tels que des voitures, des murs, des arbres et des bâtiments. En outre, SL a créé une économie épanouie en attirant plusieurs sociétés qui ont investi des millions de dollars afin de construire leurs propres produits virtuels et de la publicité.

Étant donné le manque d'informations sur SL, nous avons décidé de mener une grande étude des caractéristiques de SL. Pour ce faire, nous concevons et développons un *crawler* et un *player*. Ce sont des applications tierces pour SL qui exploitent les capacités standards des avatars pour : (1) obtenir des informations sur le monde virtuel, par exemple, les comportements des avatars et les performances du serveur, et (2) étudier la Qualité d'Expérience (QoE) de l'utilisateur. L'analyse des traces recueillies nous permet de faire des observations sur les caractéristiques de SL.

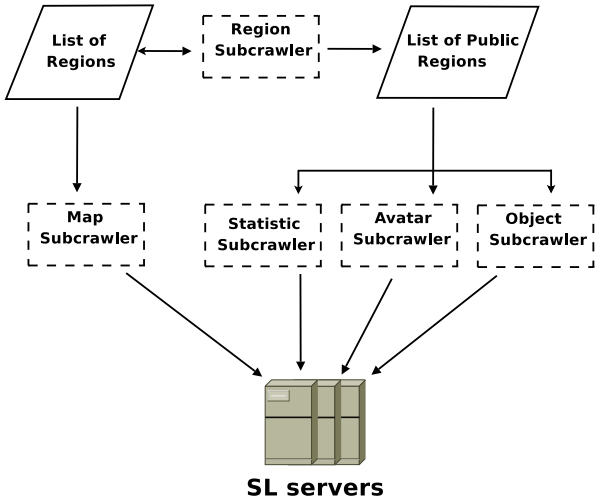


FIG. B.1: Architecture du crawler.

B.1.1 Méthodologie

Crawler L'idée principale derrière notre crawler est d'exploiter les capacités standards des avatars pour obtenir des informations sur le monde virtuel. Notre crawler est composé de plusieurs *subcrawlers*, chacun étant spécialisé dans une tâche de surveillance différente (Figure B.1).

Chaque subcrawler est une version modifiée du client SL mis en oeuvre en utilisant les bibliothèques *libsecondlife*, un projet (logiciel libre) qui vise à comprendre le protocole utilisé par SL. Un subcrawler doit être associé à un avatar inscrit sur le site SL afin de pouvoir se connecter au monde virtuel. Nous utilisons plusieurs instances de chaque subcrawler (associés à différentes identités des avatars) afin de paralléliser l'exploration. Nous décrivons, dans ce qui suit, le rôle de chaque subcrawler :

- Le *Region subcrawler* surveille SL afin de maintenir à jour une liste de ses régions. Cette information est dynamiquement mise à jour puisque des nouvelles régions peuvent être ajoutées au monde virtuel de SL.
- L'*Object subcrawler* suit l'évolution des objets dans toutes les régions publiques.
- Le *Statistics subcrawler* recueille les statistiques tenues par les serveurs des régions publiques.
- Le *Map subcrawler* surveille la position des avatars comme indiqué sur la carte SL.
- L'*Avatar subcrawler* obtient l'identité et la position des avatars qui se trouvent à l'interieur des régions publiques.

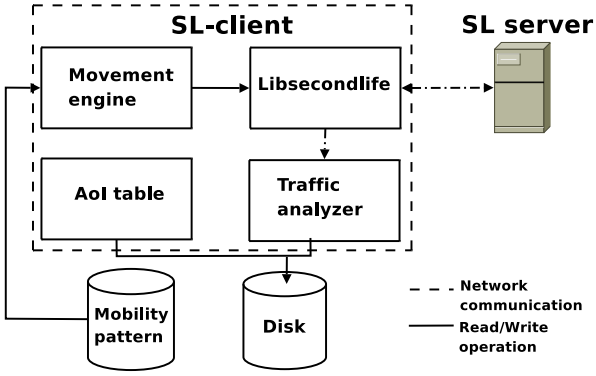


FIG. B.2: Architecture du player.

Player Nous reproduisons les comportements des avatars dans SL via des clients contrôlés afin d'évaluer la QoE de l'utilisateur. Nous utilisons libsecondlife afin de mettre en oeuvre un *player* qui automatise le comportement d'un avatar tout en récoltant des traces liées à la QoE perçu par son utilisateur.

Le *player* effectue la connexion d'un avatar à une région cible et déplace l'avatar dans cette région selon un modèle de mobilité. La figure B.2 montre l'architecture du player.

B.1.2 Analyse de Second Life

La Table B.1 résume chaque configuration des subcrawlers, la longueur des traces récoltées, et la fréquence du crawl.

subcrawler	Instances	IP@s	Régions	Fréquences	Jours
Region	3	1	-	1/24 hrs	28
Object	5	1	-	1/24 hrs	28
Statistics	60	1	12,765	1/90 min	6
Map	40	1	17,526	1/15 min	3

TAB. B.1: Synthèse du crawling de Second Life.

La Table B.2 résume le nombre total des régions découvertes par le Region subcrawler, ainsi que le nombre officiel rapporté par le site SL.

Nous constatons que le Region subcrawler découvre un nombre de régions plus grand que celui des chiffres officiels. Ces régions supplémentaires ne sont pas joignables et ont été découvertes à proximité de celles qui sont actives. Par conséquent, elles sont probablement une partie de SL réservé à l'avenir, et donc ne comptent pas dans les statistiques officielles.

	Mars 29	Avril 18	Avril 25
Régions Publiques (RS)	12,765	13,220	13,261
Régions (RS)	17,280	17,526	17,573
Régions (SLW)	13,693	N/A	14,150

TAB. B.2: Nombre de régions dans SL (SR = Region subcrawler, SLW=site internet de Second Life).

La Figure B.3 montre l'évolution dans le temps du nombre d'utilisateurs en ligne, tels qu'ils ont été mesurés par le Map subcrawler et déclarés par le Login Server (ces données sont obtenues en contrôlant le Login Server). Les deux courbes présentent le même cycle quotidien. Cependant, le Login Server rapporte de 10,000 a 20,000 plus d'utilisateurs que le Map subcrawler. En outre, pendant une panne majeure, vendredi, à 14h00, le Login Server a signalé une baisse de 10,000 avatars, alors que notre Map subcrawler a observé une diminution de 20,000 avatars. Ceci signifie que les valeurs fournies par le Login Server peuvent être erronées, et probablement calculées sur une moyenne sur une longue période.

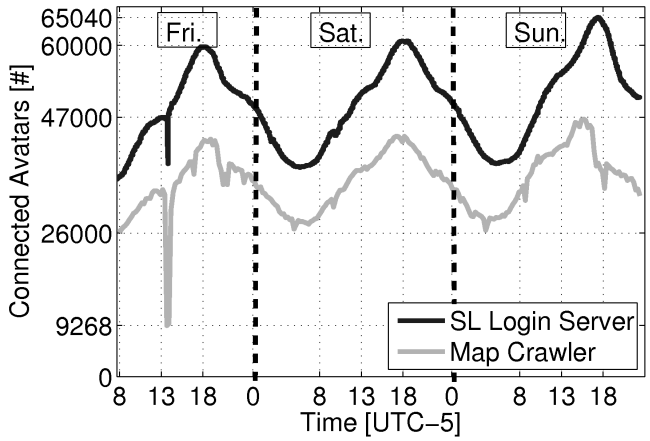


FIG. B.3: Population active au cours du temps [Map subcrawler ; SL Login Server ; Temps universel coordonné - 5].

Nous analysons maintenant l'évolution du nombre d'objets dans le temps. Pour chaque région, nous calculons la différence entre le nombre d'objets qu'elle contient le jour j, et son nombre d'objets initiaux observés le premier jour de la surveillance (29 Mars, 2008). La Figure B.4 montre quelques centiles significativs de la distribution des différences mesurées pour toutes les régions (l'écart entre les jours 6 et 9 est dû à une panne de SL). Nous observons que 50% des régions (entre les 25ème et 75ème centiles) sont presque statiques, montrant une faible variation entre ± 50 objets après 28 jours. Les 10ème et 90ème centiles restent entre ± 250 objets, montrant un taux modeste de variation d'objets dans la plupart des régions. La valeur médiane est proche de zéro, et les centiles sont presque symétriques, indiquant un taux de création d'objets similaires au taux de destruction.

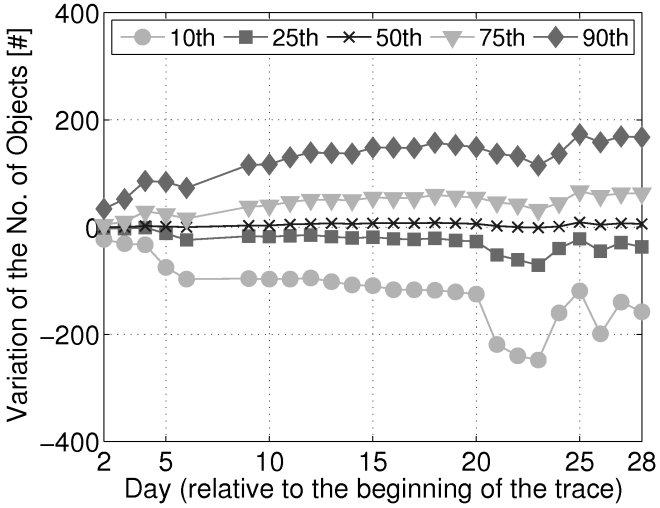


FIG. B.4: Centiles de la distribution de la variation de nombre d'objets par région ; [Object subcrawler].

B.1.3 Qualité d'Expérience des Utilisateurs

Nous exécutons le player sur plusieurs machines PlanetLab situées dans le monde entier afin de simuler des conditions réalistes du réseau. Nous obtenons des traces de la mobilité des avatars en surveillant la région Japan Resort, une région très populaire de SL. Nous gérons nos expériences sur trois régions moins populaires, c'est-à-dire généralement vides d'avatars, et qui représentent une diversité dans la composition en objets observés dans SL. Nous sélectionnons respectivement une région à *faible densité* (6 objets), une région à *densité moyenne* (130 objets) et une région à *haute densité* (541 objets) , selon le nombre d'objets qu'ils contiennent. Au cours des expériences, nous avons aussi vérifié en permanence qu'aucun utilisateur externe ne se connecte à la région et n'interfère à nos mesures.

Nous évaluons l'**incohérence** de l'Air d'Intérêt (AoI) de chaque avatar tous les $200\ ms$ ou à chaque modification de l'AoI. La Figure B.5 montre les Complementary Distribution Functions (CDFs) du rapport entre la somme des durées des périodes d'incohérence pour un avatar, et le temps total durant lequel l'avatar reste connecté à une région. On constate que dans une région à haute densité, les avatars ont un AoI plus incohérent que dans les deux autres régions. La Figure B.5 montre que les avatars avec un AoI totalement incohérent sont tout aussi probables dans les trois régions, ex., environ 8% des avatars ont un AoI incohérent pendant environ $80\% - 90\%$ de leur voyage dans SL. La raison de ce phénomène est qu'un serveur SL passe beaucoup de temps à accomplir correctement la connexion d'un avatar. Par la suite, les avatars dont les temps de session sont très courts ont un AoI incohérent la plupart du temps.

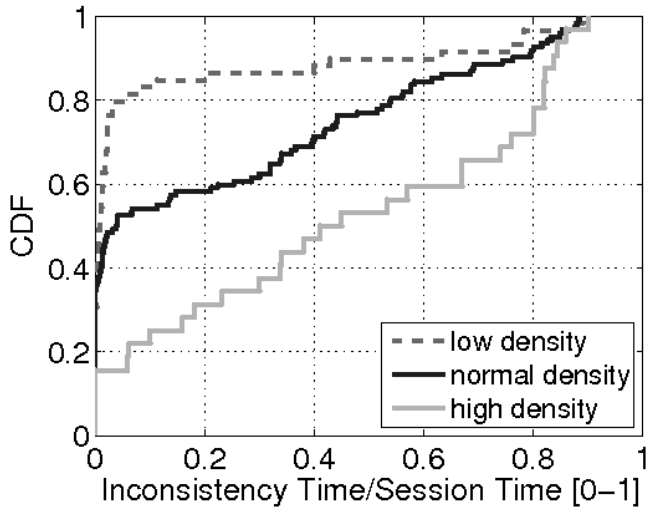


FIG. B.5: CDFs de la fraction du temps que l'AoI d'un avatar est incohérent.

Nous analysons maintenant la **durée d'incohérence** mesurée dans les trois régions afin de comprendre à quelle vitesse les serveurs SL réagissent aux incohérences des avatars. La Figure B.6 montre les CDFs de la durée d'incohérence mesurées dans les trois régions. Nous remarquons que les avatars ont des AoIs incohérents pour plus d'une seconde en 40%-50% des cas. En plus, 5% à 10% de ces incohérences durent plus de 5 secondes. Ces valeurs d'incohérence de très longue durée sont intolérables pour les utilisateurs si on considère que les valeurs acceptables de latence dans les mondes virtuels varient entre $300\ ms$ et $1\ sec$, dévoilant que les serveurs SL assurent une faible interactivité à leur avatars. La Figure B.6 montre également que la durée d'incohérence mesurée dans les trois régions peut atteindre une valeur maximale d'environ 20 secondes. Ces valeurs extrêmement élevées sont mesurées en présence de *churn*, c.à.d. connexion/déconnexion des avatars.

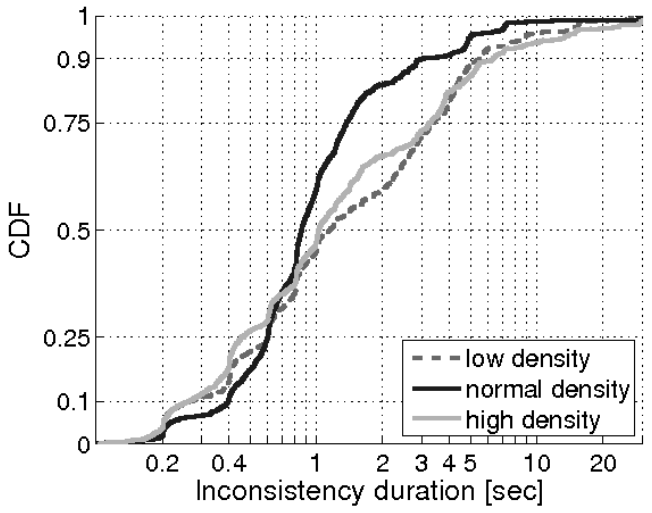


FIG. B.6: CDFs de la durée d'incohérence.

Enfin, nous analysons à quelle vitesse les avatars récupèrent les objets situés dans leur AoIs, c.à.d. la **latence de découverte**. La Figure B.7 montre les CDFs de la latence de découvert mesurée dans les trois régions. Sans surprise, plus la densité du contenu dans une région est élevée, plus il faut de temps à un avatar pour reconstruire le monde virtuel dans son AoI. Le temps de latence médian de découverte passe d'environ 4 secondes dans la région à faible et moyenne densité à environ 30 secondes dans la région à haute densité. Dans nos expériences, nous vérifions que la connexion des utilisateurs ne soit pas un goulot d'étranglement. Par conséquent, ce résultat indique que ces très longues latences de découverte mesurées dans la région de haute densité sont dues au fait que le serveur limite le taux de son trafic sortant. Toutefois, la Figure B.7 montre également que les courbes des régions à basse et moyenne densité ont aussi des valeurs de latence importantes ex., supérieures à 20 secondes.

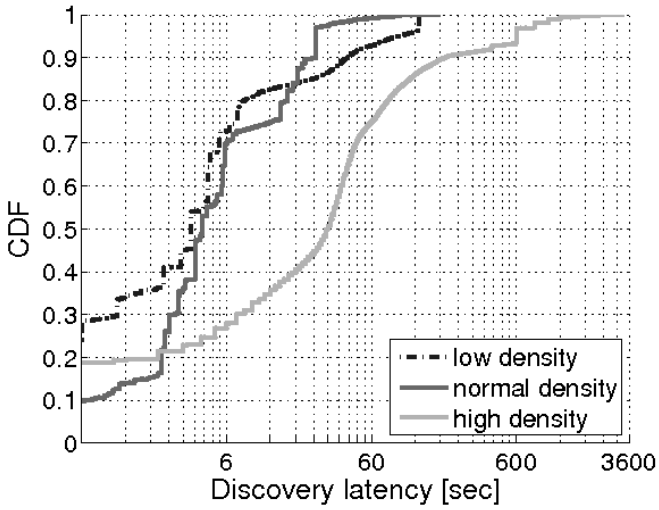


FIG. B.7: CDFs de la latence de découverte.

B.2 Gestion Distribuée des Objets

Les NVEs commerciales comme Second Life sont mis en oeuvre en utilisant une architecture Client/Serveur (C/S). Un serveur stocke une copie de tous les objets qui résident sur les terres virtuelles pendant que les clients exécutent des applications qui permettent à ces utilisateurs d'explorer le monde virtuel par leurs avatars. Pour cela, les clients envoient des *range queries* vers le serveur, c'est à dire, des demandes pour les objets dont les coordonnées spatiales sont situées dans un secteur donné. Dans la pratique, les avatars identifient l'ensemble des objets (par exemple, des arbres ou des voitures) situés dans leur environs en envoyant au serveur une range query avec range égale à la zone de visibilité de l'avatar, c.à.d. son AoI.

Les ranges queries dans les mondes virtuels peuvent être divisés en *locales* et *non-locales*. Une requête locale consiste en une demande d'objets situé dans les environs d'un avatar. Par exemple, les avatars génèrent des requêtes locales quand ils marchent, courent ou volent, afin de mettre constamment à jour leur AoIs. Une requête non-locale est une demande d'objets situés loins de l'avatar. Par exemple, les avatars génèrent des requêtes non-locales lorsqu'ils couvrent tout à coup une grande distance via l'opération de téléportation. Les requêtes locales doivent recevoir une réponse rapide pour assurer une bonne expérience de l'utilisateur. Inversement, un retard supérieur pour répondre à des requêtes non-locales peut être tolérable.

Les requêtes locales et non-locales dans les mondes virtuels sont faciles à gérer avec une architecture C/S. Cependant, cette architecture a une faible scalabilité et un coût très élevé. Les Distributed Hash Tables (DHTs) sont des alternatives à bon marché et très scalables. Les DHTs sont des architectures Pair-à-Pair (P2P) utilisées pour stocker et récupérer du contenu. Toutefois, les DHTs actuelles ne permettent pas de gérer des range queries.

Dans cette thèse, nous concevons un simple mécanisme pour la gestion des objects d'un NVE et nous utilisons ce mécanisme pour conduire des experimentation par Internet. Nous ne discutons pas ces experimentations. Successivement, nous concevons et évaluons *Walkad*, une architecture P2P pour la gestion des range queries dans les mondes virtuels. Nous concevons Walkad comme une extension de *Kademlia*, un DHT très populaire adoptée avec succès par eMule. Walkad organise le keyspace Kademlia dans un *reverse bynarie trie*, c'est à dire, une structure à arbre où les noeuds de chaque niveau de l'arbre sont étiquetés en utilisant le code de Gray.

Nous évaluons Walkad via l'émulation du réseau, et en utilisant dès traces d'objets obtenus dans Second Life. Des traces synthétiques sont utilisées pour simuler les mouvements des avatars afin d'étudier différents types de range queries. Nos résultats montrent que Walkad est une conception P2P très efficace pour les mondes virtuels. En fait, Walkad garantit à ses utilisateurs une découverte rapide du monde virtuel, tout en redistribuant la charge équitablement entre les pairs.

B.2.1 Description

On appelle *cellule* une partie du monde virtuel, et *cell-ID* une clé Kademlia associée à une cellule. Nous disons qu'à l'origine, le monde virtuel était composé d'une seule cellule. Ensuite, le monde virtuel est récursivement divisé en plusieurs cellules dés que le nombre d'objets devient supérieur à un seuil D_{max}. Nous disons que deux cellules sont *voisines* si : (1) elles ont un côté en commun, ou (2) elles sont symétriques selon l'axe précédemment utilisé dans les opérations de fractionnement. De même, nous disons que les identifiants des deux cellules sont *voisins* quand ils ont une dis-

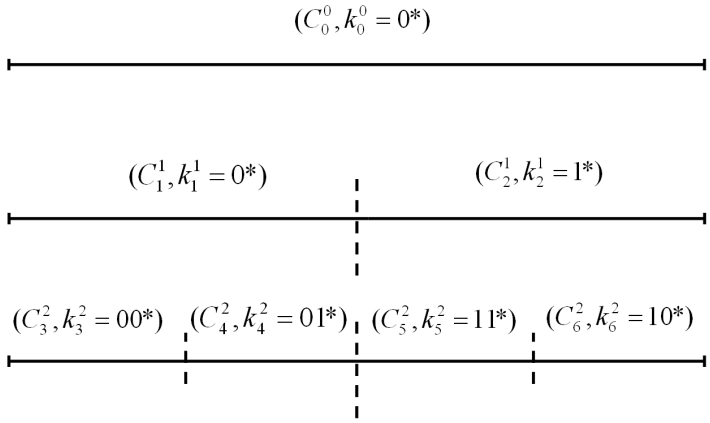


FIG. B.8: Monde virtuel avec une seule dimension indexée par Walkad.

tance d'Hamming égal à un, c'est-à-dire, quand ils ne diffèrent que par un seul bit. Par définition, un cell-ID avec l bits significatifs a l cell-IDs voisins. Pour illustrer cela, nous considérons un exemple d'un monde virtuel avec une seule dimension (Figure B.8). Nous désignons la i-ème cellule/cell-ID généré par L divisions respectivement C_i^l et k_i^l.

La partie supérieure de la Figure B.8 montre l'organisation initiale du monde virtuel. A ce stade, il y a une seule cellule, C_0^0, qui couvre le monde virtuel tout entier. La partie centrale de la Figure B.8 montre la configuration du monde virtuel après la première division, où deux nouvelles cellules ont été creéés, C_1^1 et C_2^1. Ces deux cellules sont adjacentes et donc voisines. Le partie inférieure de la Figure B.8 montre le résultat d'un nouveau fractionnement de cellules. Considérons la cellule C_3^2. Ses cellules voisines sont : la cellule C_4^2, qui est adjacente, et la cellule C_6^2, qui est symétrique de C_3^2 conformément à la longue ligne pointillée.

Walkad organise les cell-IDs dans un *reverse binary trie* pour associer à des cellules voisines des cell-IDs voisins. Dans un reverse binary trie, les noeuds de chaque niveau de l'arbre sont étiquetés avec le Gray Code, un système de numérotation binaire où deux valeurs successives ont une distance de Hamming égal à 1.

Nous expliquons maintenant comment nous organisons les cell-IDs dans un reverse binary trie en considérant l'exemple de la Figure B.8. La cell C_0^0 est attribué à cell-ID $k_0^0 = 0*$. Lorsque C_0^0 est divisée, nous générons les nouvelles cell-IDs en prenant $k_0^0 = 0*$, et en définissant le bit le moins significatif égal respectivement à 0 et 1. Nous avons donc obtenu deux nouvelles cell-IDs $k_1^1 = 0*$ et $k_2^1 = 1*$, qui ont une distance d'Hamming égale à 1. L'intuition est que pour construire un reverse binary trie dans certains cas nous devons inverser les bits ajoutés. La Figure B.8 montre que fractionner la cellule C_2^1 demande de mettre le bit le moins significatif à 0 pour $k_6^2 = 10*$ et à 1 pour $k_5^2 = 11*$ afin de garantir que les cellules C_3^2 et C_6^2, qui sont voisins, soient associées à des cell-IDs qui sont voisins aussi.

L'algorithme d'indexation que nous avons décrit organise les cell-IDs dans le key-space comme un arbre. Par conséquent, un arbre déséquilibré génère une charge déséquilibrée entre les pairs. Afin de rétablir l'uniformité dans la distribution des cell-IDs, nous divisons le monde en régions (comme dans Second Life), et nous associons à chaque région, un *region-ID*. Puis, nous effectuons une opération XOR entre le cell-IDs et le région-ID. De cette manière, la distance d'Hamming entre les cell-IDs de la même région ne change pas. Cependant l'équilibrage de charge est atteint entre les cell-IDs des différentes régions.

Walkad and Kademlia

Nous appelons *coordonnateur* un pair responsable pour une cellule. Les coordinateurs pour une cellule C_i^l indexée par le cell-ID k_i^l sont les XOR plus proches pairs définis par Kademlia. Pour chaque (cell/cell- ID) pair il existe R coordonnateurs. Un pair Kademlia garde pour chaque $0 \leq i < 160$ bits de ses identifiant un *k-bucket*, c.à.d. une liste de pairs avec une distance $2^i \leq d < 2^{(i+1)}$ à partir de lui même. Les entrées de n^{eme} k-bucket ont le n^{eme} bit différent avec l'identifiant de pair. Par conséquent, un coordinateur pour une cell-ID k_i^l garde dans l k-bucket différents les informations de routage vers les coordonnateurs des l cell-IDs voisins.

Lorsqu'une cellule est divisée, ses coordinateurs sélectionnent les coordonnateurs des nouvelles sous-cellules via une recherche Kademlia. Ensuite, ils transfèrent à ces coordinateurs une liste des (cellules/cell-IDs) pairs existants, et l'ensemble des objets situés dans la nouvelle cellule. Un pair sélectionné pour être un coordinateur pour un cell-ID k_i^l effectue une recherche Kademlia pour chaque cellule voisine existant entre les l cell-IDs avec une distance d'Hamming égale à 1 à partir de k_i^l. Cette opération remplit le k-buckets du coordonnateur avec les information de routage vers les coordinateurs voisins.

Une range query présenté par un pair P est résolu comme suit. P envoie la requête à l'un des coordonnateurs de la cellule où son propre avatar est situé. Le coordonnateur répond à la requête ou à une partie de cette requête d'après les informations dont il dispose sur les cellules voisines. Ensuite, il renvoie à P les informations qu'il connaît, c'est à dire les informations de routage vers les coordonnateurs pour les cellules qui interceptent la portée de la requête. Dans le cas où un coordinateur n'a pas une vue complète de l'entière portée d'une requête, il transmet la requête aux coordonnateurs qui gèrent les cellules les plus proches de cette portée. Cette procédure se fait par itération jusqu'à ce que la portée soit complètement couverte. Enfin, P contacte les coordonnateurs responsables de la portée de la requête pour récupérer les objets situés dans cette partie du monde virtuel.

B.2.2 Evaluation Experimentale

Nous évaluons Walkad sur un cluster local en utilisant Modelnet et une topologie Internet synthétique générée par Inet. Nous utilisons une configuration classique Kademlia avec k-buckets de taille $k = 20$, et $R = 10$. Nous générons un monde virtuel réaliste en utilisant des traces d'objets obtenus à l'intérieur de cinq régions Second Life très populaires.

Nous avons d'abord évalué la **latence**, c'est à dire le temps nécessaire pour répondre aux range queries, en fonction de la taille du réseau N et du type de range query. Pour générer des range queries différentes, nous simulons des avatars qui marchent, courent, volent et se téléportent, selon le modèle Random Waypoint. Dans toutes ces expériences, nous avons fixé D_{max} à 10. La Figure B.9 montre que les range queries générées par un avatar qui marche, court ou vole sont toutes résolves en environ 100 à 130 ms en moyenne. En effet, tous ces mouvements génèrent des range queries locales. Inversement, les requêtes non-locales générées par un avatar en téléportation nécessite environ le double du temps, par exemple, jusqu'à $200\ ms$ dans le pire des cas. La Figure B.9 montre également que l'ensemble de la latence augmente légèrement avec la taille du réseau N. En fait, le nombre de sauts de routage dépend de la taille du monde virtuel plutôt que de la taille du réseau. Toutefois, lorsque le réseau est très petit, des pairs deviennent coordinateurs de plusieurs cellules, ce qui réduit le nombre effectif de sauts de routage ainsi que la latence.

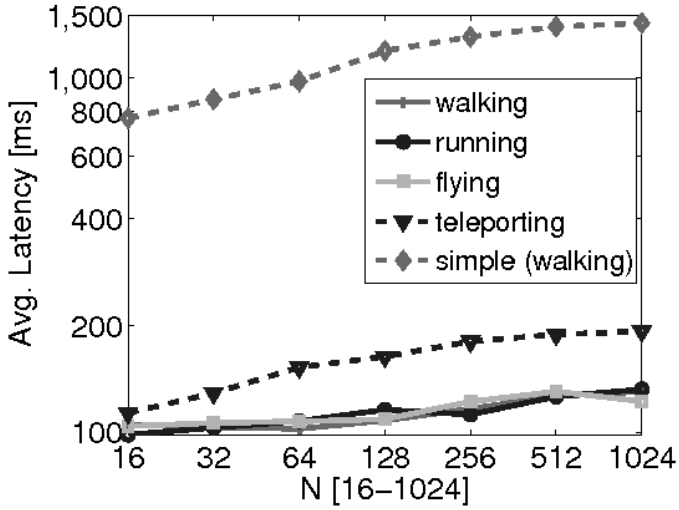


FIG. B.9: $D_{max} = 10$; $N = [16 - 1024]$; Avatar=$[Walk; Run; Fly; Teleport]$.

Nous analysons maintenant l'équilibrage de la charge, c-à-d., comment les cell-IDs sont distribueés entre les pairs (Figure B.10). Nous observons que dans la Figure B.9 la charge est répartie équitablement entre pairs quand le réseau grandit. Par exemple, lorsque $N \geq 256$, la différence dans la fraction de cell-IDs pour 90% des pairs est plus petite que 1%. Seulement 10% des pairs sont responsables d'une plus vaste frac-

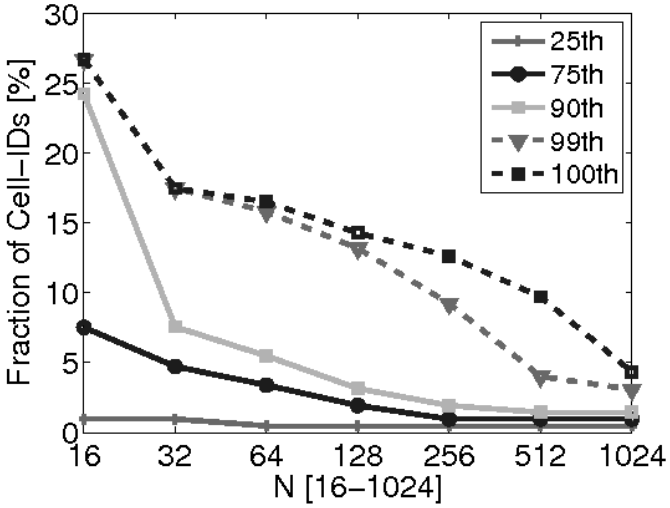


FIG. B.10: Centiles de la distribution de la fraction des cell-IDs par pair ; cinq régions Second Life ; $D_{max} = 10$; $N = [16 - 1024]$.

tion de cell-IDs. Cela est dû au fait que nous examinons un monde virtuel restreint composé seulement de cinq régions. Dans un monde virtuel composé de 100 régions nous avons mesuré que seulement 1% des pairs stockent une plus grande portion du monde virtuel.

B.3 Gestion Distribuée des Avatars

La caractéristique essentielle des NVEs est la possibilité pour ses utilisateurs de communiquer. La communication entre avatars est basée sur le geste, le chat ou même la voix. Afin d'assurer des intèractions agréable aux utilisateurs, l'architecture d'un NVE doit s'assurer que des informations correctes sont propagées parmi les avatars en temps réel. Cette tâche est appelée *gestion des avatars*.

Dans cette thèse, nous réalisons un effort majeur pour évaluer, comprendre et améliorer la gestion des avatars dans les NVEs en utilisant les ressources des utilisateurs. L'idée clé est que les pairs partagent une connexion dans le réseau P2P lorsque leurs avatars intéragissent dans le NVE. Cette tâche est accomplie en exploitant les coordonnées des avatars dans le monde virtuel afin de conduire la stratégie de sélection des pairs. Actuellement, l'approche du *réseau Delaunay* est le plus populaire approche de ce genre.

Au départ, nous évaluons expérimentalement la QoE qu'une gestion des avatars basée sur Delaunay fournit aux utilisateurs. Nous intégrons un module de Delaunay dans un client SL et effectuons des expérimentations en utilisant des machines PlanetLab comme hôtes et les régions SL comme aire de jeux. Nous utilisons des

traces obtenu dans SL pour simuler les mouvements réels des avatars. Nous montrons qu'une gestion distribuée des avatars pour SL garantit à ses utilisateurs une majeur réactivité ainsi qu'une expérience plus correcte par rapport à une approche centralisée.

Successivement, nous concevons et évaluons plusieurs optimisations pour des NVES fondées sur Delaunay : (1) un algorithme de clustering qui améliore la *réactivité* de l'NVE en présence de grands groupes d'avatars, et (2) le *réseau Delaunay social*, un réseau P2P pour NVEs qui enforce la sécurité des pairs en utlisant les relations d'amitié qui existent entre les avatars.

Dans ce qui suit, nous décrivons l'évaluation expérimentale du réseau Delaunay integré dans le client Second Life. Nous ne discutons pas les extensions Delaunay concus.

B.3.1 Le Réseau Delaunay

Le *réseau Delaunay* est un réseau "overlay" dont la topologie est définie par une triangulation de Delaunay. Dans ce qui suit, nous donnons une définition formelle de la triangulation de Delaunay :

Definition 2 *La triangulation de Delaunay d'un ensemble de points N dans $\Re^2$ est une triangulation des points $DT(N)$ tel qu'aucun point p ne réside à l'intérieur du cercle circonscrit d'aucun triangle en $DT(N)$.*

Les coordonnées des avatars dans le monde virtuel sont utilisées pour générer la triangulation de Delaunay, et par conséquent construir un réseau Delaunay entre les utilisateurs de l'NVE. De cette façon, les interactions entre avatars basés sur une métrique de proximité correspondent à des connexions Internet entre leurs utilisateurs finaux. La Figure B.11 montre un exemple d'une triangulation de Delaunay construit entre les avatars d'un monde virtuel.

Un noeud interne d'une triangulation de Delaunay à deux dimensions avec une population suffisamment grande, c.à.d. un noeud qui ne se trouve pas sur l' "enveloppe convexe"[1] de la triangulation, a un nombre moyen de voisins tel que six. Cela signifie que chaque pair dans le réseau Delaunay correspondant est connecté à un nombre fini de pairs indépendamment de la population de l'NVE.

Un avatar qui participe au réseau Delaunay surveille continuellement ses voisins afin de maintenir une triangulation correcte avec le temps. À mesure que les avatars

[1] L'enveloppe convexe d'un ensemble de points X dans un espace vectoriel $V \in \Re^k$ est l'ensemble e minimum convexe contenant X.

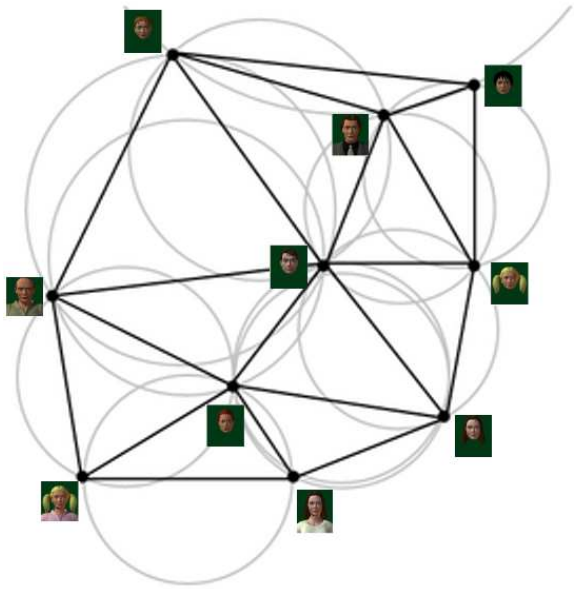


FIG. B.11: Triangulation de Delaunay entre les avatars d'un monde virtuel.

bougent, les liens Delaunay sont ajoutés et supprimés pour maintenir une triangulation valide et cohérente.

B.3.2 Le Client P2P-SL

Nous utilisons la bibliothèque libsecondlife pour déployer un client P2P-SL. Le client P2P-SL inclut les caractéristiques fondamentales du client SL officiell, ex., les operations de connexion/déconnexion et les mouvements des avatars, sans les operations qui causent une utilisation intensive du processeur, ex., le rendu en trois dimensions du monde virtuel.

Le client P2P-SL n'a pas besoin d'être contrôlé par l'homme. Par exemple, les traces des avatars contenant leur mouvements ainsi que leur durée de session peuvent être utilisées pour automatiser les opérations du client. L'aspect innovateur du client P2P-SL est la possibilité de communiquer directement avec d'autres clients P2P-SL sans la nécessité d'un serveur central.

Afin de permettre une communication directe entre les utilisateurs de SL, le client P2P-SL comprend un module qui implémente le protocol Delaunay. Nous développons le module réseau Delaunay en utilisant HyperCast[2], un ensemble de bibliothèques Java qui permettent à un programmeur de concevoir facilement des applications distribuées exploitant différents types de réseau P2P, ex., le réseau Delaunay et Pastry. HyperCast maintient une *table de voisinage* qui contient des informations de

[2]http ://www.hypercast.org

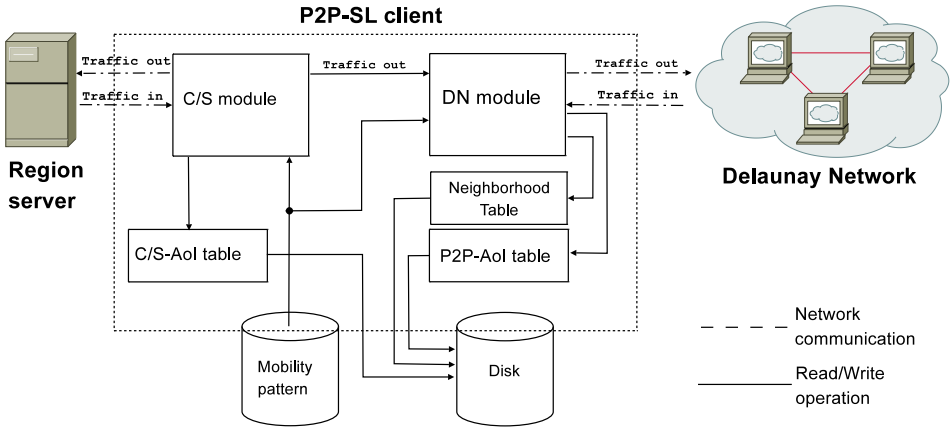


FIG. B.12: Le client P2P-SL.

routage vers les voisins dans l'overlay, ex., les adresses IP des voisins de Delaunay. La maintenance du réseau P2P est assurée par des messages de heartbeat que chaque pair envoie aux pairs existant dans sa table de voisinage.

Nous construisons une gestion distribuée des avatar pour SL au dessus du réseau Delaunay mis en place chez les clients P2P-SL. Pour cela, nous interceptons les mises à jour de l'état de l'avatar générées par le client et nous les dupliquons dans le réseau Delaunay. La diffusion des mises à jour de l'état d'un avatar est effectuée sur le réseau Delaunay exploitant une communication UDP. La motivation de ce choix est que notre client P2P-SL n'a pas besoin de garanties sur la livraison des paquets. La Figure B.12 montre une description graphique du client P2P-SL.

B.3.3 Evaluation

Nous comparons une architecture P2P et C/S pour Second Life en se concentrant sur la QoE perçue par des utilisateurs de SL automatisés. On lance le client P2P-SL sur plusieurs machines PlanetLab et nous alimentons avec nos avatars contrôlés une région SL vide d'objets. De cette façon, les objets n'interfèrent pas avec le modèle de mobilité des avatars. Nos avatars automatisés reproduisent une mobilité réelle en utilisant une trace recueillie dans la region SL appelée Japan Resort. Nous reproduisons le comportement de 207 avatars différents au cours d'une heure.

Nous calculons l'**l'incohérence** et la **durée d'incohérence** afin de comparer la QoE perçue par les utilisateurs SL respectivement dans une architecture C/S et P2P. Nous commençons par examiner la probabilité d'avoir des incohérences dans les AoIs des avatars. Nous évaluons l'incohérence de l'AoI pour chaque avatar tous les $200\ ms$ ou à chaque modification de l'AoI. La Figure B.13 montre la CDF des valeurs d'incohérence mesurées sur les C/S et P2P Second Life. Comme nous avons tracé les valeurs d'incohérence dans l'échelle logarithmique (axe des abscisses de la Fig-

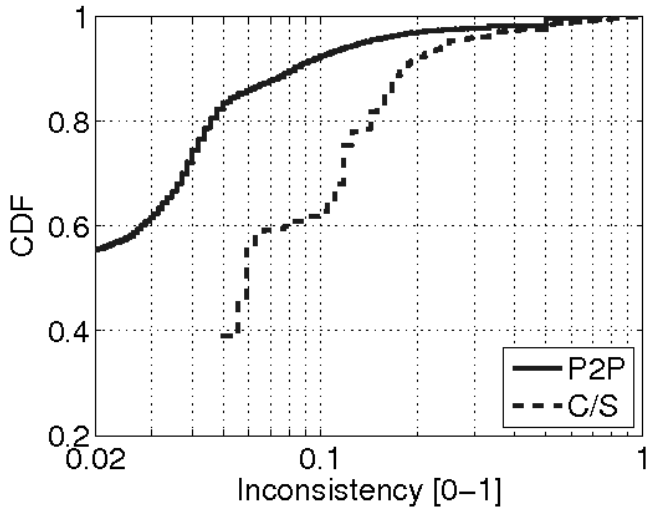


FIG. B.13: CDF de l'incohérence.

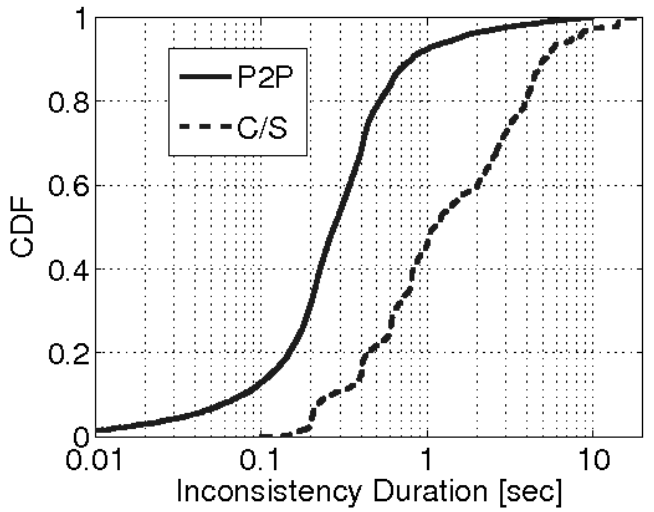


FIG. B.14: CDF de la durée de l'incohérence.

ure B.13), les deux courbes sont tronquées respectivement pour des valeurs d'incohérence égales à 0,02 pour l'architecture P2P et 0,05 pour l'architecture C/S, c.à.d. les plus petites valeurs non-nulles mesurées au cours de nos expériences.

La Figure B.13 montre que l'architecture P2P est toujours plus cohérente que celle du C/S. Les avatars ont une vue parfaite de leur AoIs, c.à.d. une incohérence égale à 0, dans environ 55% des cas, comparativement à 40% des cas en C/S. La distance entre les deux courbes est presque constante pour des valeurs d'incohérence inférieures à 0,2, indiquant que l'architecture P2P produit un gain dans l'expérience de l'utilisateur d'environ 20%. Pour des valeurs d'incohérence superieures à 0,3-0,4 les deux courbes deviennent très proches. Ces valeurs élevées d'incohérence arrivent en présence de churn (opérations de connexion/deconnexion) et des agglomérats

d'avatars. Pendant que le serveur SL souffre de ces événements en raison d'une augmentation de sa charge, le réseau P2P souffre des difficultés à maintenir une triangulation Delaunay consistante.

Nous analysons maintenant la durée de l'incohérence dans le P2P et C/S Second Life afin de comprendre quelle architecture réagit plus rapidement aux incohérence des avatars (Figure B.14). Comme pour les résultats d'incohérence, P2P surpasse nettement l'actuelle architecture C/S de SL. Environ 90% du temps l'incohérence dans le P2P est résolu en moins d'une seconde, soit environ 5 fois plus rapidement que dans le C/S. Ce résultat est très prometteur si on considère que les valeurs tolérables d'interactivité dans les jeux en ligne varient entre $300\ ms$ et $1\ sec$. Inversement, la Figure B.14 dévoile des valeurs inacceptables de durée d'incohérence dans l'actuelle architecture C/S de SL, ex., 40% des incohérences durent plus de 2 secondes.

List of Publications

[1] Matteo Varvello, Ernst Biersack, and Christophe Diot. Dynamic Clustering in Delaunay-Based P2P Networked Virtual Environments. In *NETGAMES'07*, Melbourne, Australia, September 2007.

[2] Matteo Varvello, Fabio Picconi, Christophe Diot, and Ernst Biersack. Is There Life in Second Life ? In *CONEXT'08*, Madrid, Spain, December 2008.

[3] Christoph Neumann, Nicolas Prigent, Matteo Varvello, and Kyoungwon Suh. Challenges in Peer-to-Peer Gaming. *SIGCOMM CCR*, 37(1) :79–82, 2007.

[4] Matteo Varvello, Ernst Biersack, and Christophe Diot. A Networked Virtual Environment over KAD. In *CONEXT'07, Ext. Abstract*, New York, NY, USA, December 2007.

[5] Matteo Varvello, Christophe Diot, and Ernst Biersack. P2P Second Life : experimental validation using Kad. In *INFOCOM'09*, Rio De Janeiro, Brazil, April 2009.

[6] Matteo Varvello, Christophe Diot, and Ernst Biersack. A Walkable Kademlia Network for Virtual Worlds. In *IPTPS'09*, Boston, MA, USA, April 2009.

[7] Matteo Varvello, Christophe Diot, and Ernst Biersack. A Walkable Kademlia Network for Virtual Worlds. In *INFOCOM'09, Ext. Abstract*, Rio De Janeiro, Brazil, April 2009.

[8] Matteo Varvello, Ernst Biersack, and Christophe Diot. Dynamic Clustering in Delaunay-Based P2P Networked Virtual Environments. In *SIGCOMM'07, Ext. Abstract*, Kyoto, Japan, September 2007.

[9] Matteo Varvello, Stefano Ferrari, Ernst Biersack, and Christophe Diot. Distributed Avatar Management for Second Life. In *NETGAMES'09*, Paris, France, November 2009.