# Designing Peer-to-Peer Overlays: a Small-World Perspective

THÈSE N$^{O}$ 4327 (2009)

PRÉSENTÉE LE 6 MARS 2009

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Laboratoire de systèmes répartis

SECTION DES SYSTÈMES DE COMMUNICATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Šarūnas GIRDZIJAUSKAS

acceptée sur proposition du jury:

Prof. B. Faltings, président du jury
Prof. K. Aberer , directeur de thèse
Dr G. Chockler, rapporteur
Prof. R. Guerraoui, rapporteur
Prof. S. Haridi, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2009

# Contents

# Abstract

The Small-World phenomenon, well known under the phrase “six degrees of separation”, has been for a long time under the spotlight of investigation. The fact that our social network is closely-knitted and that any two people are linked by a short chain of acquaintances was confirmed by the experimental psychologist Stanley Milgram in the sixties. However, it was only after the seminal work of Jon Kleinberg in 2000 that it was understood not only why such networks exist, but also why it is possible to efficiently navigate in these networks. This proved to be a highly relevant discovery for peer-to-peer systems, since they share many fundamental similarities with the social networks; in particular the fact that the peer-to-peer routing solely relies on local decisions, without the possibility to invoke global knowledge. In this thesis we show how peer-to-peer system designs that are inspired by Small-World principles can address and solve many important problems, such as balancing the peer load, reducing high maintenance cost, or efficiently disseminating data in large-scale systems. We present three peer-to-peer approaches, namely Oscar, Gravity, and Fuzzynet, whose concepts stem from the design of navigable Small-World networks.

Firstly, we introduce a novel theoretical model for building peer-to-peer systems which supports skewed node distributions and still preserves all desired properties of Kleinberg’s Small-World networks. With such a model we set a reference base for the design of data-oriented peer-to-peer systems which are characterized by non-uniform distribution of keys as well as skewed query or access patterns. Based on this theoretical model we introduce Oscar, an overlay which uses a novel scalable network sampling technique for network construction, for which we provide a rigorous theoretical analysis. The simulations of our system validate the developed theory and evaluate Oscar’s performance under typical conditions encountered in real-life large-scale networked systems, including participant heterogeneity, faults, as well as skewed and dynamic load-distributions.

Furthermore, we show how by utilizing Small-World properties it is possible to reduce the maintenance cost of most structured overlays by discarding a core network connectivity element – the ring invariant. We argue that reliance on the ring structure is a serious impediment for real life deployment and scalability of structured overlays. We propose an overlay called Fuzzynet, which does not rely on the ring invariant, yet has all the functionalities of structured overlays. Fuzzynet takes the idea of lazy overlay maintenance further by eliminating the need for any explicit connectivity and data maintenance operations, relying merely on the actions performed when new Fuzzynet peers join the network. We show that with a sufficient amount of neighbors, even under high churn, data can be retrieved in Fuzzynet with high probability.

Finally, we show how peer-to-peer systems based on the Small-World design and with the capability of supporting non-uniform key distributions can be successfully employed for large-scale data dissemination tasks. We introduce Gravity, a publish/subscribe system capable of building efficient dissemination structures, inducing only minimal dissemination relay overhead. This is achieved through Gravity's property to permit non-uniform peer key distributions which allows the subscribers to be clustered close to each other in the key space where data dissemination is cheap. An extensive experimental study confirms the effectiveness of our system under realistic subscription patterns and shows that Gravity surpasses existing approaches in efficiency by a large margin.

With the peer-to-peer systems presented in this thesis we fill an important gap in the family of structured overlays, bringing into life practical systems, which can play a crucial role in enabling data-oriented applications distributed over wide-area networks.

**Keywords:** Peer-to-Peer Systems, Structured Overlays, Small-World Networks, Non-uniform Key Distributions, Publish/Subscribe Systems.

# Résumé

Le phénomène du Petit-Monde, bien connu sous le nom de “six degrés de séparation”, était l’objet de recherches pendant de longues années. Le fait que notre réseau social est étroitement ficelé tel que deux individus sont liés par une courte chaîne de connaissances a été confirmé par le psychologue expérimental Stanley Milgram dans les années soixante. Toutefois, c’est le travail séminal de Jon Kleinberg en 2000 qui nous a aidés à comprendre non seulement la raison de l’existence de ces réseaux, mais aussi pourquoi il est possible de naviguer efficacement dans ces réseaux. Cette découverte a été très pertinente pour les systèmes pair-à-pair, comme ces derniers ont des similitudes fondamentales avec les réseaux sociaux, surtout le fait que le routage pair-à-pair se base uniquement sur des décisions locales, sans la possibilité d’utiliser un aperçu global du réseau. Cette thèse a pour but de démontrer comment les conceptions de systèmes pair-à-pair inspirées par les principes Petit-Monde peuvent résoudre plusieurs problèmes importants, tel que l’équilibrage de la charge des pairs, la réduction du coût élevé de l’entretien, ou la diffusion efficace des données dans des systèmes à grande échelle. Nous proposons trois mécanismes pair-à-pair – Oscar, Gravity, et Fuzzynet – dont les concepts proviennent de la conception de réseaux Petit-Monde navigables.

Tout d’abord, nous introduisons un nouveau modèle théorique pour la construction de systèmes pair-à-pair qui prend en considération les distributions inégales de nœuds tout en gardant toutes les propriétés désirées des réseaux Petit-Monde de Kleinberg. Avec un tel modèle nous avons établi une base de référence pour la conception de systèmes pair-à-pair axés sur les données et qui sont distingués par une distribution de clés non-uniforme, ainsi que des modèles de requête et d’accès non-uniformes. Sur la base de ce modèle théorique nous introduisons Oscar, un overlay qui utilise une nouvelle technique évolutive d’échantillonnage pour la construction de réseaux; nous détaillons une analyse théorique rigoureuse d’Oscar. Les simulations de notre système valident la théorie développée et évaluent la performance d’Oscar sous des conditions typiques présentes dans des réels réseaux à grande échelle, y compris l’hétérogénéité des participants, les défauts, ainsi que les distributions de charge non-uniformes et dynamiques.

En outre, nous démontrons comment l’utilisation des propriétés Petit-Monde permet de réduire le coût de l’entretien de la plupart des overlays structurés en se débarrassant d’un élément principal de connectivité – l’invariant de l’anneau. Nous soutenons que la dépendance de la structure de l'anneau est un obstacle important au déploiement actuel et à l'évolutivité des overlays structurés. Nous proposons Fuzzynet, un overlay qui ne dépend pas de l'invariant de l'anneau et qui conserve néanmoins toutes les fonctionnalités des overlays structurés. Fuzzynet pousse l'idée de l'entretien paresseux des overlays encore plus loin en éliminant la nécessité d'opérations explicites de connectivité et d'entretien des données, en s'appuyant uniquement sur les actions effectuées quand de nouveaux pairs Fuzzynet rejoignent le réseau. Nous démontrons qu'avec un nombre suffisant de voisins, même avec un taux de rotation élevé, on peut récupérer les données dans Fuzzynet avec une grande probabilité.

Enfin, nous démontrons comment les systèmes pair-à-pair conçus avec les principes Petit-Monde et capables de soutenir des distributions non-uniformes de clés peuvent être utilisés avec succès pour la diffusion de données à grande échelle. Nous introduisons Gravity, un système de publication/souscription capable de construire des structures de diffusion efficaces, induisant un temps inactif minimal pour le relais de la diffusion. Ceci est possible parce que Gravity soutient des distributions non-uniformes de clés, ce qui permet aux souscripteurs de se regrouper les uns près des autres dans l'espace des clés où la diffusion des données n'est pas coûteuse. Une étude expérimentale approfondie confirme l'efficacité de notre système dans un modèle de souscriptions réaliste et montre que Gravity dépasse les techniques existantes par une grande marge en matière d'efficacité.

Avec les systèmes pair-à-pair présentés dans cette thèse, nous comblons une lacune importante dans la catégorie des overlays structurés, en rendant possibles des systèmes pratiques qui pourraient jouer un rôle crucial dans la réalisation d'applications axées sur les données et distribuées sur des réseaux de grande taille.

**Mots clefs:** Systèmes Pair-à-Pair, Overlays Structurés, Réseaux Petit-Monde, Distributions Non-uniformes de Clés, Systèmes de Publication/Souscription.

# Acknowledgements

There are several people without whom this thesis would not have been possible, and whom I would like to thank. First of all, I wish to thank my advisor, Prof. Karl Aberer, for giving me the opportunity of pursuing the thesis under his supervision, and for all his guidance and constructive feedback. I am also very thankful to the members of my thesis committee, Dr. Gregory Chockler, Prof. Rachid Guerraoui and Prof. Seif Haridi, as well as the president of the committee Prof. Boi Faltings, for the time spent on reviewing this manuscript, and for their valuable comments on improving its content.

I would like to thank all my colleagues I collaborated with while working on this thesis, especially Anwitaman Datta, Wojciech Galuba, Manfred Hauswirth, Vasilios Darlagiannis, Fabius Klemm and many others. I am also thankful to the anonymous reviewers of my papers, who provided me with useful comments and helped to improve the quality of the papers, and so, indirectly, the quality of this thesis. I want to thank all my current and former colleagues from LSIR, who created a wonderful working environment in the lab. Special thanks goes to Chantal for her unparalleled logistical support.

I owe many thanks to my colleagues from IBM Research in Haifa, where I did my internship, for the interesting, fruitful and exciting three months. I want to extend my thanks to Gregory Chockler, who made my internship possible. Thanks to Alexey, Vita, Eddie, Gena and many others, who made my stay in Haifa a great fun.

I thank all my friends whom I met in Switzerland, for making the life here a wonderful experience: Marta & Maciej, Mallory & Maxim, Irina & Valya, Natasha & Denis, Dan, Wojtek, Adriana, Parisa, Ivana, Marcin, Simas, Audrius, Daiva, Iuli, Oana, Razvan, Olga, Alexei, Phil, Roman, Sebastian, Fabius and many many others. I would especially like to thank my best friend here and a great officemate - Gleb, and my wonderful flatmates Kasia & Michal for tolerating me and giving me a great deal of support.

Most importantly, I wish to express a special gratitude to my parents Violeta and Stasys, my sister Rūta and my grandma Ermina for their continuous and unconditional love and support, which – despite being separated miles-away – I always felt here in Switzerland. Finally, my biggest thanks goes to my main driving force here - my girlfriend Ivana, for her love, support and encouragement which I received during all my studies.

# Chapter 1

# Introduction

There is no reason anyone would want a computer in their home.

KEN OLSEN, president, chairman and founder of Digital Equipment Corp., 1977

The world has changed dramatically since the famous words of Ken Olsen in 1977. During the last three decades computers irreversibly conquered our homes. Their ever growing numbers and extensive connectivity sparkled the growth of the Internet, which rapidly shifted from a military-oriented application to an integral part of our daily life. The modern world became flooded with countless interconnected electronic devices – laptops, servers, handhelds, mobile phones, etc., which are capable of processing, transmitting and storing data. It is estimated that there already exist over one billion computers in the world and by 2015 this number is likely to double. The total traffic among these devices reaches 8 terabytes per second and their combined storage space is estimated to hit a staggering size of 255 exabytes[1]. Thus, the effective management and utilization of such sheer numbers of available resources became one of the main challenges of the XXI$^{st}$ century.

Such vast numbers of interconnected devices became a fertile environment for new kind of computing substrates in a digital world, such as peer-to-peer systems. The intrinsic idea behind these systems is to utilize otherwise idle resources dispersed all over the world in a *"for the end-users from the end-users"* fashion. With the appearance of peer-to-peer systems many novel applications emerged, like large scale distributed computing (e.g., BOINC[2]), file-sharing (e.g., BitTorrent[3]), or peer-to-peer telephony (e.g., Skype[4]). According to a recent study, peer-to-peer systems are the biggest consumer of bandwidth in the current Internet, responsible for between 49 and 84 percent of the overall Internet traffic, which can even be as high as 95% at

[1] http://www.kk.org/thetechnium/
[2] http://boinc.berkeley.edu/
[3] http://www.bittorrent.com/
[4] http://www.skype.com/

nightime[5].

However, the design of such decentralized systems poses many challenges. The designers of peer-to-peer systems have to cope with millions of autonomous resources which operate in highly dynamic and volatile environments with almost no reliability guarantees. The essential requirement for most of these systems is to provide an efficient access to any resource in the system. In other words, any peer-to-peer participant should have the ability to find (i.e., navigate to) any other participant or resource without global knowledge, central control, or dedicated infrastructure. Although, as we will see in the upcoming chapters, there were many ways devised to navigate in such decentralized systems, the most efficient navigation patterns resemble those in human social networks, which by their nature are distributed and without central control. In the following, we will discuss in more detail the concept of efficient navigation in social graphs, or *Small-World networks*, and the consecutive impact on the design of peer-to-peer systems.

## 1.1. Small-World Inspired Design

Probably every person once in a while finds himself in a situation where the phrase *"Oh, what a small world!"* perfectly describes the astonishment over the discovery of sharing a common friend or a friend-of-a-friend with a complete stranger. This turns even more bizarre if that stranger is from another country or from a completely different part of the world. The awareness of such a fascinating phenomenon is likely to have existed for centuries, but its scientific investigation started only with the research of the experimental psychologist Stanley Milgram in the sixties. Milgram devised an experiment with which he wanted to check whether our social world is indeed small, i.e., that even complete strangers will be linked through friends of friends in just a few hops. The experiment had very simple rules. He randomly picked a person from the American mid-west that had to forward a mail-package to a target person of Milgram's choice. However, the recipients of the package could not send it directly to the target but only to somebody they actually knew personally. The results were intriguing if not surprising. Milgram counted that the average length of the number of steps required for a package to reach the destination was staggeringly small – only around six forwarding hops. Thus, it was a clear confirmation that the world is indeed small. Since then, a lot of research effort has been dedicated for understanding and modeling the Small-World phenomenon.

The first mathematical explanations were attributed to the short diameter of the graph of the social network. Random graphs, however, failed to model social networks, since it was known that despite the similarities in diameter, social networks had much higher clusterization (i.e., one's two friends are likely to be friends themselves) than random graphs. Watts and Strogatz [Watts and Strogatz 1998] came up with a network model which could generate net-

[5] http://www.ipoque.com/news-and-events/news/ipoque-internet-study-2007-p2p-file-sharing-still-dominates-the-worldwide-internet.html

works which had both - high clusterization and low diameter. They proposed to start with a graph which shows “heavy clustering” (e.g., nearest-neighbor graph) and then to re-wire some edges by changing one end point to a node picked uniformly at random.

Yet, this model did not completely answer all the questions related to the phenomenon of Small-Worlds and Milgram’s experiment. It was not unexpected that social graphs had a short diameter, i.e., the fact that there *exists* a short acquaintance path between any two persons. However, it was still unclear why completely decentralized and local (packet forwarding) decisions could *find* that shortest path by navigating on the underlying social graph. This was answered by the seminal work of Kleinberg [Kleinberg 2000] where he showed that efficient navigation based only on local knowledge is not possible on a graph which is generated by uniform random rewiring or the addition of uniformly distributed shortcuts. However, Kleinberg proved that there exists a family of Small-World graphs where a decentralized search algorithm is optimal and cannot perform better. To be able to model such graphs, Kleinberg had to introduce a distance notion in the model. This family of graphs had a specific property as the long-range edges were not established uniformly randomly, but the probability that any two nodes are connected by a long-range link depended on the distance between them. More specifically, the nodes of Kleinberg’s “routing efficient” Small-World graph were populated on a $r$-dimensional lattice where a distance function $d(u, v)$ could be defined between any two nodes $u$ and $v$ (i.e., the graph was embedded in a $r$-dimensional metric space). In the model every node was assigned to have a small (constant) number of short-range links to its immediate neighbors and at least one long-range link. Kleinberg proved, that a decentralized greedy routing algorithm performs the best, i.e., the expected length of a search path is polylogarithmic in the size of the graph, when the probability of long-range link establishment between two nodes $u$ and $v$ is proportional to $d(u, v)^{-r}$, where $r$ is dimensionality of the space. In such a way, graphs constructed according to Kleinberg’s Small-World model not only have a small diameter and exhibit high clustering, but are also navigable. This network construction model is extremely relevant to the field of peer-to-peer, since many peer-to-peer systems have an underlying metric space and for navigation rely only on local decisions, usually without having any global knowledge. Indeed, as it was observed later, the topologies of numerous existing peer-to-peer systems resemble *Kleinbergian* Small-World graphs and exhibit many of their properties. Thus, the unveiling of this “routing efficient” Small-World design finally shed some light on understanding how social networks operate and set a base for the designers of peer-to-peer systems.

The systems developed within the scope of this thesis, namely *Oscar*, *Gravity* and *Fuzzynet*, share a common flavor – they are all based on a “routing efficient” Small-World design. As it will be shown in the upcoming chapters, Small-World networks are unique in their nature, since on the one hand they provide a structure which is proven to be easy to navigate, while on the other hand, they provide ample randomization in the system which allows Small-World based peer-to-peer systems to be flexible enough for adapting to various peer-to-peer settings with heterogeneous resources or to harsh high-churn peer-to-peer environments. This thesis is a display of techniques showing how to deal with several important problems in the field of peer-to-peer systems with the Small-World network model.

Throughout this thesis we demonstrate how Small-World networks can be effectively employed to solve problems of peer-to-peer system design such as balancing the load among heterogeneous peers, reducing high maintenance cost of peer-to-peer topologies, or efficiently disseminating data in large-scale systems. In the following sections we will introduce these problems in more detail.

## 1.2. Motivation

### 1.2.1. Towards Structured Peer-to-Peer Systems

Although the term “peer-to-peer” was coined relatively recently, the concept itself is a much older one. Already the rise of the Internet brought the first instances of peer-to-peer architectures like the Domain Name System (DNS), the Simple Mail Transfer Protocol (SMTP) and USENET. These architectures were intrinsically decentralized and represented the symmetric nature of the Internet, where every node in the system had equal status and assumed cooperative behavior of the other nodes. The beginning of the file-sharing era and the rise and fall of the first file-sharing peer-to-peer system Napster[6] (2000-2001) paved the way for the second generation of peer-to-peer overlays like Gnutella[7] (2000) and Freenet [Clarke *et al.* 2001]. Their simple protocols and unstructured nature made these networks robust and avoided Napster’s drawbacks like having a single-point-of-failure. Since 2001, these peer-to-peer overlays became extremely popular and accounted for the majority of the Internet traffic. However, the design of unstructured networks had an intrinsic weakness because of their reliance on very costly network flooding. Restricted flooding was not a good remedy either because of the unreliable resource discovery. To improve this, Gnutella introduced super-peer based hierarchical networks which alleviated the problem to a certain extent. However, a fundamental shift in the design of peer-to-peer systems occurred with the introduction of *structured overlays* which use the existing resources more effectively.

Structured overlay networks use more efficient routing techniques and their topology is not arbitrary. The link establishment among the peers is usually strictly defined by the specific protocols. The topologies can result in various structures like rings, toruses, hypercubes, de-Bruijn networks, etc. A particular instance of structured peer-to-peer overlays are Distributed Hash Tables (DHTs) enabling an efficient lookup service, by using a predefined hashing algorithm to assign virtual ownership for a particular resource (e.g., P-Grid [Aberer 2001], Chord [Stoica *et al.* 2001], DKS [Alima *et al.* 2003], Symphony [Manku *et al.* 2003]). In contrast to the traditional Hash Table, DHTs share the global hash function among all the participating peers

[6] www.napster.net
[7] http://www.gnutellaforums.com/

and the DHT protocols ensure that any part of a global hash table is easily reachable (usually in a logarithmic number of steps). In addition, replication is used to sustain the persistence and data availability in the system.

### 1.2.2. DHTs and Data-Oriented Applications

Many popular peer-to-peer systems are designed for data-oriented (or data-sharing) applications (e.g., Gridella[8], KaZaA[9]). These applications enable transparent access to internet-scale distributed databases without resorting to a centralized index. Such systems are usually built on top of structured overlays (e.g., DHTs) and/or hierarchical super-peer networks, where super-peers are often organized in a structured manner (e.g., FastTrack[10]). However, as we will see next, most of the existing DHTs can seriously limit the querying potential of data-oriented peer-to-peer systems.

Although a vast majority of structured peer-to-peer systems are conceptually similar, they differ in the rules which describe how to choose neighboring links at each peer to form routing tables. These rules heavily depend on the nature of the peer identifers (keys), which are acquired by using specific hash functions. To ease the network construction task and to effectively balance data load, most of the structured peer-to-peer systems use *uniform* hash functions (like SHA-1) for assigning identifers to peers and resources (e.g., shared data). The usage of uniform hash functions ensures that peers in the system are unlikely to be overloaded and enables the utilization of simple and unambiguous neighbor selection protocols which guarantee a balanced node degree. The use of uniform hash functions, however, limits the capabilities of data-oriented overlays to simple identifer lookup. More complex queries like, e.g., range queries, become extensively ineffective since uniform hash functions disperse otherwise correlated data over many different peers, thus making the access highly inefficient if not unscalable. Therefore, semantic data processing cannot be successfully tackled by conventional, uniform hash function based peer-to-peer systems.

Furthermore, practical scalable peer-to-peer systems need to take into account heterogeneity explicitly in the system's design. For data-oriented overlays, heterogeneity occurs due to the both the peculiarities of the environment and the application characteristics. Measurement studies [Stutzbach *et al.* 2005] of deployed peer-to-peer systems show heterogeneity arising because of either diverse availability of resources like storage, bandwidth, computation, and content at peers, or variation in individual willingness to contribute resources to the system, as well as software artifacts like default configurations. Thus, uniform hash functions are rendered to be ineffective facing the consequences of heterogeneous environments since under such circumstances data-oriented applications are inevitably characterized by non-uniform distribution of keys over the key-space as well as skewed query or access patterns.

[8] http://www.p-grid.org/implementation/help.html
[9] http://www.kazaa.com/
[10] http://www.fasttrack.nu/

This implies that overlay networks have to be able to handle hash functions which produce non-uniform key distributions (e.g., order-preserving hashing). The design of such overlay networks has to take the resulting skewed key distributions into account and adapt the construction mechanisms accordingly, which is not a straightforward task. However, most of the existing peer-to-peer approaches avoid addressing these issues, instead taking advantage of uniformity assumptions on peers' capacity in terms of bandwidth consumption and storage capacities, which might limit the practicality for realistic peer-to-peer environments. Thus, designing efficient overlay networks for data-oriented applications remains a challenge in peer-to-peer systems, which we address in this thesis.

### 1.2.3. Ring Maintenance in Peer-to-Peer Systems

One of the main obstacles to further adoption of structured peer-to-peer systems is mainly related to their sophisticated and costly maintenance. Many structured overlay networks rely on a ring invariant as the core network connectivity element. The responsibility ranges of the participating peers and the navigability principles (greedy routing) heavily depend on the ring structure. For correctness guarantees, each node needs to eagerly maintain its immediate neighboring links - the ring invariant. Hence, historically, most existing structured overlays have de facto considered it necessary [Alima *et al.* 2003; Manku *et al.* 2003; Stoica *et al.* 2001]. However, the ring maintenance is an expensive task and it may not even be possible to maintain the ring invariant continuously under high churn, particularly as the network size grows [Freedman *et al.* 2005].

Unfortunately, the environments in which peer-to-peer systems usually operate are highly unreliable. Various routing anomalies in the network, peers behind firewalls and the presence of Network Address Translators (NATs) create non-transitivity effects which might disrupt the communication among the network participants. It is quite common in real-life networks that some pairs of alive peers cannot directly communicate to each other (e.g., between two firewalled peers); however, it is possible for them to communicate indirectly through a third peer. As it has been shown in [Freedman *et al.* 2005], such non-transitive connectivity may misdirect nodes to wrongly set their ring neighbors, thus leading to a violation of the ring invariant and disrupting the overlay's functional correctness.

In this thesis we argue that the ring is not only unnecessary, but also relying on such a ring invariant leads to some undesirable consequences. In certain cases, the existing greedy-routing mechanisms cannot deal with even a single fault/break in the ring on the routing path. On the other hand, in a dynamic environment where the peer lifetime is a few minutes for the majority of them, the ring is susceptible to continuous breakages. This in turn incurs high maintenance cost, and despite whatever high maintenance effort, there is at no point any absolute guarantee that the ring is indeed intact. The larger the number of peers, the more likely it is that the ring invariant is violated. Thus, the reliance on the ring structure is a serious impediment for real life deployment and scalability of structured overlays.

In this thesis we deal with this problem by introducing a novel overlay design technique, which is based on Small-World construction principles and does not rely on the ring invariant, yet has all the functionalities of structured overlays.

### 1.2.4. Efficiency in Publish/Subscribe Systems

Another promising area of employing data-oriented peer-to-peer networks nowadays is related to publish/subscribe systems [Castro *et al.* 2002; Eugster *et al.* 2003]. Publish/subscribe is a popular communication middleware that allows users to subscribe to topics of interest, and then be notified of posted messages related to any of the topics in their subscriptions. Traditionally, most uses of publish/subscribe have been limited to building applications integrating multiple (possibly diverse) data sources, such as event processing engines, live event broadcast, RSS feed readers, etc. Recently, there has been a growing interest in applying publish/subscribe to support communication in an emerging class of applications involving fine-grained information sharing on massive scales [Ostrowski *et al.* 2007, 2008], such as, e.g., on-line gaming, Internet chat rooms, Second Life, etc.

In order to adequately address the scaling needs of these applications, a publish/subscribe solution must be able to effectively deal with large populations of dynamic users, large numbers of topics, and arbitrary subscription patterns. However, many traditional solutions concentrate the message processing load in a few fixed system components (such as centralized servers, or fixed hierarchies thereof), and therefore, do not scale well as the system grows in size. Thus, publish/subscribe approaches became very tempting targets for applying peer-to-peer techniques. Several decentralized publish/subscribe implementations have been proposed (e.g., [Bhola *et al.* 2002; Castro *et al.* 2002; Voulgaris *et al.* 2006]), where the nodes are typically organized in a peer-to-peer network, whose links are then used to propagate published data.

Some of the proposed overlay-based implementations are quite effective in dealing with scaling issues (such as the number of nodes and geographical spread) mainly because of the sub-linear degree and low diameter properties of their underlying communication graphs. However, due to possible lack of connectivity among the nodes sharing the same interests, the message dissemination overhead could potentially be quite high. The existing peer-to-peer based publish/subscribe protocols are typically oblivious to the actual node subscriptions, i.e., do not exploit correlations among peer-subscriptions. As a result, a message published on a certain topic needs to traverse a large number of uninterested peers before reaching all of its subscribers, thus resulting in a high message dissemination cost.

In this thesis, we therefore address this issue by proposing a novel Small-World based publish/subscribe system that exploits similarity in the individual node subscriptions.

## 1.3. Outline and Contributions

In this thesis we introduce a theoretical framework to design structured peer-to-peer overlays which support non-uniform hash functions. Based on our theoretical findings, we propose a novel overlay network, called *Oscar*, which not only is capable of dealing with key distributions of any complexity but also takes advantage of the heterogeneity of peer environments. Our approach is based on scalable sampling techniques and the design of peer-to-peer overlays which are based on the randomized networks exhibiting Small-World properties like high clusterization and small diameter.

Furthermore, we introduce another Small-World based peer-to-peer design, *Fuzzynet*, which allows to reduce the maintenance cost of structured peer-to-peer systems by eliminating the maintenance intensive structure of the ring. Fuzzynet takes the idea of lazy overlay maintenance further by dropping any explicit connectivity and data maintenance requirement, relying merely on the actions performed when new Fuzzynet peers join the network. We show that with sufficient amount of neighbors ($O(\log N)$, comparable to traditional structured overlays), even under high churn, data can be retrieved in Fuzzynet with high probability.

For efficient handling of publish/subscribe systems we propose the *Gravity* technique, which significantly reduces message dissemination cost in the system as compared to the existing approaches. Gravity is an Oscar-based publish/subscribe system that exploits similarity in the individual node subscriptions to build efficient dissemination structures while retaining fixed node degrees. Gravity's key mechanism is to dynamically cluster the nodes with similar subscriptions by placing them close to each other on the unit ring, thus resulting in a network where nodes with similar interests are closely connected. The messages are then disseminated over multicast trees that preserve peer's proximity in the network resulting in a low publication cost. Since such a design results in highly skewed peer-key distributions, Gravity exploits the advantages of Oscar overlay's abilities to efficiently handle skewed key distributions of any complexity.

In the following we provide a short summary of the main thesis chapters:

**Part I: Fundamentals**

In ***Chapter 2: Peer-to-Peer Systems***, we describe the most important concepts related to peer-to-peer systems and their design. We present a complete taxonomy of peer-to-peer systems and familiarize the reader with the most prominent peer-to-peer approaches in each category. We also discuss different routing and maintenance strategies.

In ***Chapter 3: Reference Architecture***, we propose a reference model for overlay networks which is capable of modeling different approaches in the domain of peer-to-peer systems in a generic manner. It is intended to allow peer-to-peer designers to assess the properties of concrete systems, to establish a common vocabulary, to facilitate the qualitative comparison of the systems, and to serve as the basis for defining a standardized API to make overlay networks interoperable.

**Part II: Small-World Inspired Overlays**

In ***Chapter 4: On Small World Graphs in Non-uniformly Distributed Key Spaces***, we show that the topologies of most logarithmic-style peer-to-peer systems like Pastry [Rowstron and Druschel 2001], Tapestry [Zhao *et al.* 2004] or P-Grid [Aberer 2001] resemble Small-World graphs. Inspired by Kleinberg's Small-World model [Kleinberg 2000] we extend the model of building "routing-efficient" Small-World graphs and propose two new models. We show that the graph, constructed according to our model for uniform key distribution and logarithmic out-degree, will have similar properties as the topologies of structured peer-to-peer systems with logarithmic out-degree. Moreover, we propose a novel theoretical model of building graphs which supports uneven node distributions and preserves all desired properties of Kleinberg's Small-World model. With such a model we are setting a reference base for nowadays emerging peer-to-peer systems that need to support uneven key distributions.

Based on the theoretical model of Chapter 4 we present an overlay network, called Oscar, in ***Chapter 5: Oscar: Structured Overlay For Heterogeneous Environments***. Oscar simultaneously deals with heterogeneity as observed in the Internet (capacity of computers, bandwidth), as well as non-uniformity observed in data-oriented applications, and can deal with peer key-distributions of any complexity. We demonstrate through simulations that our technique performs well and significantly surpasses similar techniques like Mercury [Bharambe *et al.* 2004] for realistic workloads.

In ***Chapter 6: Fuzzynet: Ringless Routing in a Ring-like Structured Overlay*** , we propose an approach called Fuzzynet, which circumvents the need for a ring and the associated problems like non-transitivity and costly maintenance. By introducing the Fuzzynet technique, we set a base for a completely lazy-maintenance design of peer-to-peer systems where the only maintenance action is taken upon peers joining the network. Fuzzynet is based on the connectivity principles of navigable Small-World networks [Kleinberg 2000]. It does not require a ring structure, yet it has all the functionalities of contemporary structured overlay networks. We validate our novel design principles by simulations, as well as PlanetLab experiments and compare them with ring based overlays.

In ***Chapter 7: Gravity: An Interest-Aware Publish/Subscribe System***, we consider the problem of efficient data dissemination in a decentralized publish/subscribe systems in the presence of large numbers of topics and arbitrary subscription patterns. We introduce Gravity, our publish/subscribe system based on Oscar overlay that achieves communication efficiency while preserving fixed node degree by biasing the link creation process so that the nodes sharing similar interests are more likely to be closely connected (i.e., clustered). A thorough experimental study confirms the effectiveness of our system given realistic subscription patterns and shows that Gravity surpasses existing approaches in efficiency by a large margin.

## 1.4. Selected Publications

This thesis is based on the following publications:

- Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *On Small World Graphs in Non-uniformly Distributed Key Spaces.* The 1st IEEE International Workshop on Networking Meets Databases, April 8-9 2005 Tokyo, Japan.
- Karl Aberer, Luc Onana Alima, Ali Ghodsi, Sarunas Girdzijauskas, Manfred Hauswirth, Seif Haridi: *The Essence of P2P: A Reference Architecture for Overlay Networks.* The 5th IEEE International Conference on Peer-to-Peer Computing, August 31-September 2 2005, Konstanz, Germany.
- Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *Oscar: Small-world Overlay for Realistic Key Distributions.* The Fourth International Workshop on Databases, Information Systems and Peer-to-Peer Computing, September 11, 2006, Seoul, Korea.
- Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *Oscar: A Data-Oriented Overlay For Heterogeneous Environments.* The 23rd International Conference on Data Engineering, April 16-20, 2007, Istanbul, Turkey.
- Sarunas Girdzijauskas, Gregory Chockler, Roie Melamed, Yoav Tock: *Gravity: An Interest-Aware Publish/Subscribe System Based on Structured Overlays.* The 2nd International Conference on Distributed Event-Based Systems, July 1-4, 2008, Rome, Italy.
- Sarunas Girdzijauskas, Wojciech Galuba, Vasilios Darlagiannis, Anwitaman Datta, Karl Aberer: *Fuzzynet: Ringless Routing in a Ring-like Structured Overlay* To be published in Peer-to-Peer Networking and Applications Journal, Springer.
- Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *Structured Overlay For Heterogeneous Environments: Design and Evaluation of Oscar.* To be published in ACM Transactions on Autonomous and Adaptive Systems.
- Wojciech Galuba, Sarunas Girdzijauskas: *Peer to Peer Overlay Networks: Structure, Routing and Maintenance* Entry in Springer's upcoming Encyclopedia of Database Systems.

# Part I

# Fundamentals

# Chapter 2

# Peer-to-Peer Systems

For animals, the entire universe has been neatly divided into things to (a) mate with, (b) eat, (c) run away from, and (d) rocks.

Terry Pratchett, Equal Rites

## 2.1. Taxonomy of Peer-to-Peer Overlay Networks

There are many features of peer-to-peer overlays by which they can be characterized and classified [Androutsellis-Theotokis and Spinellis 2004; Risson and Moors 2006]. However, strict classification is not easy since many features have mutual dependencies on each other making it difficult to identify the distinct overlay characteristics (e.g., overlay topologies vs. routing in overlays). Although every peer-to-peer overlay can differ by many parameters, each of them will have to have a certain network structure with distinctive routing and maintenance algorithms allowing the peer-to-peer application to achieve its purpose. Thus, most commonly, peer-to-peer overlays can be classified by:

- Purpose of use;
- Overlay Structure;
- Employed routing mechanisms;
- Maintenance strategies.

### 2.1.1. Purpose of Use

Peer-to-peer overlays are used for an efficient and scalable sharing of individual peers' resources among the participating peers. Depending on the type of the resources which are shared, the peer-to-peer overlays can be identified as being oriented for:

- Data-sharing (data storage and retrieval);
- Bandwidth-sharing (streaming);
- CPU-sharing (distributed computing).

*Data-sharing* peer-to-peer overlays can be further categorized by their purpose to perform one or more specific tasks like file-sharing (by-far the most common use of the peer-to-peer overlays), information retrieval (peer-to-peer web search), publish/subscribe services and semantic web applications. Data-sharing peer-to-peer overlays arguably belong to the most popular category in the peer-to-peer family. Examples of such data-sharing networks include systems like BitTorrent[1], DKS [Alima *et al.* 2003], P-Grid [Aberer 2001], publish/subscribe systems such as Pastry-based Scribe [Castro *et al.* 2002], peer-to-peer information retrieval systems such as YaCy-Peer[2], AlvisP2P[3] [Podnar *et al.* 2007; Skobeltsyn *et al.* 2007], etc.

*Bandwidth-sharing* peer-to-peer overlays are to some extent similar to data-sharing overlay networks, however, they mainly aim at the efficient streaming of real-time data over the network. The efficient overlay design provides the ability to find several disjoint paths from source to destination and therefore, to significantly boost the performance of the data streaming applications. Bandwidth-sharing peer-to-peer overlays are mostly found in peer-to-peer telephony, peer-to-peer video/TV, sensor networks and peer-to-peer publish/subscribe services. For example, currently Skype[4] is arguably the most prominent peer-to-peer streaming overlay application.

For the computationally intensive tasks, when the CPU resources of a single peer cannot fulfill its needs, a *CPU-sharing* peer-to-peer network can provide plenty of CPU resources from the participating idle overlay peers. Currently, only computationally demanding scientific experiments employ such a strategy for tasks like simulation of protein folding or analysis of astronomic radio signals. Although *not* being a pure peer-to-peer overlay, Berkeley Open Infrastructure for Network Computing (BOINC) is very popular among such networks, supporting several distributed computing projects such as SETI@home, folding@home, AFRICA@home, etc.

In this thesis we will focus on data-sharing overlays like Oscar (Chapter 5), Fuzzynet (Chapter 6), and Gravity (Chapter 7), the latter belonging to two categories: data-sharing and bandwidth-sharing.

### 2.1.2. Overlay Structure and Design

Peer-to-peer overlays significantly differ in the topology of the networks they form. There exists a wide scope of possible overlay instances, ranging from centralized to purely decentralized ones,

[1] http://www.bittorrent.com/
[2] http://www.yacyweb.de/
[3] http://globalcomputing.epfl.ch/alvis/
[4] http://www.skype.com/

however, most commonly, three classes of network topology are identified:

- Centralized Overlays;
- Decentralized Overlays;
- Hybrid Overlays.

Depending on the routing techniques and whether the resource distribution affects link establishment methods, overlay networks can be also classified into *structured* and *unstructured* peer-to-peer overlays.

#### 2.1.2.1. Centralized Overlays

Peer-to-peer overlays based on *centralized* topologies are pretty efficient since the interaction between peers is facilitated by a central server which stores the global index, deals with the updates in the system, distributes tasks among the peers or quickly responds to the queries and gives complete answers to them (Figure 2.1(a)). However, not all applications fit the centralized network overlay model. Centralized overlays usually fail to scale with an increasing number of participating peers. The centralized component rapidly becomes the performance bottleneck. The existence of a single-point-of-failure (e.g., Napster[5]) also prevents many potential data-sharing applications from using centralized overlays.

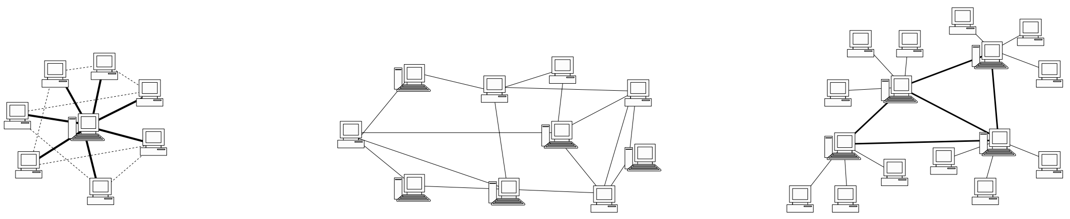
#### 2.1.2.2. Decentralized Overlays

Because of the aforementioned drawbacks, *decentralized* structured and unstructured overlays emerged, which use a purely decentralized network model, and do not differentiate peers into servers or clients, but treat all of them equally - as they were both servers and clients at the same time (Figure 2.1(b)). Thus, such peer-to-peer overlays successfully deal with the scalability requirement and can operate without any central authority.

The simplest decentralized overlays are usually *unstructured*. Unstructured overlay networks use flooding-based routing and the distribution of the resources among the peers is usually unrelated to the network topology.

The most prominent unstructured peer-to-peer system is Gnutella[6]. Nevertheless, many other unstructured peer-to-peer systems have been developed. These systems vary broadly with respect to the underlying topology, the routing techniques, as well as replication and maintenance cost. E.g., SWAN [Liu *et al.* 2006] is operating over an underlying Small-World network, whereas Yappers [Ganesan *et al.* 2003] provides a peer-to-peer look-up service over an arbitrary topology. Another unstructured peer-to-peer technique, called Gia [Chawathe *et al.* 2003] combines biased random walks with one-hop data replication, BubbleStorm [Terpstra *et al.* 2007] provides a probabilistic data retrieval over a peer-to-peer network.

[5] http://www.napster.com/
[6] http://www.gnutellaforums.com/



(a) Centralized Overlay. Central peer facilitates the interactions among the leaf-peers.

(b) Decentralized Overlay. No central authority, all peers treated equally.

(c) Hybrid overlay. Hierarchical topology, interconnected super-peers locally serve the subsets of leaf-peers.

**Figure 2.1.** Examples of Peer-to-Peer Overlays

Because of their simplicity, unstructured overlays are pretty robust to network and peer failures, although they are inefficient due to high bandwidth consumption during querying.

#### 2.1.2.3. Structured Overlay Networks

*Structured* overlay networks use more efficient routing techniques and the topology of the structured overlays is not arbitrary but typically exhibit Small-World properties, specifically high clusterization and low network diameter. The link establishment among the peers is strictly defined by specific protocols. The topologies can result in various structures like rings (Chord [Stoica *et al.* 2001], SkipNets [Harvey *et al.* March 2003], Hieras [Xu *et al.* 2003]) , toruses (CAN [Ratnasamy *et al.* 2001]), hypercubes (HyperCuP [Schlosser *et al.* 2002]), butterfly networks (Viceroy [Malkhi *et al.* 2002]), de-Bruin networks (Koorde [Kaashoek and Karger 2003]) or more loose randomized networks, which do have properties of Small-World networks (e.g., Freenet [Clarke *et al.* 2001], P-Grid [Aberer 2001], Symphony [Manku *et al.* 2003]). A particular instance of structured peer-to-peer overlays is a Distributed Hash Table (DHT) enabling an efficient lookup service, by using a predefined hashing algorithms to assign an ownership for a particular resource (e.g., Chord [Stoica *et al.* 2001], Kademlia [Maymounkov and Mazières 2002] etc.). In contrast to the traditional Hash Table, the DHTs share the global hashing information among all the participating peers and the DHT protocols ensure that any part of a global hash table is easily reachable (usually in logarithmic steps) and there is enough replication to sustain the consistency in the system.

Structured peer-to-peer overlays are easily recognizable by a characteristic feature – an identifier (key) space. Peers and resources are mapped into that identifier space and have an unambiguous position within it. Furthermore, every peer has a responsibility area within the identifier space and the peer manages all the resources with the identifiers which fall into that area. A more formal description on the identifier space and the related concepts will be provided in Chapter 3.

We classify structured overlays by the hash function used to map peers and resources into the identifier space. There are two distinct groups: those that use uniform hash functions and those that support non-uniform ones. The most basic way to assign identifiers to peers and resources is using uniform hash functions like SHA-1 because of the load balancing effect that such hashing provide. Such uniform hashing produces random identifiers on the identifier space for both: the participating peers and the available resources. This leads to a uniform distribution of resources on peers, which inherently avoids major load-balancing problems[7]. Thus, systems which use uniform hash functions are by far dominant in the field of structured peer-to-peer systems, like Chord, DKS, Pastry, Tapestry, Symphony, Kademlia and many others.

However, the use of the uniform hash functions comes with a price as well - the existing semantic relationship (correlation) between the peers or resources is inevitably lost with the use of a uniform hash function. This might be a considerable impediment for most of the data-oriented applications. Thus, despite easy handling of the load, the use of such systems is limited to applications which operate only with specific single key lookups (point-queries), i.e., retrieving resources based on their application specific identifier. Yet, most of the existing data-oriented applications require more sophisticated functionality to be supported (e.g., range queries) which becomes very inefficient to address when using uniform hash functions.

#### 2.1.2.4. Structured Overlays with Order-Preserving Hashing

To answer complex queries efficiently, peer-to-peer systems are required to use hash functions which preserve the semantic order among the resource keys in the key space, reflecting the relationship exhibited by the resources at the application level. Preserving such an order can significantly boost the performance of data-oriented peer-to-peer systems [Bharambe *et al.* 2004]. Alas, order-preserving hash functions imply that the resulting resource keys are likely to produce skewed distributions over the key space because of the non-uniform resource distribution at the application level. Since the topologies of most structured overlays are built assuming that peer-keys are distributed uniformly at random, the non-uniform resource hashing destroys a self-regulating load balancing property of these systems.

In order to balance the load in the system, peer-keys have to follow the distribution of resource keys. In this way, all peers are responsible for a similar number of resources in the system. This leads to skewed peer-key distributions in the presence of the skewed resource key distributions. Thus, most of the data-oriented applications have to rely on structured overlays which can support such non-uniform key distributions. However, not many peer-to-peer systems capable of handling such conditions have been developed. The most notable

[7] Some load imbalance due to a natural variance of the randomized hashing might still be observed [Stoica *et al.* 2001].

examples of these overlays are CAN, P-Grid, Mercury [Bharambe *et al.* 2004], Skip list based approaches ( [Aspnes *et al.* 2004; Aspnes and Shah 2003; Ganesan *et al.* 2004; Harvey *et al.* March 2003]) and Chord# [Schütt *et al.* 2005]. However, most of them have been suffering from specific drawbacks such as unbounded routing tables (P-Grid, CAN), failure to deal with complex skews (Mercury), or inability to work in heterogeneous environments (Chord#, SkipNets). More details on the problems related to some of these systems will be addressed in Chapter 5.

#### 2.1.2.5. Hybrid Overlays

There also exist many *hybrid* peer-to-peer overlays (super-peer systems) which trade-off between different degree of topology centralization and structure flexibility. Some prominent examples of such systems include Brocade [Zhao *et al.* 2002], LH$*$ [Litwin *et al.* 1996], Skype[8], KaZaA[9], etc. Hybrid overlays use a hierarchical network topology consisting of regular peers and super-peers, which act as local servers for the subsets of regular peers (Figure 2.1(c)). For example, a hybrid overlay might consist of the super-peers forming a structured network which serves as a backbone for the whole overlay, enabling an efficient communication among the super-peers themselves. Hybrid overlays have an advantage over simple centralized networks, since the super-peers can be dynamically replaced by regular peers, hence do not constitute single points of failure while retaining the benefits of peer-to-peer systems with central components.

However, on the down side, allocating peers as super-peers is not a straightforward task. Moreover, the design of super-peer systems implies that there are only two distinct and predefined classes of peers; whereas in reality there is a wide range of peers with diverse resource potentials, which continuously change over time. Thus, only the systems which can adapt to ever-changing heterogeneous peer environments (e.g., Oscar, cf. Chapter 5) can fully employ the entire potential of the available resources.

### 2.1.3. Routing

Peer-to-peer overlay networks enable the peers to communicate with one another even if the communicating peers do not know their addresses in the underlying network. For example, in an overlay deployed over the Internet, a peer can communicate with another peer without knowing its IP address. The way this is achieved in the overlays is by *routing* overlay messages. Each overlay message originates at a source and is forwarded by the peers in the overlay until the message reaches one or more destinations. A number of routing schemes have been proposed.

[8] http://www.skype.com/
[9] http://www.kazaa.com/

#### 2.1.3.1. Routing in Unstructured Overlays

Unstructured overlay networks use mainly two mechanisms to deliver routed messages: flooding and random walks. When some peer $v$ receives a flood message from one of its overlay neighbors $w$, then $v$ forwards the flood message to all of its neighbors except $w$. When $v$ receives the same flood message again, it is ignored. Eventually the flood reaches all of the destinations. Figure 2.2 illustrates typical routing operations in unstructured overlays. The circles and solid lines represent the overlay topology. The dashed arrows illustrate the flow of messages. Peers routing in an unstructured network do not know the exact location of the destinations so they have to either look in all possible directions via flooding or randomly walk to find the destination peer.

For example, in Gnutella, a file-sharing peer-to-peer system, a peer $s$ that wants to download a file floods the network with queries. If some peer $d$ that has the file desired by $s$ is reached by the query flood, then $d$ sends a response back to $s$. Flooding consumes a significant amount of network bandwidth. To reduce it, the flooded messages typically contain a Time-To-Live (TTL) counter included in every message that is decremented whenever the message is forwarded. This limits how far the flood can spread from the source but at the same time lowers the chance of reaching the peer that holds the searched file.

The high bandwidth usage of flooding has led to the design of an alternative routing scheme for unstructured overlay networks: *random walks* [Lv *et al.* 2002]. Instead of forwarding a message to all of the neighbors, it is only forwarded to a randomly chosen one. Depending on the network topology random walks provide different guarantees of locating the destination peer(s), however, all of the random walk approaches share one disadvantage: a significant and in most cases intolerable delivery latency.

#### 2.1.3.2. Routing in Structured Overlays

As more peers join the overlay network and there are more messages that need to be routed, flooding and random walks quickly reach their scalability limits. This problem has prompted the research on structured overlays (and the development of super-peer networks).

In structured overlays each peer has a unique and immutable identifier chosen when the peer joins the overlay. The peer identifiers enable efficient routing in structured overlays. Each routed message has a destination identifier selected from the peer identifier set. Instead of blindly forwarding the message to all neighbors as in the unstructured overlays, a peer in a structured overlay uses the destination identifier to forward the message only to one neighbor. The next hop neighbor is selected to minimize the distance in the identifier space from the current message holder to the destination.

Such routing is possible only in peer-to-peer overlay networks which posses an identifier space with a notion of distance. For example, in Chord [Stoica *et al.* 2001] identifiers are selected from the set of integers $[0, 2^m - 1]$ and are ordered in a modulo $2^m$ circle, where $m = 160$.

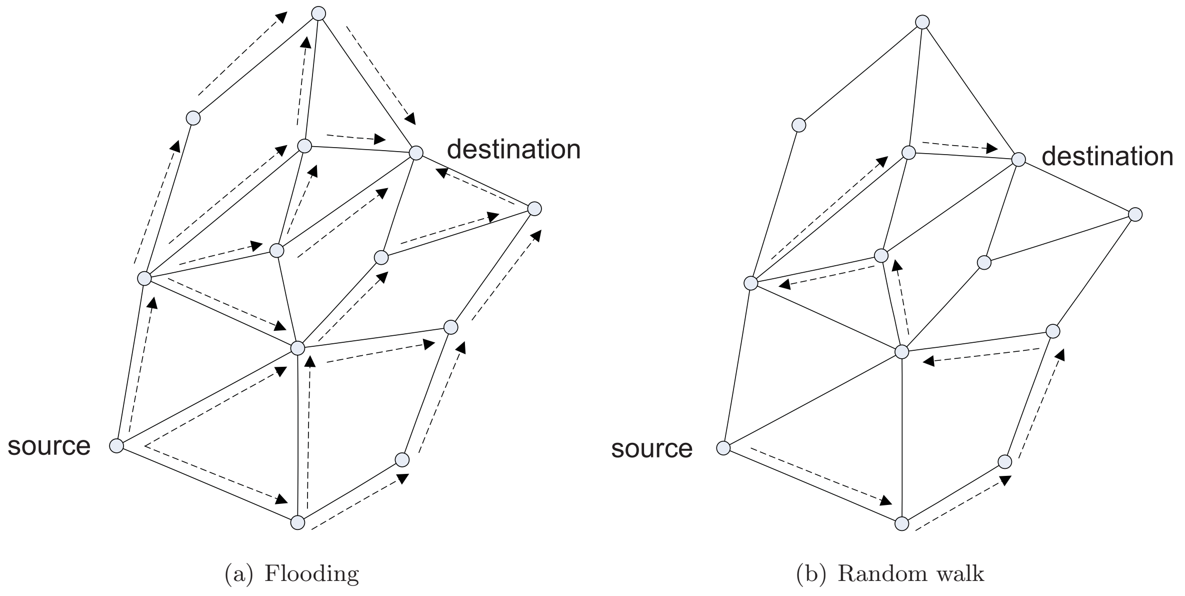


**Figure 2.2.** Routing in unstructured overlay networks

The distance $d(x, y)$ between two identifiers $x$ and $y$ is defined as the difference between $x$ and $y$ on that identifier circle, i.e., $d(x, y) = (y - x) \bmod 2^m$. In another overlay, Kademlia [Maymounkov and Mazières 2002], the identifiers are 160-bit integers and the distance between two identifiers $x$ and $y$ is defined as their exclusive bitwise OR (XOR) interpreted as an integer, i.e., $d(x, y) = x \oplus y$. Figure 2.3 illustrates the typical routing procedure in the Chord overlay. In this figure, the big circle represents the peer identifier space with IDs in the interval $[0, 2^5]$. The small circles are the peers and the numbers beside them are their IDs. Peer 7 is connected (solid arrows) to peers with exponentially increasing distance from 7: $7 + 1 = 8$, $7 + 2 = 9$, $7 + 4 = 11$, $7 + 8 = 15$, $7 + 16 = 23$. Assume that peer 7 wants to route a message to peer 28. Dashed arrows represent the routing path. Peer 23 is the neighbor of 7 that is closest on the ring to the destination (28) and that neighbor is chosen as the first hop for the message delivery. One hop is not enough to reach 28, but peer 23 brings the message closer to the destination and peer 27 finally delivers it. The greedy routing rule of always selecting as next hop the one that brings the message as close as possible to its destination is the main building block of all structured overlay networks.

Although current structured overlays differ in the details of how they make use of the peer identifiers for routing, they are all based on the same general *greedy routing* principle. When some peer $v$ receives a message with a given destination identifier, it forwards the message to that next hop whose identifier is the closest to the destination identifier. In other words, in every hop the message gets as close as possible to the destination. Routing terminates when one of the peers decides it is the destination for the message. This decision is application dependent. For example, in a Distributed Hash Table each peer knows for which hash table

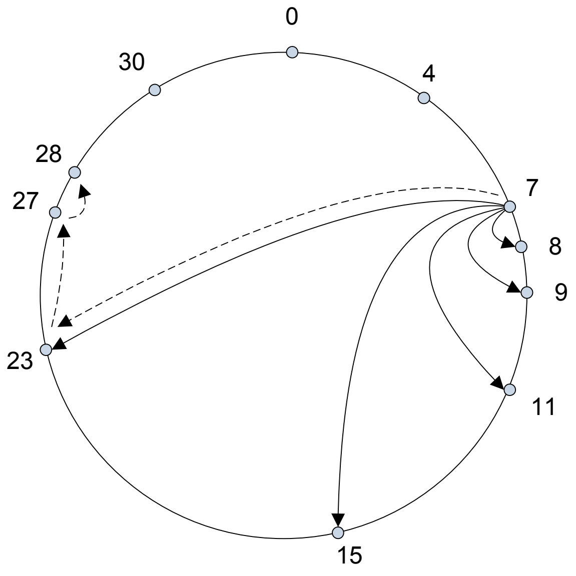


**Figure 2.3.** Routing in Chord

keys it is responsible for. The key space is mapped onto the peer identifier space in DHTs and the destination identifier of each DHT lookup message specifies the hash table key $K$ the lookup is querying for. The lookup message is greedily routed hop by hop from the origin until the message reaches a peer responsible for $K$. Routing then terminates and the responsible peer sends a lookup response to the origin. The lookup response contains the hash table value stored under the key $K$ as long as it exists.

### 2.1.4. Maintenance

Peer-to-peer systems are commonly deployed in environments characterized by high dynamicity, peers can depart or join the system at any time. These continuous joins and departures are commonly referred to as *churn*. Instead of gracefully departing from the network peers can also abruptly fail or the network connection with some of its neighbors may be closed. In all of these cases the changes in the routing tables may adversely affect the performance of the system. The overlay topology needs to be *maintained* to guarantee message delivery and routing efficiency.

There are two main approaches to overlay maintenance: proactive and reactive. In *proactive maintenance* peers periodically update their routing tables such that they satisfy the overlay topology invariants. For example, Chord periodically runs a "stabilization" protocol to ensure that every peer is linked to other peers at exponentially increasing distance. This ensures routing efficiency. To ensure message delivery each Chord peer maintains connections to its immediate predecessor and successor on the Chord ring.

In contrast to proactive maintenance, *reactive maintenance* is triggered immediately after the detection of a peer failure or peer departure. The missing entry in the routing table is replaced with a new one by sending a connect request to an appropriate peer.

Failures and departures of peers are detected in two ways: by probing or through usage. In probe-based failure detection each peer continuously runs a ping-response protocol with each of its neighbors. When ping timeouts occur repeatedly the neighbor is considered to be down and is removed from the routing table. In usage-based failure detection when a message is sent to a neighbor but not acknowledged within a timeout, the neighbor is considered to have failed.

The more neighbors a peer must maintain the higher the bandwidth overhead incurred by the maintenance protocol. In modern structured overlays maintenance bandwidth typically scales as $O(\log(N))$ in terms of the network size.

## 2.2. Peer-to-Peer Reference Model

The success of the peer-to-peer concept has created a huge diversity of approaches. A wide range of algorithms, structures, and architectures for overlay networks have been proposed, integrating knowledge from many different communities, such as networking, distributed systems, databases, graph theory, agent systems, complex systems, etc. The terminologies and abstractions used, however, have become quite inconsistent, which makes it very hard to assess and compare different approaches. Although a large number of overlay networks have been devised, only very few works on unifying architectures exist. Only a few relevant attempts to remedy this situation exist so far.

For example, JXTA [Gong 2001] defines a 3-layer architecture (kernel, services, application), XML-based communication protocols, and basic abstractions, such as peer groups, pipes, and advertisements. JXTA intends to provide a uniform programming platform for peer-to-peer applications and facilitate interoperability. It provides well-defined APIs and a clear separation of concerns in its architecture. However, it is not intended to describe the structural and functional properties of overlay networks. Dabek et al. [Dabek *et al.* 2003] propose a common API for structured overlays, basically for CAN [Ratnasamy *et al.* 2001], Chord [Stoica *et al.* 2001], Pastry [Rowstron and Druschel 2001], and Tapestry [Zhao *et al.* 2004]. The API only takes into account structured overlays and the used abstraction are at a very low level (C programming interface level), so that using it as a general architecture for modeling overlay networks is not possible. In [Gummadi *et al.* 2003] classifications for structured overlay networks, e.g., deterministic and randomized networks, are introduced, however, it does not provide a general reference architecture for peer-to-peer networks.

In the following chapter we will present such a reference model for overlay networks which is capable of modeling different approaches in this domain in a generic manner. It is intended to allow researchers and users to assess the properties of concrete systems, to establish a common vocabulary for scientific discussion, to facilitate the qualitative comparison of the systems, and to serve as the basis for defining a standardized API to make overlay networks interoperable.

# Chapter 3

# Reference Architecture for Overlay Networks

Believe me, that was a happy age, before the days of architects, before the days of builders.

Lucius Annaeus Seneca

## 3.1. Overview

In this chapter we introduce a reference model for overlay networks which is capable of modeling all existing approaches in this domain. We focus on *decentralized overlay networks* such as Gnutella[1], Freenet [Clarke *et al.* 2001], CAN [Ratnasamy *et al.* 2001], Chord [Stoica *et al.* 2001], P-Grid [Aberer *et al.* 2005b], DKS [Alima *et al.* 2003], etc., as this class is the most relevant one. From a modeling point of view, *centralized peer-to-peer (P2P) systems*, such as Napster, are simply client-server architectures where the participants can directly communicate after a discovery phase (similar to a DNS name lookup and then contacting a web server, for example). *Hierarchical P2P systems* such as Kazaa, basically consist of a decentralized overlay network of super-peers for locating resources that are used by the normal peers. Thus, these system can be modeled by our proposed model with an additional client-server step when contacting a super-peer.

The model is intended to support the assessment of system properties, establishes a common vocabulary, facilitates the qualitative comparison of the systems, and can serve as the basis for defining a standardized API to make overlay networks interoperable. The major contributions of our model are (1) a conceptual model capturing the concept of embedding a graph into a virtual identifier space, which is fundamental for all overlay networks and (2) a well-defined peer architecture comprising user-level interfaces for applications wanting to use the

[1] http://www.gnutellaforums.com/

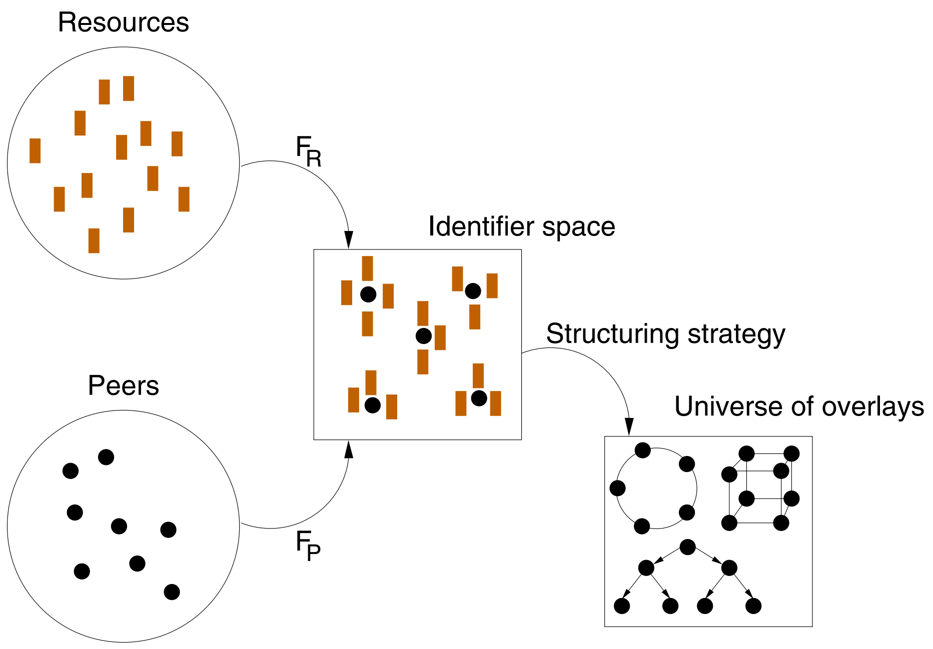


**Figure 3.1.** Overlay network design decisions

overlay, interfaces for intra-network communication among homogeneous peers, and interfaces for cooperation among heterogeneous overlay networks.

## 3.2. Conceptual Model for Overlay Networks

In any overlay network a group of peers $\mathcal{P}$ provides access to a set of resources $\mathcal{R}$ by mapping $\mathcal{P}$ and $\mathcal{R}$ to an (application-specific) identifier space $\mathcal{I}$ using two functions $F_P : \mathcal{P} \to \mathcal{I}$ and $F_R : \mathcal{R} \to \mathcal{I}$. These mappings establish an association of resources to peers using a closeness metric on the identifier space. To enable access from any peer to any resource a logical network is built, i.e., a graph is embedded into the identifier space. These basic concepts of overlay networks are depicted in Figure 3.1.

Each specific overlay network is characterized by the decisions made on the following six key design aspects:

1. choice of an identifier space
2. mapping of resources and peers to the identifier space
3. management of the identifier space by the peers
4. graph embedding (structure of the logical network)
5. routing strategy

6. maintenance strategy

In taking these design decisions the following key requirements for overlay networks are addressed:

**Efficiency**: Routing should incur a minimum number of overlay hops (with minimum "physical" distance) and the bandwidth (number and size of messages) for constructing and maintaining the overlay should be kept minimal.

**Scalability**: The concept of scalability includes many aspects. We focus on numerical scalability, i.e., very large numbers of participating peers without significant performance degradation.

**Self-organization**: The lack of centralized control and frequent changes in the set of participating peers requires a certain degree of self-organization, i.e., in the presence of *churn* the overlay network should self-reconfigure itself towards stable configurations. This is a *stabilization* requirement as external intervention typically is not possible.

**Fault-tolerance**: Participating nodes and network links can fail at any time. Still all resources should be accessible from all peers. This is typically achieved by some form of redundancy. This is also a *stabilization* requirement for the same reason as above. Fault-tolerance implies that the *partial failure* property of distributed systems [Tel 1994] is satisfied, i.e., even if parts of the overlay network cease operation, the overlay network should still provides an acceptable service.

**Cooperation**: Overlay networks depend on the cooperation of the participants, i.e., they have to trust that the peers they interact with behave properly with respect to routing, exchange of index information, quality of service, etc.

In the following we will provide detailed formal specifications for these key design concepts of overlay networks and discuss the issues related to the requirements listed above.

### 3.2.1. Choice of Identifier Space

A central decision in designing an overlay network is the selection of the *virtual identifier space* $\mathcal{I}$ which has to possess some *closeness metric* $d : \mathcal{I} \times \mathcal{I} \rightarrow R$, where $R$ denotes the set of real numbers. $d$ must satisfy properties 1–3 below and if possible should satisfy properties 4–5.

$$\forall x, y \in \mathcal{I} \; : \; d(x,y) \geq 0 \qquad (1)$$

$$\forall x \in \mathcal{I} \; : \; d(x,x) = 0 \qquad (2)$$

$$\forall x, y \in \mathcal{I} \; : \; d(x,y) = 0 \;\Rightarrow\; x = y \qquad (3)$$

$$\forall x, y \in \mathcal{I} \; : \; d(x,y) = d(y,x) \qquad (4)$$

$$\forall x, y, z \in \mathcal{I} \; : \; d(x,z) \leq d(x,y) + d(y,z) \qquad (5)$$

If $d$ satisfies all the five properties then $(\mathcal{I}, d)$ is a metric space. However, in many cases only the first three properties will be satisfied. In this case we call $(\mathcal{I}, d)$ a *pseudo-metric space.*

The choice of the virtual identifier space is important for several reasons:

- *Addressing:* The identifier space plays the role of an address space used for identifying resources in the overlay network. Each peer and resource in an overlay network receives a virtual identifier taken from $\mathcal{I}$ (explicitly or implicitly).
- *Scalability:* To support very large systems, $\mathcal{I}$ has to be very large. Through a mapping $F_P$ each peer with a physical address in $\mathcal{P}$ is assigned a virtual identifier from $\mathcal{I}$. This is an application of the well-known principle of indirection for achieving numerical scalability.
- *Location-independence:* The virtual identifier space allows peers to communicate which each other irrespective of their actual physical location. This addresses physical address changes and enables mobility.
- *Clustering of resources with peers:* The closeness metric $d$ enables the clustering of resources with peers based on proximity. This is discussed in detail in Section 3.2.3.
- *Message routing:* Virtual identifiers and the closeness metric $d$ are essential for realizing efficient routing.
- *Preservation of application semantics:* As virtual identifiers can be defined in an application-specific way, application semantics, for example, "proximity" of resources (clustering), can be preserved.

**Examples:** CAN [Ratnasamy *et al.* 2001] uses a Euclidean space with virtual identifiers being coordinates in this space. The distance function $d$ is the Euclidean distance. P-Grid [Aberer *et al.* 2005b] uses a prefix-preserving hash function on strings, i.e.,

$$\forall s_1, s_2 : s_1 < s_2 \Rightarrow h(s_1) < h(s_2)$$

($<$ denotes lexicographic order). Identifiers in P-Grid are bit strings and $d$ is defined as (for a $k$-bit identifier $a$ and an $l$-bit identifier $b$):

$$d(a,b) = min(|\sum_{i=1}^{k} a_i 2^{-i} - \sum_{i=1}^{l} b_i 2^{-i}| \ , \ 1 - |\sum_{i=1}^{k} a_i 2^{-i} - \sum_{i=1}^{l} b_i 2^{-i}|)$$

while in Chord [Stoica *et al.* 2001] and DKS [Alima *et al.* 2003] the identifier space is a subset of the natural numbers of size $N$ and

$$d(x,y) = (y - x) \ mod \ N.$$

### 3.2.2. Mapping to the Identifier Space

The mapping $F_P : \mathcal{P} \to \mathcal{I}$ associates peers with a unique virtual identifier from $\mathcal{I}$. Different approaches can be distinguished by the properties of the chosen functions $F_P$:

- *Completeness:* $F_P$ may be complete or partial. When $F_P$ is partial, peers might (temporarily) not be associated with an identifier.

- *Morphism:* If no replication (for fault-tolerance) is required, $F_P$ will be one-to-one (injective), i.e., $\forall p, q \in \mathcal{P} : p \neq q \;\Rightarrow\; F_P(p) \neq F_P(q)$. However, the more typical case is that the system uses replication and the mapping is not injective.
- *Dynamicity:* $F_P$ can be either statically defined, e.g., by its physical address or other unique attributes, or dynamically change over time. In order to simplify our notations, in the following we will focus on the structural aspects and will not explicitly represent time-dependency in our notations.

Additionally, $F_P$ may satisfy certain distributional properties, for example, that the range of values of $F_P$ follows a certain distribution in space $\mathcal{I}$, e.g., uniform. Such properties may then be exploited, for example, for load balancing. The properties $F_P$ satisfies will be denoted as $\mathcal{C}_{F_P}$ in the following.

The mapping $F_R : \mathcal{R} \rightarrow \mathcal{I}$ associates resources with identifiers from $\mathcal{I}$. The choice of this mapping can be critical for the application using the resources. Typically "semantic closeness" of resources, e.g., resources frequently requested jointly, can be translated into closeness of identifiers. Thus, the possibility of using application-specific identifiers is taking advantage of this. If the resources should be identified uniquely, $F_R$ has to be injective. The distribution of identifiers generated by $F_R$ has an important impact on the load-balancing properties of the overlay network embedded into the space $\mathcal{I}$.

**Examples:** A standard example for $F_P$ and $F_R$ is a uniform hashing function as, e.g., used by Chord [Stoica *et al.* 2001]. This will generate a uniform distribution of peers on the identifier space and implicitly provides load-balancing as also the resource identifiers are uniformly distributed. However, clustering of information will not be possible and thus higher-level search predicates such as range queries will be expensive to process. P-Grid's mapping functions on the other hand supports clustering but thus requires an explicit load-balancing strategy.

### 3.2.3. Management of the Identifier Space

At any point in time, $\mathcal{I}$ is managed by the set of current peers $\mathcal{P}$. The responsibility for peers for specific identifiers is captured by a function $\mathcal{M} : \mathcal{I} \rightarrow 2^{\mathcal{P}}$, which associates with each identifier of a resource $r$, $i = F_R(r) \in \mathcal{I}$, the set of peers that are managing $r$. Through $\mathcal{M}$, each peer $p$ is assigned *responsibility* for the set $\mathcal{M}^{res}(p)$ of identifiers, where $\mathcal{M}^{res}(p) = \bigcup_{Q \in 2^{\mathcal{P}}, p \in Q} \mathcal{M}^{-1}(Q)$. Locating a resource $r$ corresponds to finding a peer in $\mathcal{M}(F_R(r))$. The lookup operation of overlay networks typically provides an implementation of $\mathcal{M}$ through routing. We may identify various basic properties for $\mathcal{M}$:

- *Completeness:* $\mathcal{M}$ may be complete or partial. When $\mathcal{M}$ is incomplete, identifiers might (temporarily) not be associated with a peer. Typically the mapping will be complete, such that each point of the identifier space is under the responsibility of some peers, i.e., $\forall i \in \mathcal{I} : \exists p \in \mathcal{P} : p \in \mathcal{M}(i)$

- *Cardinality:* To provide fault-tolerance, $\mathcal{M}$ typically contains more than one element, i.e., a set of peers is responsible for managing each identifier ($\forall i \in \mathcal{I} : |\mathcal{M}(i)| > 1$).
- *$\mathcal{M}$ induced by proximity:* A standard way to specify $\mathcal{M}$ is that identifiers are associated with their closest peers, i.e.,

$$p \in \mathcal{M}(i) \Rightarrow d(F_P(p), i) = min_{q \in \mathcal{P}} d(F_P(q), i).$$

- *Dynamicity:* $\mathcal{M}$ typically changes dynamically as the set of peers and their mapping to the identifier space changes.
- *Uniformity of replication:* The cardinality of $\mathcal{M}$ (which corresponds to the degree of replication) may be constant or uniformly distributed to ensure comparable availability of resources. Non-uniform distributions can be used to adapt the availability of resources to application requirements, e.g., popularity of resources.

In the following, $\mathcal{C}_{\mathcal{M}}$ denotes the properties $\mathcal{M}$ satisfies.

**Examples:** In Chord a peer with virtual identifier $a$ is responsible for the interval $(predecessor(a), a]$. In P-Grid a peer with a $k$-bit path $a$ is responsible for all identifiers in the interval

$$\left[\sum_{i=1}^{k} a_i 2^{-i} \ , \ 2^{-k} + \sum_{i=1}^{k} a_i 2^{-i}\right).$$

## 3.2.4. Graph Embedding

An overlay network can be modeled as a *directed graph*, $G = (\mathcal{P}, \mathcal{E})$, where $\mathcal{P}$ denotes the set of vertices (i.e., peers) and $\mathcal{E}$ denotes the set of edges. Due to the dynamics in overlay networks, $G$ is time-dependent, but as before we will not explicitly denote this. By virtue of this graph we define a *neighborhood* relationship $\mathcal{N} : \mathcal{P} \to 2^{\mathcal{P}}$, such that for a given peer $p$, $\mathcal{N}(p)$ is the set of peers with which peer $p$ maintains a connection, i.e., there is a directed edge $(p, q)$ in $\mathcal{E}$ for $q \in \mathcal{N}(p)$.

The properties of the overlay network relate to properties of the directed graph generated by $\mathcal{N}$ and to the properties of the embedding of the graph into the (pseudo-) metric space $(\mathcal{I}, d)$. Purely structural properties of the graph can be further distinguished into local and global properties, i.e., whether they relate to local characteristics of graph nodes or to global characteristics of the graph. Typical global properties of the graph are the following:

- *Uniqueness:* For deterministic systems, e.g., Chord, DKS, for a given set $\mathcal{P}$ and mapping $F_P$ only one valid network $\mathcal{N}$ exists. In randomized systems such as P-Grid and randomized Chord, multiple valid $\mathcal{N}$ are possible.
- *Graph diameter:* A small diameter provides lower bounds on the latency of routing in the network.

- *Connectivity*: Some overlay network approaches may require that the overlay graph is connected at any time.

- *Distributional properties:* These are typically distributional properties of node degrees. A frequently occurring class of graphs are power-law graphs [Mitzenmacher 2001]. Other distributional properties relate to the clustering coefficient of the graph.

Typical local properties of the graph include:

- *Minimal out-degree:* This property is beneficial to ensure fault-tolerance, when many neighbors fail.

- *Maximal out-degree:* This property is relevant for ensuring bounded maintenance cost for connections to other peers.

- *Distributional properties of in-degree:* These are relevant for load balancing in the message forwarding.

More complex properties refer to relationships of the graph structure to the distance function. These relationships are tightly intertwined with the strategy for efficient routing in an overlay network. Typical examples of such constraints are:

- *Local connectivity:* This property ensures that peers are connected to some specific subset of their immediate neighbors. An example of such a requirement for a given peer $p$ would be
$$\forall q \in \mathcal{P} : d(F_P(p), F_P(q)) < d_{min} \Rightarrow q \in \mathcal{N}(p).$$

- *Long-range connectivity:* Many overlay network designs are structurally similar to Small-World graphs as introduced by Kleinberg [Kleinberg 2000]. These graphs are constructed such that long range connections satisfy the condition
$$P[q \in \mathcal{N}(p)] \propto \frac{1}{d(F_P(p), F_P(q))^{-d}},$$
where $d$ is the dimensionality of the identifier space. Many overlay networks satisfy more strict variations of this condition.

The properties $\mathcal{N}$ satisfies are denoted by $\mathcal{C}_{\mathcal{N}}$ in the following. At this point we are able to completely characterize the structural aspects of overlay networks by the following definition:

**Definition.** The structure of an overlay network $O \in \mathcal{O}$ for a set of peers $\mathcal{P}$ is given by
$$O = (\mathcal{I}, d, F_P, \mathcal{C}_{F_P}, \mathcal{M}, \mathcal{C}_{\mathcal{M}}, \mathcal{N}, \mathcal{C}_{\mathcal{N}}).$$

### 3.2.5. Routing Strategy

The basic service an overlay network provides is to route a request for an identifier $i$ to a peer $p_r$ responsible for it, i.e., $p_r \in \mathcal{M}(i)$. Routing is a distributed process using the overlay network. We model it by asynchronous message passing: $route(p, i, m)$ forwards a message $m$ to a peer $p$ responsible for $i$. A routing strategy can be described by a potentially non-deterministic function $\mathcal{R} : \mathcal{P} \times \mathcal{I} \rightarrow 2^P$, which selects at a given peer $p$ with neighborhood $\mathcal{N}(p)$ for a target identifier $i$ the (set of) next peers $\mathcal{R}(p, i) \in \mathcal{N}(p)$, to which the message is forwarded. In structured overlay networks routing typically is greedy, i.e.,

$$d(i, F_P(q)) < d(i, F_P(p))$$

for all $q \in \mathcal{R}(p, i)$. In unstructured overlay networks the set $\mathcal{R}(p, i)$ may contain several peers.

Properties of routing algorithms are characterized by their associated cost measures, such as latency, number of hops, and probability of successful routing. Given a routing algorithm together with an overlay network structure the properties regarding the expected usage of the peers' resources can be analyzed.

### 3.2.6. Maintenance Strategy

Participation of peers in an overlay network dynamically changes over time. Each peer can freely decide to join or leave an overlay network at any time. These changes, referred to as *churn* in the literature, can happen quite frequently. To maintain the structural integrity of an overlay network a *maintenance strategy* is required, which compensates for changes to the network structure due to peers going offline or failure of network connections.

In all overlay networks, joining the network is done explicitly by a join operation, whereas leaving typically is implicit as peers may simply go offline or crash or their network connection may drop. Regardless whether peers leave gracefully or not, changes in the participation in an overlay network typically require the application of a maintenance strategy. Aside from access control aspects, i.e., who is allowed to participate, this basically requires to repair routing tables which have been invalidated due to churn, i.e., to maintain the connectivity of the underlying graph [Ghodsi *et al.* 2004]. Maintenance strategies can be classified [Aberer *et al.* 2004] into *proactive correction* (PC) using periodic probing or heartbeats to repair inconsistencies, and *reactive mechanisms*, with the sub-categories *correction on use* (CoU), e.g., P-Grid and DKS, *correction on failure* (CoF), e.g., P-Grid, and correction on change (CoC), e.g., Chord.

The practical usability of an overlay network critically depends on the efficiency of the maintenance strategy. The goal is to maintain a "sufficient" level of consistency while minimizing effort. Since a dynamically evolving overlay network on top of a dynamically changing physical network is a complex dynamical system, the goal is to arrive at a stable dynamic equilibrium for a variety of conditions while guaranteeing successful routing.

### 3.2.7. Other Properties

There are a number of further properties which we can only briefly mention here due to space limitations.

Constraints, such as those introduced in the previous sections, can be guaranteed at different levels of strictness: If the constraints are valid all the time, they are *invariants* of the system; if the constraints hold *eventually*, they may be satisfied after *self-stabilization* of the system induced by changes to the systems state (e.g., [Dolev *et al.* 2007; Shaker and Reeves 2005]); if the constraints hold *probabilistically*, they are satisfied with a specific probability either all the time or eventually.

By taking into account the *physical characteristics* of peers, such as their network location, their storage capacity, etc., additional properties can be specified which are in particular useful to obtain insights and control over the performance characteristics of the overlay network, for example, efficiency of routing and reliability of the network. An important example of such a property is *locality of routing.* A possible formulation of such a property is a constraint on the stretch introduced by the overlay network, i.e., that the physical distance of the path traversed to reach a node does not exceed the distance of the shortest physical path by a given stretch factor.

## 3.3. Reference Architecture

From an application-oriented perspective, any middleware technology—and we see P2P systems and specifically overlay networks as a form of middleware—should provide powerful and easy to use abstractions that hide implementation details as much as possible from the user/implementer while offering enough control and access options to actually meet application requirements. Additionally, the abstractions should be defined in a way that the concrete infrastructure implementing the middleware functionality can be replaced without requiring to rewrite code.

Given these goals, we see P2P systems based on overlay networks as layered systems as depicted in in Figure 3.2 (for a single node). From a user's perspective a P2P system facilitates to realize a specific application by sharing resources with other users and using services provided by the P2P layer. One particularly important example of such a service is P2P data storage, which allows to insert, search, and access data items. This service as well as the applications take advantage of the basic resource location service provided by the P2P basic layer that implements the overlay network.

This simple layered architecture supports separation of concern between the application layer, the generic services of a P2P system and the basic overlay network of a P2P system. It facilitates to replace a specific implementation of a P2P system, or selected services and layers, that an application is using by alternative implementations. In order to support this form of

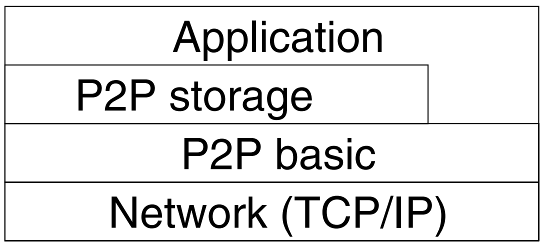


**Figure 3.2.** Layered architecture view

modularity it is important to provide a standardized specification of the interfaces among the layers. In Figure 3.3 we provide a class diagram that provides the core of such an interface specification. It is based on the conceptual model we have introduced in Section 3.2.

Overlay networks are based on the embedding of a graph into an *Identifier Space* which provides a closeness metric. Each *Peer* is mapped into this space, i.e., it is assigned an *Identifier* from the virtual identifier space, which defines its current position in this space and (indirectly) the subset of identifiers the peer is responsible for as described in Sections 3.2.2 and 3.2.3. Note that a peer's position can change over time. How the partitioning of the identifier space is done, i.e., how a node is assigned a coordinate and responsibility, is subject to the specific overlay approach. A *Peer* is uniquely identified by an immutable name (*immutableName)* and maintains a neighborhood (*neighbors*), i.e., references to other peers (*PeerReference*) for forwarding. Each *PeerReference* includes the referenced peer's immutable name, its position in the identifier space, i.e., responsibility, and physical network address (IP address or symbolic name). As this information changes over time, each peer has to apply a maintenance strategy as discussed in Section 3.2.6 to have a consistent view (depending on the specific overlay network). The number of neighbors a peer maintains and the strategy how neighbors are selected is defined by the *Constraints* of the overlay network which depend on properties of the identifier-resource and identifier-peer association strategies, the graph embedded in the identifier space, and its constraints, etc.

As shown in in Figure 3.2, we distinguish two layers of functionality. The basic layer (*P2P Basic Interface*) provides the low-level operations which the overlay needs to be able to function. Its main functionalities, besides the mandatory *join* and optional *leave* operations, are the *lookup* and *route* operations. The *lookup* function allows an application to find a peer by its identifier to be able to directly communicate with it (point-to-point), for example, for transferring data items. The *route* operation, which lookup typically builds on, allows the user to send a message to any peer responsible for a given identifier. A message can contain any data specified by the application, for example, the data to be stored by the peer or a synchronization request among replicas. *routeToReplicas* propagates a given message to the set of peers responsible for the same identifier. *getLocalPeer* returns the administrative information about the local peer and *getNeighbors* provides the list of neighbors of a peer, i.e., its routing table information.

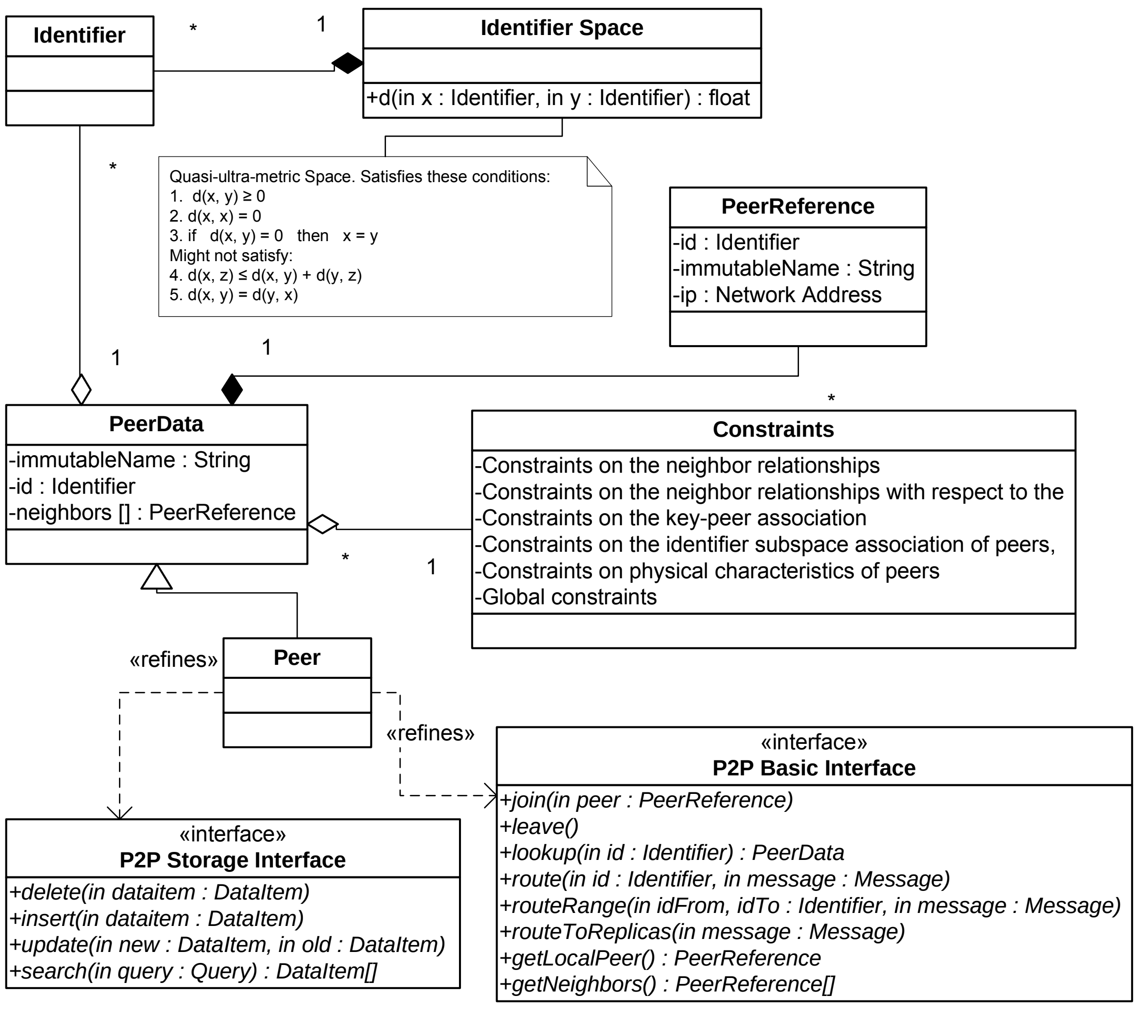

Identifier
*
1
Identifier Space
+d(in x : Identifier, in y : Identifier) : float
*
Quasi-ultra-metric Space. Satisfies these conditions:
1. d(x, y) ≥ 0
2. d(x, x) = 0
3. if d(x, y) = 0 then x = y
Might not satisfy:
4. d(x, z) ≤ d(x, y) + d(y, z)
5. d(x, y) = d(y, x)
PeerReference
-id : Identifier
-immutableName : String
-ip : Network Address
1
1
*
PeerData
-immutableName : String
-id : Identifier
-neighbors [] : PeerReference
Constraints
-Constraints on the neighbor relationships
-Constraints on the neighbor relationships with respect to the
-Constraints on the key-peer association
-Constraints on the identifier subspace association of peers,
-Constraints on physical characteristics of peers
-Global constraints
*
1
«refines»
Peer
«refines»
«interface»
P2P Basic Interface
+join(in peer : PeerReference)
+leave()
+lookup(in id : Identifier) : PeerData
+route(in id : Identifier, in message : Message)
+routeRange(in idFrom, idTo : Identifier, in message : Message)
+routeToReplicas(in message : Message)
+getLocalPeer() : PeerReference
+getNeighbors() : PeerReference[]
«interface»
P2P Storage Interface
+delete(in dataitem : DataItem)
+insert(in dataitem : DataItem)
+update(in new : DataItem, in old : DataItem)
+search(in query : Query) : DataItem[]

**Figure 3.3.** Conceptual decomposition

The storage layer (*P2P Storage Interface*) builds on these functionalities and provides the typical data management functionalities of inserting, updating, deleting, and querying data, that made the P2P paradigm popular. The resources affected by the functions are specified via the *DataItem* abstraction that includes the resource's data and the application-specific key(s) to be used by the storage layer to generate a corresponding identifier, i.e., map the data item to its position in the identifier space. This can then be used by the basic layer to find the responsible peer(s) and perform the requested operation. The *DataItem* set returned by *search* includes both the application-specific keys and the identifiers of the found resources.

We would like to emphasize that Figure 3.3 provides a *minimal* model, i.e., it provides what we identified as the minimal common denominator for different overlay network approaches. All parts of the architecture can be (and in fact are) extended by concrete systems. For example, each system will typically have more structured message types. For example, in Gnutella, as one of the simplest systems, a *join* operation would mean the issuing of a *Ping* message which has a simple structure holding a descriptor ID (to prevent loops in the routing), a payload descriptor, a time-to-live counter, a hop counter and a field defining the length of the payload. Yet, extensions of our model are intuitive and simple: A concrete system can basically "subclass" and "extend" any of the components in Figure 3.3.

## 3.4. Interoperability

Up to now we have introduced a conceptual model and abstract interfaces to capture the specific properties of a given overlay network approach. In practice, multiple overlay networks will co-exist simultaneously in a physical network, which raises issues of managing multiple overlay networks and interoperability.

We consider an overlay network as a group of peers $\mathcal{P}$ that share the same specification of their specific overlay network mechanisms. The sharing of this specification is a problem of group management and can be done either explicitly or implicitly.

With an *explicit management* explicit group identifiers $\mathcal{G}$ (e.g., URIs) are used to identify an overlay network and are bound to a specific type of overlay network by a mapping $\mathcal{T} : \mathcal{G} \rightarrow \mathcal{O}$, which associates the identifier with a specification of an overlay network. We consider this as providing the overlay network with a type (or schema in database parlance). Thus, every peer joining a group $g \in \mathcal{G}$ obtains the associated type information and adheres to the specification. The issue of non-complying peers is related to security and trust which we cannot elaborate further here. As a consequence, joining an overlay network would only be possible if the joining peer uses the same group identifier as the peers of the network.

With *implicit management* a group of peers is considered as participating in the same overlay network if they use the same overlay network specification. Thus, there is no global knowledge on the existence of a specific overlay network, but the network results from the cooperation of peers using the same specification. Thus, when joining, a peer obtains/shares the specification with the peer to which it joins.

Another interesting aspect of group management in an overlay network is the *degree of coupling*. In *tightly coupled* overlay network the overlay graph is at any time connected. This implies that such an overlay network has to be initiated by a single peer (that could, for example, determine the identifier and specification of the network properties, when explicit group management is used). Chord is an example of a tightly coupled overlay network. In *loosely coupled* overlay networks different overlay graphs based on the same specification (e.g., using implicit group management) can evolve, merge, or split. Gnutella and P-Grid are examples of loosely coupled overlay networks.

The approach to implicitly manage groups of peers participating in the same overlay network suggests a more general view of how groups of peers constructing overlay networks may work together. In order to interact, it is in fact not necessary that the type of overlay network is exactly the same, but it may be sufficient that the specifications are compatible. This approach can be observed for some practical overlays systems, such as Gnutella. Multiple versions of overlay protocols can work together, and different peers may use different policies, e.g., with respect to network connectivity.

For characterizing the possibilities of interoperability among peers participating in different overlay networks $O_1$ and $O_2$, we can systematically compare the specifications of the networks. We assume that at the level of protocols, $O_1$ and $O_2$ are compatible by following the API defined in Section 3.3 and using compatible protocol messages. This is a purely syntactic agreement. The classification of interoperability follows the concepts described in Section 3.2 and we can distinguish the following levels of structural interoperability:

- *Compatible Identifiers:* The identifier spaces $\mathcal{I}_1$ and $\mathcal{I}_2$ are the same or can be related to each other by applying a transformation. Then for identifiers in $i \in \mathcal{I}_1 \cap \mathcal{I}_2$, peers from both $O_1$ and $O_2$ can route messages to the resources identified by $i$. Routing would be processed independently in $O_1$ and $O_2$. Thus, peers can play the role of gateways among different overlay networks.

- *Compatible Identifier Spaces:* If additionally the distance functions (possibly after applying a transformation) are compatible, peers from $O_1$ may use peers from $O_2$ (and vice versa) and their knowledge on neighbors to integrate them into their own routing tables.

- *Compatible Structures:* If additionally the structural constraints of two overlay networks are in a subsumption relationship, i.e., one of the overlay networks is more constrained but compatible with the more general overlay network, peers of the more constrained network may participate as peers in the less constrained network by adopting the routing and maintenance algorithms of the less constrained network.

An important open issue, when exploiting these forms of structural interoperability, are the effects on the performance of the routing and maintenance mechanisms and the impact on certain structural properties of the overlay networks, such as distributional properties. These questions are closely related to the study of overlay networks built by peers with highly heterogeneous resources, a topic which has been studied only to a very limited degree so far.

## 3.5. Validation of the Reference Architecture

In this section we will briefly describe key aspects of a representative set of overlay networks—Chord [Stoica *et al.* 2001], DKS [Alima *et al.* 2003], P-Grid [Aberer *et al.* 2005b], Pastry [Rowstron and Druschel 2001], Symphony [Manku *et al.* 2003], Freenet [Clarke *et al.* 2001], and Gnutella – in terms of our architecture to demonstrate its validity. Additionally, we provide a brief qualitative comparison of the systems.

### 3.5.1. Identifier Space

The identifier spaces are very similar for all logarithmic-style overlay networks (P-Grid, Chord, Pastry, Symphony, etc.). In these approaches identifiers are chosen from an alphabet with radix $b$, e.g., $b = 2$ for P-Grid, Chord, and Symphony, $b = 16$ for Pastry. Some of them limit the identifier length, e.g., Pastry uses 128-bit, Chord and DKS use 160-bit length identifiers, whereas in P-Grid identifiers can be of arbitrary length. A similar distance function is shared by all of these overlays, though there are some subtle differences. In P-Grid, Pastry, and Symphony the distance $d(u, v)$ of two identifiers $u$ (of length $k$) and $v$ (of length $l$) is

$$min\left(\left|\sum_{i=1}^{k} u_i b^{-i} - \sum_{i=1}^{l} v_i b^{-i}\right|,\ 1 - \left|\sum_{i=1}^{k} u_i b^{-i} - \sum_{i=1}^{l} v_i b^{-i}\right|\right).$$

Note that in Pastry's case $k = l$, and for these systems $d(u, v)$ is symmetric as $d(u, v) = d(v, u)$. For example, in P-Grid,

$$d(\text{“0000”}, \text{“10”}) = d(\text{“10”}, \text{“0000”}) = 0.5.$$

The identifier space in Chord is not symmetric, i.e., $d(u, v) \neq d(v, u)$. $d(u, v)$ can be defined as

$$\left(\left(\sum_{i=1}^{k} v_i 2^{-i} - \sum_{i=1}^{k} u_i 2^{-i}\right) + 1\right)\ mod\ 1.$$

Thus,

$$d(\text{“001”}, \text{“111”}) = 0.75,$$

but

$$d(\text{“111”}, \text{“001”}) = 0.25.$$

In Freenet the situation is slightly different. Due to the way Freenet identifies nodes, it uses an $r$-dimensional 160-bit identifier space. $r$ depends on the data items a peer stores, but usually $r = 50$. $d(u, v)$ between two Freenet peers is the Euclidean distance in this multidimensional space.

### 3.5.2. Mapping to the Identifier Space

**Mapping of peers:** The key difference among the overlays with respect to this mapping is whether the virtual identifier is assigned to a peer randomly or the peer adopts the identifier depending on environment conditions, e.g., depending on the data a peer and its neighboring peers store. In Chord, Pastry, and Symphony the virtual identifier is generated using some random function and assigned to a peer upon joining the overlay and remains stable. In DKS identifiers can be mapped order-preservingly based on their domain name, e.g., lexicographic ordering, to ensure that nodes in the same organizational domain are logically close in the identifier space.

Most of the logarithmic-style overlay approaches like Chord, Pastry, Symphony, or P-Grid have a one-dimensional identifier space. In Chord, and Pastry identifiers are assigned by hashing the node's IP address using SHA-1. In contrast, in P-Grid each peer initially is responsible for the whole identifier space and has an empty identifier which grows bit by bit in the lifetime of the peer depending on which other peers it encounters and what type of data they and the peer itself store. Similarly in Freenet, each node assigns itself an identifier vector of size $r$, consisting of $r$ 160-bit elements representing the $r$ identifiers of data items the peer stores. Additionally, the identifier of a Freenet peer changes during its lifetime depending on the queries it handles. Thus, the identifiers in P-Grid and Freenet dynamically change, whereas in Chord, Pastry, and Symphony they are static.

**Mapping of resources:** Mapping of resources (data items) is done similarly to mapping peers. Usually it is done by hashing a data key, e.g., the filename, with SHA-1 (Chord, Pastry, Freenet). While this implicitly distributes the assigned identifiers uniformly in the identifier space and thus provides a simple load-balancing mechanism, it destroys the semantics of keys, e.g., their application-specific clustering, which can be exploited to provide efficient data access. To prevent this, P-Grid, for example, uses a prefix-preserving hash function, i.e., $u < v \Rightarrow h(u) < h(v)$. This has advantages in query processing but requires an additional and more complex load-balancing strategy. It is crucial that this mapping of resources is deterministic, static, and globally known.

### 3.5.3. Management of Identifier Space

In P-Grid each peer is responsible for resource identifiers that share the largest common prefix with the peer's identifier, i.e., a resource identifier is managed by the peer with the closest identifier in terms of P-Grid's distance function. For example, peer "0011" is responsible for resource identifier "001110101", if no peer with a longer common prefix exists. The situation in Freenet is very similar: Each peer is responsible for the resource identifiers which are numerically closest to one of the peer's elements in its vector identifier. Also in Pastry, and Symphony a similar condition applies. Data items are managed by the peer with the closest identifier. For example, identifier "2A83" will be managed by peer "2A84" if no peers "2A82" and "2A83" exist. As Chord's and DKS's identifier spaces are asymmetric, the situation is slightly different. A peer is responsible for all identifiers in the interval between its own identifier and the identifier of its predecessor on the ring. In all these approaches the responsibility of a peer may dynamically change due to arrivals or departures of peers in the overlay.

### 3.5.4. Graph Embedding

It has already been shown that peers cooperating in Freenet evolve the graph into a Small-World graph. For logarithmic-style overlay approaches, [Girdzijauskas *et al.* 2005] shows that these approaches form graphs according to Kleinberg's Small-World principles [Kleinberg 2000].

It is proven that such graphs belong to the special class of "routing efficient" Small-World networks where decentralized, greedy search algorithms provide the best performance. Therefore, conceptually all of these approaches build similar Small-World graphs with certain constraints for each case. E.g., Symphony by its nature of construction forms a Small-World graph. In the other logarithmic-style overlay cases each peer $u$ views the identifier space as partitioned in $\log(N)$ partitions where each partition is $b$ times bigger than the previous one ($b$ is the radix of the identifier alphabet). The routing table of $u$ in such systems contains $\log_b(N)$ links to some nodes from each partition. In Chord's case the chosen node will be the one with the smallest identifier of the given partition, while Pastry and P-Grid use any random node in the partition, which is a more relaxed constraint.

## 3.6. Conclusions

Based on a stringent analysis of current overlay networks, we discussed and formally described the key design aspects in the domain of overlay networks. We used our assessments to define a reference architecture for overlay networks specifically addressing API and interoperability aspects. This is the first reference architecture for overlay networks which includes a formal codification and definition of all design aspects. To validate the correctness and general applicability of our approach we applied it to model a representative set of overlay networks.

The reference architecture presented in this chapter, establishes a standardized vocabulary and facilitates the assessment of properties of overlays for qualitative comparison and can serve as the basis for the definition of a standardized API. In the next part of the thesis we will present three peer-to-peer systems, which were developed based on the vocabulary established in this chapter.

# Part II

# Small-World Inspired Overlays

# Chapter 4

# On Small World Graphs in Non-uniformly Distributed Key Spaces

The greatest challenge to any thinker is stating the problem in a way that will allow a solution.

Bertrand Russell

## 4.1. Overview

Since the appearance of the first generation of structured P2P systems like Chord [Stoica *et al.* 2001] and P-Grid [Aberer *et al.* 2005b] the number of proposed solutions for structured P2P overlay networks has been growing rapidly, and it became somewhat difficult to qualitatively and quantitatively compare them. Still, it was not hard to notice that many of the proposed solutions share similar properties and were structured in a comparable way. Among the wide range of proposed structured P2P overlay networks a majority can be characterized as *logarithmic-style P2P overlay networks* which, although being different in their maintenance algorithms, share the same structural properties and search algorithm characteristics. E.g. the expected search cost in P2P systems such as balanced P-Grid [Aberer *et al.* 2005b], Chord [Stoica *et al.* 2001], Pastry [Rowstron and Druschel 2001], Tapestry [Zhao *et al.* 2004] or Kademlia [Maymounkov and Mazières 2002] is $O(\log N)$ and the peers in each of them maintain on average $O(\log N)$ entries in their routing tables, where $N$ is the size of the network.

In this chapter we provide one way to better understand the common characteristics of these systems by relating them to the seminal work on Small-World graph construction introduced by Kleinberg [Kleinberg 2000]. This applies in particular for randomized overlay network structures [Gummadi *et al.* 2003] and randomized variants of deterministic structures such as randomized Chord [Manku 2003; Zhang *et al.* 2003]. The first contribution of this chapter is to clarify the relationship among logarithmic structured overlays and Kleinberg's model. We introduce a modified version of Kleinberg's model where the out-degree of the overlay graph is logarithmic instead of constant. This not only provides a better insight into the nature of existing logarithmic-style overlay networks, but also a foundation to develop less constrained overlay network structures and to trade-off between search and maintenance cost by choosing the routing table sizes flexibly by varying from constant to logarithmic size.

Furthermore, within the same framework we address a problem that is receiving recently increasing interest in many data-oriented P2P applications. In this type of applications it is important to preserve semantic relationships among resource keys, such as ordering or proximity, to allow semantic data processing, such as complex queries or information retrieval. This implies that the construction of (efficient) overlay networks has to support the case of non-uniformly distributed resource keys while exhibiting good load-balancing properties. Examples of overlay networks that have been proposed to address non-uniform key distributions are CAN [Ratnasamy *et al.* 2001], P-Grid [Aberer *et al.* 2005b] and Mercury [Bharambe *et al.* 2004]. CAN and P-Grid can partition the key-space upto any granularity, such that each partition has a balanced number of keys assigned to them. In doing so each of the CAN and P-Grid overlay networks sacrifice some desirable properties. Search efficiency in terms of the number of overlay hops can't be guaranteed in CAN for arbitrary partitioning of the key-space (zones). In contrast, P-Grid's randomization helps retaining routing efficiency [Aberer *et al.* 2005a], however, peers require more than logarithmic routing states. Mercury [Bharambe *et al.* 2004] uses heuristics to deal with the presence of skewed key-spaces and utilizes Small-World connectivity. We provide a formalized theoretical framework that covers the whole class of "routing efficient" Small-World networks for skewed key-spaces, including Mercury's heuristics. We prove that in such an overlay network both routing latency and the number of routing states per peer stay $O(\log N)$ independent of the skew of the key-space partition.

By proposing the extension to Kleinberg's model we are providing a foundation for a novel type of structured overlay networks that supports load balancing for unbalanced key and workload distributions, tradeoffs routing table size with search cost, and is expected to be robust due to use of randomization. We will show later in Chapter 5 how such model can be realized in practice by presenting Oscar overlay building principles.

## 4.2. Background

As it was mentioned in the Introduction, the investigation of Small-World phenomenon in social networks started in the sixties [Milgram 1967]. Since then there were numerous works and proposals to explain and model Small-World graphs. One approach for building Small-World graphs was presented by Watts and Strogatz [Watts and Strogatz 1998]. The idea was to randomly rewire a regular graph. Starting from a regular graph with constant out-degree, with probability parameter $p \in [0..1]$ at each node an edge is re-linked to another randomly chosen node. With the parameter $p = 1$, one obtains a completely random graph and with $p = 0$, the graph remains regular. When the probability $p$ is between 0 and 1, one obtains a wide range of Small-World graphs, that have properties of both regular and random graphs: high clustering coefficient and low diameter. Kleinberg proved [Kleinberg 2000] that among that wide range of Small-World graphs, there exists only one class of Small-World graphs in which decentralized (greedy) routing is most efficient. Kleinberg proposed different algorithms for constructing Small-World graphs. The idea is to rewire the links to other nodes not uniformly at random, but depending on the distance to the other node.

In Kleinberg's model nodes populate a regular $k$-dimensional lattice and each node has a *neighboring link* to the neighboring nodes that are a unit distance away from the given node. Each node also has a constant number of *long-range links* that are chosen among the whole set of nodes. Node $u$ chooses to have a long-range link to $v$ with probability inversely proportional to $d(u,v)^r$, where $d(u,v)$ is the distance between these nodes and $r$ is a structural parameter. It was proven that to construct "routing-efficient" Small-World graph (where greedy distance minimizing routing will perform best) is possible iff the structural parameter $r$ is equal to the space dimension.

Several other works employ various Small-World properties for building P2P systems, e.g., Symphony [Manku *et al.* 2003], Mercury [Bharambe *et al.* 2004] or SWAM [Banaei-Kashani and Shahabi 2004] to name a few. There are other works in the area trying to extend Kleinberg's model and to use his ideas to improve the performance of P2P networks, like [Franceschetti and Meester 2006; Zhang *et al.* 2002].

### 4.2.1. Notations and Definitions

In the following we will introduce a variation of Kleinberg's model which shows that the properties of standard logarithmic cost overlay networks, i.e., logarithmic cost of routing (in terms of overlay hops) with logarithmic size routing tables, can be achieved under much weaker assumptions than usually made. Since many existing DHT proposals are based on one-dimensional key spaces (e.g., Chord, P-Grid, Pastry), we will give the result for this case, and more precisely for the case of an interval topology. Analogous result can be given for other topologies, in particular the ring topology. Unlike as in Kleinberg's proof, we relax our assumptions such that we do not need peers to be connected in a grid, but only that they are randomly distributed in the space according to some probability density function $f$.

Before introducing the model we will recall some notations and definitions used in this chapter:

- $G$: the graph representing P2P overlay network

- $N$: number of nodes (peers) in the P2P system (the graph $G$)

- $\mathcal{I}$: identifier (key) space where nodes (peers) are populated
- $F_{\mathcal{P}}(p)$: identifier (key) of node $p$
- $f$: the probability density function setting the manner of how identifiers of the nodes (peers) are distributed in $\mathcal{I}$

## 4.3. Extended Kleinberg's Small-World model for Uniform Key Distribution and Logarithmic Out-degree

First we extend Kleinberg's Small-World model for Uniform Key Distribution, i.e., $f = const$. We model a P2P overlay network as a directed graph $G = (P, E)$ with $N$ nodes. Each peer of a P2P overlay network corresponds to a node in the graph and the routing table entries of this peer correspond to the outgoing edges from that node[1]. The nodes are embedded into the key-space $\mathcal{I}$ by uniformly randomly distributing them on the unit interval $[0; 1)$ such that each node $u$ obtains an unique identifier $F_{\mathcal{P}}(u) \in [0; 1)$. The distance among two nodes $u$ and $v$ is given as

$$d(u, v) = |F_{\mathcal{P}}(v) - F_{\mathcal{P}}(u)|. \tag{4.1}$$

The edges $E$ of the graph $G$ can be classified into *neighboring edges* $NE$ and *long-range edges* $LE$. Each node $u$ has two neighboring edges: one to the left neighbor and one to the right neighbor. This condition makes $G$ always connected. Different to Kleinberg's model we assume that a node has $\log_2 N$ long-range edges (instead of a constant number of long-range edges). Node $u$ can have long-range edge to any node $v \in LE_u$ for which $|F_{\mathcal{P}}(v) - F_{\mathcal{P}}(u)| \geq \frac{1}{N}$ and $v$ is chosen such that

$$P[v \in LE_u] \propto \frac{1}{d(u, v)}.$$

With the condition $|F_{\mathcal{P}}(v) - F_{\mathcal{P}}(u)| \geq \frac{1}{N}$ we restrict the choice of long-range edges to nodes that are not too close. Routing in such an overlay network is based on *greedy distance minimizing routing.* In each step a node $u$ forwards a search request for a target key $t$ to the node with the minimal distance to the target node $t$ among all nodes reachable through an edge from $u$. We prove that under this model, the expected search cost in number of overlay hops is $O(\log_2 N)$ as in all logarithmic cost P2P overlay networks. The proof is structurally the same as for Kleinberg's model, however, the bounds have to be derived differently due to the changed model.

**Theorem 4.1.** *The expected routing cost in the graph built according to "Model for uniform key distribution" using greedy distance minimizing routing is $O(log_2 N)$.*

[1] We use the terms "peer" and "node" interchangeably.

**Proof.** The probability that a node $u$ chooses a node $v$ as a long range contact is $\frac{\frac{1}{d(u,v)}}{\sum_{v \in LE_u} \frac{1}{d(u,v)}}$. First we have to calculate the upper bound of $\sum_{v \in LE_u} \frac{1}{d(u,v)}$ for any node $u$. The sum can acquire its highest value when it is calculated for a node $u$ which is at the center of the key-space. Thus, if we measure the sum for $F_{\mathcal{P}}(u) = \frac{1}{2}$, it gives an upper-bound. The distance from $u$ to the closest node will be at least $d_u \geq \frac{1}{N}$. We can calculate expected mean of inverse distance values from the node $u$ to all the other nodes given probability density function $f(x)$ as $2\int_{d_h}^{\frac{1}{2}} \frac{1}{x} f(x)dx$. Because nodes are distributed uniformly $f(x) = 1$ and $\sum_{v \in LE_u} \frac{1}{d(u,v)}$ is upper-bounded by:

$$N2\int_{\frac{1}{N}}^{\frac{1}{2}} \frac{1}{x}dx = 2N \ln x|_{\frac{1}{N}}^{\frac{1}{2}} < 2N \ln N. \tag{4.2}$$

Hence, the probability that node $u$ will choose $v$ as one of its long-range links is at least

$$\frac{1}{d(u,v)2N \ln N}. \tag{4.3}$$

Let us view the key-space as $\log_2 N$ partitions $A_1, A_2, .., A_{\log_2 N}$, where each partition $A_j$ is populated by the nodes whose distance from the target node $t$ is $[2^{-\log_2 N + j - 1}; 2^{-\log_2 N + j})$. During greedy routing after a node forwards the search request to node $s$ we say that the message is at partition $A_j$ if the distance between the current message holder $s$ and the target $t$ is within the range $2^{-\log_2 N+j-1} \leq d(s,t) < 2^{-\log_2 N+j}$. We calculate the probability $P_{next}$ that the current message holder has at least one long-range link to some node $v$ in some partition $A_l$ where $l < j$, i.e., the current message holder can forward the message closer to the target at least by one partition. There are at least $2N2^{-\log_2 N+j-1}$ such nodes. The distance from the current message holder to the most distant node in the partition $A_l$ is at most $2^{-\log_2 N+j-1} + 2^{-\log_2 N+j} = 3 * 2^{-\log_2 N+j-1}$. Using (4.3) we can determine that the probability that node $u$ will choose some $v$ in $A_l$ as one of its long-range links is at least

$$\frac{2N2^{-\log_2 N+j-1}}{3 * 2^{-\log_2 N+j-1} \cdot 2N \ln N} = \frac{1}{3 \ln N}. \tag{4.4}$$

Since each node has $\log_2 N$ long-range links, $P_{next}$ is on expectation at least

$$P_{next} \geq \ 1 - \left(1 - \frac{1}{3 \ln N}\right)^{\log_2 N} > 1 - e^{-\frac{1}{3 \ln 2}} = c. \tag{4.5}$$

Thus, when each node has $\log_2 N$ long-range links the lower bound of the probability that the message will be forwarded closer to the target partition does not depend on $N$ and is a constant that we denote by $c$. The probability that the message will stay in the same partition when node $u$ forwards it to the next node is at most $P_{same} \leq 1 - c$.

Let us denote by $X_j$ the total number of hops and $EX_j$ as the expected total number of hops that greedy distance minimizing routing will make within the partition $A_j$ before jumping into some partition $A_l$ that is closer to the target $t$, i.e., $l < j$. If $N_{A_j}$ is the number of nodes in $A_j$ then we have

$$\begin{aligned} EX_j &= \sum_{i=0}^{N_{A_j}} iPr[X_j = i] < \sum_{i=0}^{\infty} i(P_{same})^i P_{next} \\ &= \sum_{i=0}^{\infty} i(1-c)^i c = \frac{1-c}{c}. \end{aligned} \tag{4.6}$$

There exist $\log_2 N$ partitions of the key-space and the expected number hops in each of them is less than $\frac{1-c}{c}$, so the expected total number of hops that the algorithm will need, including the long-range hops is at most $(\frac{1-c}{c} + 1)\log_2 N$. The expected number of nodes that algorithm will have to visit using neighboring edges from the partition $A_1$ to the target node $t$ is $N \int_0^{\frac{1}{N}} f(x)dx = 1$. Therefore, the total expected number of hops is $\frac{1}{c}\log_2 N + 1$, i.e., $O(\log_2 N)$.

*q.e.d.*

Note that a tighter bound can be derived by determining the expected number of long-range hops, and thus our derivation is gives a pessimistic upper-bound, which nonetheless suffice to prove that the expected cost is $O(\log_2 N)$.

### 4.3.1. Similarities with Logarithmic-Style Peer-2-Peer Overlays

Notice the fact that $EX_j$ is a small constant. This means that each logarithmic partition of the key-space is reached in a constant number of hops. This result can be explained by the fact, that such a Small-World graph possesses nice "probabilistic partitioning" properties which are also widely exploited in traditional logarithmic-style P2P overlays. Indeed, in traditional logarithmic-style P2P overlays each peer $u$ views the identifier space partitioned in $\log_2 N$ logarithmic partitions of identifier space where each partition is twice bigger than the previous one (or $k$ times bigger if we consider base $k$ logarithmic partitioning, e.g., in Pastry $k = 16$). The routing table of $u$ in such systems contains $\log_2 N$ links to some node from every partition. E.g., in Chord [Stoica *et al.* 2001] the chosen node will be with the smallest identifier of the given partition, in Pastry [Rowstron and Druschel 2001] and P-Grid [Aberer *et al.* 2005b] - any random node of the partition. While routing, the message in every next hop is being routed to a node which belongs to a partition, that is at least twice ($k$ times) smaller than the previous partition where the previous message holder (node) used to be. Therefore, we can imagine such a P2P network as a space where the message approaches the target with steps of exponentially decreasing in size.

Overlays based on a graph built according to the above mentioned variation of Kleinberg's model, will have a very similar topology and routing properties as logarithmic-style P2P overlays. Indeed we can partition the identifier space of any node $u$ into $\log_2 N$ partitions $A_1, A_2.., A_{\log_2 N}$, where $A_j$ consists of all nodes whose distance from $u$ is between $2^{-\log_2 N+j-1}$ and $2^{-\log_2 N+j}$ (every next partition is twice bigger as the previous one). It is interesting to observe, that in this case node $u$ has almost equal probabilities to choose the long-range neighbor from each of these partitions. Therefore, when each node chooses $\log_2 N$ long-range neighbors in the same way, they will be uniformly distributed among the partitions, whereas in logarithmic-style P2P overlays $\log_2 N$ neighbors would be chosen strictly from each partition. We can consider logarithmic-style P2P overlay topologies as one "special case" of Small-World topology with stronger restrictions. This provides insight into how our modification of Kleinberg's model relaxes existing logarithmic cost overlay networks where routing entries have to point to each logarithmic partition of the key-space. Hence the possibility to generalize and model the behavior of logarithmic-style P2P topologies from a Small-World model point of view.

The feature of "Kleinbergian" graphs to model logarithmic P2P topologies suggests a more flexible manner of maintaining the networks. One of the possibilities would be to maintain a variable number of entries in routing tables for a tradeoff of logarithmic to polylogarithmic search cost, an observation that was also made in Symphony [Manku *et al.* 2003]. It also implies that the networks built according to "Kleinbergian" style would be more robust and resistant to network churn. Even in the case of connectivity loss, the routing cost will be at worst poly-logarithmic given we have at least one long-range link and the neighboring links intact.

## 4.4. Extended Small-World Model for Skewed Key Distribution

The main purpose of P2P overlay networks is to distribute resources among peers, such that resources can be be efficiently located and the workload is distributed as uniformly as possible among peers. In most standard P2P overlay networks uniform workload distribution is achieved by applying randomized hashing functions, such as SHA-1, to resource identifiers such that the hashed identifiers are uniformly distributed in the key-space. Then by also uniformly distributing peers in the key-space an approximately uniform load distribution is achieved. However, in many data-oriented P2P applications it is important to preserve relationships among resource keys, such as ordering or proximity, to allow semantic data processing, such as complex queries or information retrieval. Thus, uniform key distribution cannot be assumed, and in order to achieve uniform workload peers will be distributed non-uniformly in the key-space. In addition, different resources might be associated with different workload patterns, e.g., query frequency, which require further adaptations in the distribution of the peers over the key-space.

In the following we show that the construction we introduced in the previous Section 4.3 can be extended to peers, distributed non-uniformly in the key-space, without loosing routing efficiency in terms of either the expected routing latency or the number of routing states per peer. This provides the theoretical foundation for developing a novel class of P2P overlay networks that are able to deal with non-uniform load distributions.

### 4.4.1. Model for Skewed Key Distribution

We assume that there exists a mechanism that assigns peers according to a non-uniform distribution in the key-space adapting to the load-distribution (e.g., storage), such that the balanced number of data objects are assigned to each peer, irrespectively of their distribution in the key-space. Several examples of such mechanisms have been recently discussed in the literature [Aberer *et al.* 2005a; Ganesan *et al.* 2004; Wang *et al.* 2004b]. Thus, each peer acquires its identifier according to a non-uniform probability density function $f$. In order to account for the non-uniform peer distribution peers have to choose their long-range neighbors in graph $G$ according to the following criterion: a peer $u$ chooses peer $v$ as long-range neighbor with a probability that is inversely proportional to the integral of probability density function between these two nodes, i.e.,

$$P[v \in LE_u] \propto \frac{1}{|\int_{F_{\mathcal{P}}(u)}^{F_{\mathcal{P}}(v)} f(x)dx|}. \tag{4.7}$$

As in the previous model we restrict the choice of long-range neighbors to the peers that are not too close, therefore, $v \in LE_u$ for which $|\int_{F_{\mathcal{P}}(u)}^{F_{\mathcal{P}}(v)} f(x)dx| \geq \frac{1}{N}$. Using these criterions we claim that routing in the resulting overlay network is as efficient as in the case of uniform (balanced) key distribution.

**Theorem 4.2.** *The expected routing cost in the graph built according to the "Model for skewed key distribution" using greedy distance minimizing routing is $O(log_2 N)$.*

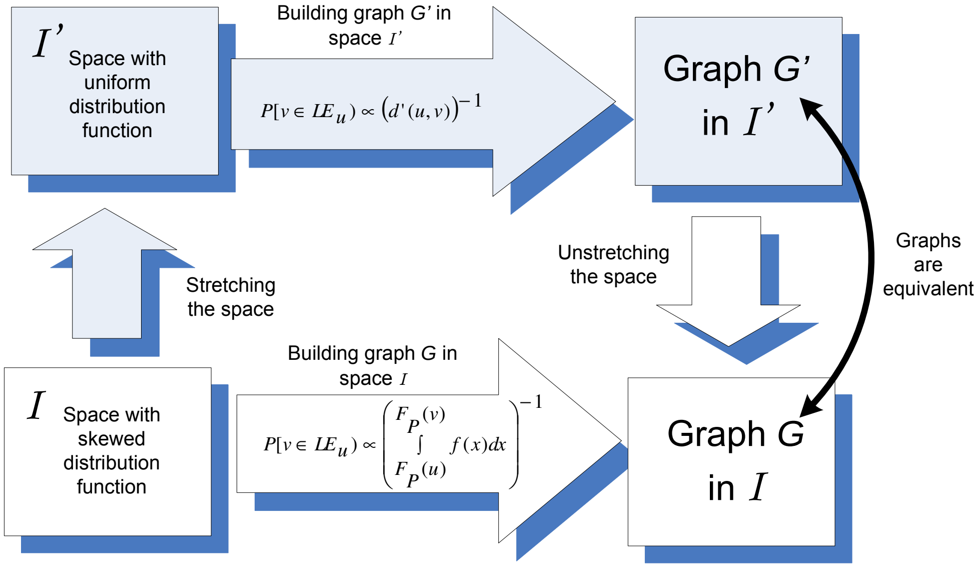

$I'$ Space with uniform distribution function
Building graph $G'$ in space $I'$
$P[v \in LE_u) \propto (d'(u,v))^{-1}$
Graph $G'$ in $I'$
Stretching the space
Unstretching the space
Graphs are equivalent
$I$ Space with skewed distribution function
Building graph $G$ in space $I$
$P[v \in LE_u) \propto \left(\int_{F_P(u)}^{F_P(v)} f(x)dx\right)^{-1}$
Graph $G$ in $I$

**Figure 4.1.** Space normalization

**Proof.** We have to show that by using the modified selection criterion for long-range links we are constructing a routing-efficient graph $G$. The schema of the proof is depicted in

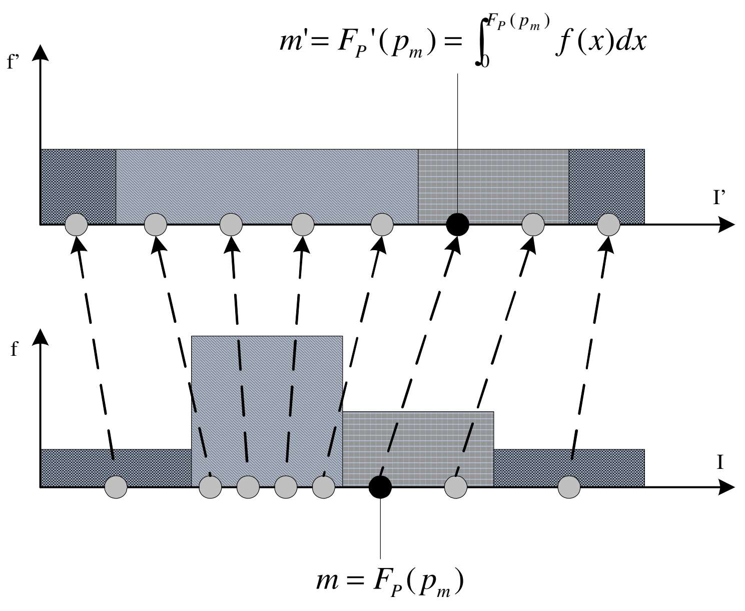


**Figure 4.2.** Mapping of nodes from $\mathcal{I}$ to $\mathcal{I}'$

Figure 4.1. The idea underlying our model is to normalize the space $\mathcal{I}$ in such a way that the normalized space $\mathcal{I}'$ will have a uniform probability density function $f'$. Any node $u$ with identifier $F_{\mathcal{P}}(u)$ in the space $\mathcal{I}$ will have a corresponding identifier $F'_{\mathcal{P}}(u)$ in the space $\mathcal{I}'$. The value of identifier $F'_{\mathcal{P}}(u)$ is chosen as $F'_{\mathcal{P}}(u) = \int_0^{F_{\mathcal{P}}(u)} f(x)dx$, such that

$$\int_0^{F_{\mathcal{P}}(u)} f(x)dx = \int_0^{F'_{\mathcal{P}}(u)} f'(x)dx$$

and peer identifiers are uniformly distributed in $\mathcal{I}'$. The distance between two nodes $u$ and $v$ in the space $\mathcal{I}'$ can be represented as

$$d'(F'_{\mathcal{P}}(u), F'_{\mathcal{P}}(v)) = |\int_{F_{\mathcal{P}}(u)}^{F_{\mathcal{P}}(v)} f(x)dx|. \tag{4.8}$$

As described in the previous section we already know how to construct a “routing-efficient” graph, i.e., choosing long-range links proportional to $\frac{1}{d'(F'_{\mathcal{P}}(u), F'_{\mathcal{P}}(v))}$. As we have already proven, the resulting graph $G'$ will be “routing-efficient”, i.e., the expected search cost using greedy distance minimizing routing will be $O(\log_2 N)$.

As shown in Figure 4.2 using the original criterion for selecting long-range links for uniform key distribution in space $\mathcal{I}'$, i.e., inverse proportional to $d'(F'_{\mathcal{P}}(u), F'_{\mathcal{P}}(v))$, is equivalent to choosing long-range links directly in space $\mathcal{I}$ using the modified criterion, i.e., inverse proportional to $\int_{F_{\mathcal{P}}(u)}^{F_{\mathcal{P}}(v)} f(x)dx$. The resulting graph $G$ in the original space $\mathcal{I}$ will have the same connectivity as the graph $G'$ constructed in space $\mathcal{I}'$, although the peers have different identifiers. The search efficiency depends only on the connectivity of the graph, therefore, the resulting graph $G$ will have the same search efficiency as graph $G'$, i.e., $O(\log_2 N)$. *q.e.d.*

### 4.4.2. Building Efficient Structured Overlay for Non-uniformly Distributed Key Spaces

The adaptation of our model in practice is straight-forward in the case where each peer in the P2P network knows the global key distribution, i.e., the probability density function $f$. In such a case the following network construction model can be applied.

While joining the network, some peer $u$ generates a value according to probability density function $f$ and assigns it as its identifier. The peer $u$ contacts any known peer and issues a query with that identifier. When $u$ gets an answer from some peer $v$ (in this case $v$ has the closest identifier to $u$), $u$ announces to $v$ that it will become its immediate neighbor. Both $u$ and $v$ correct in their routing tables of the immediate neighboring links.

Since the peer $u$ knows the function $f$ it can calculate the pdf $h_u$ that satisfies (5.1). The peer $u$ draws $\log_2 N$ random values according to $h_u$ and queries for these values. The peers that respond are added to $u$'s routing table as long-range neighbors. In such a way the peer $u$ completely joins the network.

The task however, is more complicated for a more realistic situation, where peers do not have information of the distribution $f$ and have to acquire it locally, by interacting with other peers. Moreover, the distribution $f$ may vary over time, further complicating the design of a practical system. In such a case, at each peer an iterative process of revising its routing table according to the current knowledge on $f$ has to be employed. The above mentioned steps have to be repeated whenever a peer obtains more precise information about $f$. Such iterative process can be performed indefinitely if the function $f$ changes over time in the system. In this way the topology would be always self-adjusted to the current conditions of the system.

In the next chapter we will investigate in more detail what efforts are needed and what operations have to be performed at every peer to be able to predict locally a good enough approximation of the probability density function $f$ for the construction of "routing-efficient" P2P networks.

## 4.5. Conclusions

The work of Kleinberg on Small-World graphs caused a stir in P2P community. It boosted the research towards investigating randomized topologies [Manku 2003; Manku *et al.* 2004] and even resulted in new proposals such as Symphony [Manku *et al.* 2003] and Mercury [Bharambe *et al.* 2004]. We used Kleinberg's model to provide a perspective on existing standard P2P overlay networks and to explain their nature. In this chapter we introduced two variants of Kleinberg's model which allow to model a large class of P2P overlay networks. The first model for uniform key distribution and logarithmic out-degree allows us to better understand the behavior of logarithmic-style P2P overlay networks. The flexibility of Kleinberg's model demonstrates the possibility of making flexible logarithmic P2P topologies by allowing them to change routing table size from constant to logarithmic. With our second model we showed that with Kleinberg's principle of building "routing-efficient" networks we can build P2P topologies for skewed key distributions. With such model we are setting a reference base for P2P systems that need to support uneven key distributions. In the following chapter we will show how the ideas behind this model can be implemented in practice.

# Chapter 5

# Oscar: Structured Overlay For Heterogeneous Environments

I was working on the proof of one of my poems all the morning, and took out a comma. In the afternoon I put it back again.

Oscar Wilde

## 5.1. Overview

The first generation of structured overlays realized distributed hash tables (DHTs) which are ill-suited for anything but exact queries. The need to support range queries necessitates systems which can handle uneven load distributions. However, such systems suffer from practical problems - including poor latency, disproportionate bandwidth usage at participating peers or unrealistic assumptions on peers' homogeneity, in terms of available storage or bandwidth resources.

Based on the theoretical work presented in the previous chapter, we consider a system which is capable not only of supporting uneven load distributions, but also of operating in heterogeneous environments, where each peer can autonomously decide how much of its resources to contribute to the system. We provide the theoretical foundations of realizing such a network and present the Oscar system based on these principles. We show that Oscar can construct efficient overlays given arbitrary load distributions by employing a novel scalable network sampling technique. The simulations of our system validate the theory and evaluate Oscar's performance under typical challenges encountered in real-life large-scale networked systems, including participant heterogeneity, faults and skewed and dynamic load-distributions. Thus, the Oscar system presented in this chapter fills in an important gap in the family of structured overlays, bringing into life a practical internet-scale index, which can play a crucial role in enabling data-oriented applications distributed over wide-area networks.

This chapter is organized as follows. Section 5.2 motivates the need of structured overlays capable of supporting non-uniform key distributions and discusses the drawbacks of the existing solutions. In Section 5.3 we recapitulate the main ideas presented in the previous chapter, namely the problem of dealing with skewed key spaces. We present the concept and the algorithms of our proposed system in Section 5.4. In Section 5.5 we give the theoretical analysis of Oscar and in Section 5.6 we validate the analysis and evaluate the performance of our proposed approach in the simulation environment. We conclude the chapter in Section 5.7.

## 5.2. Motivation

DHTs like Chord or Pastry, and later works (e.g. [Hui *et al.* October, 2006; Manku *et al.* 2003]) which followed the same paradigm, provide only limited support for data-oriented applications. Essentially these approaches rely on uniform hashing of resource identifiers to achieve load-balancing properties for efficient operations. Thus, these systems support only exact match queries. This spurred research on a next generation of order-preserving overlay networks. Preserving ordering relationships among keys is essential for data-oriented queries like range, similarity and sky-line queries. With data distributions as they occur in practice, these systems face load-balancing problems with respect to the amount of resources individual peers have to manage. This led to a number of research efforts addressing this problem in various ways. Systems like CAN [Ratnasamy *et al.* 2001], Mercury [Bharambe *et al.* 2004], P-Grid [Aberer *et al.* 2005b], skip graphs [Aspnes and Shah 2003; Harvey *et al.* March 2003] and derivatives [Aspnes *et al.* 2004; Ganesan *et al.* 2004; Guerraoui *et al.* 2006] looked into some sub-problems, like addressing load-balance under non-uniform key distributions. However, these approaches usually take advantage of uniformity assumptions on peers' capacity in terms of bandwidth consumption and storage capacity, which limits their practicality for realistic peer-to-peer environments. Also most of the approaches suffer from shortcomings with respect to essential properties of operation, e.g., the search efficiency in terms of the number of overlay hops cannot be guaranteed in CAN for an arbitrary partitioning of the key-space (zones). Storage-load balanced P-Grid may have highly imbalanced peer degrees. Skip graphs need to have $O(\log N)$ level rings at each peer where level ring neighbors are determined by a peer's membership vector and the existing skew in the system. Such a design omits a possibility to choose routing table entries in a randomized manner. Therefore, Skip graphs lack the flexibility which is provided by the truly randomized approaches (e.g., based on Small-World construction principles like Mercury) and cannot address some of the heterogeneity issues, e.g., different constraints on storage and bandwidth at each peer. To account for the heterogeneity in the system and to balance the storage load among the peers the *virtual peer* concept can be employed in the overlay network (e.g., [Dabek *et al.* 2001; Stoica *et al.* 2001]). However, virtual peers do not completely solve the problem, since the routing table size of any physical peer might explode and start growing linearly with the number of virtual peers, regardless the bandwidth capacity of that physical peer.

Small-world construction principles [Kleinberg 2000] do not restrain a peer from determining the size of the key-space based on both storage and bandwidth constraints, or having limited amount of routing links. The in and out-degrees of a peer can vary depending on peers' local decision, still providing guarantees of efficient search globally. Because the links are chosen randomly the in-degree of a peer can also be easily adjusted, where individual peers refuse further connections based on a local decision. Such features of Small-World approaches enable accommodating and exploiting peer heterogeneity - storage as well as bandwidth. Peers are free to choose the maximum amount of outgoing and incoming links locally, depending on their bandwidth budget to maintain the links as well as to cater to the query traffic, based on their locally perceived bandwidth or other constraints. Similarly, peers are free to choose the key-space to be responsible for. This may be based on their storage capacity and bandwidth constraint to answer the corresponding queries.

As discussed in the introduction of the thesis, from a more general perspective most structured overlay networks can be seen as special cases of a small world graph [Kleinberg 2000] embedded into the key space used for resource identification. In such networks searches are usually performed by greedy routing, however, other navigation techniques can be implemented, like routing with "lookahead" [Manku *et al.* 2004] or "cautious greedy-routing" [Barbella *et al.* 2007]. In the previous chapter it has been shown that a network embedding into the key space can be always performed in a way that peers adapt to skews in the key distribution by proportionally partitioning the key space, while retaining the small world graph structure and thus efficient routing. Mercury [Bharambe *et al.* 2004] based its approach on the same observation and proposed a structured overlay network which load-balances among peers and attempts to construct a Small-World graph based overlay. For properly choosing long-range links Mercury needs to know about the key distribution, for which a simple sampling procedure is employed. However, the sampling technique that Mercury uses to determine the candidates for long-range links does not scale given complex distributions which actually occur in practice. Mercury can deal with *simple* monotonous skewed distributions but the sampling technique is inadequate for *real-world* distributions which are typically highly complex (cf. Section 5.3). In our initial work on the Oscar system [Girdzijauskas *et al.* 2006] we have shown that in certain cases which occur frequently in reality, Mercury nodes suffer large imbalance in in-degree, which results in poor search performance and unnecessary congestion. If the originally occurring in-degree imbalance is prevented by local restrictions at peers, naturally there is a further deterioration of performance. Differently said, the lower the in-degree imbalance without imposing local restrictions, the lesser the performance deterioration when in-degree restrictions are additionally imposed. In general, the problem with most of the current state-of-the-art approaches dealing with non-uniform key distributions is the resulting node degree imbalance.

The aforementioned problem with Mercury arises because it uses an *approximation* of the global key distribution from a limited set of uniform samples for constructing the network.

Since it is not possible to correctly approximate a distribution with a limited number of samples if it is highly complex, the resulting P2P networks will have poor performance. Therefore, in this chapter we suggest the algorithms for constructing a routing efficient overlay network based on scalable sampling strategy. We call the resulting system *Oscar*, which stands for *o*verlay network built using *sca*lable sampling of *r*ealistic distributions. To our knowledge this is the first scalable sampling approach that has been proposed for construction and maintenance of structured overlay networks for non-uniform key distributions while coping with and exploiting peer heterogeneity. The Oscar overlay enjoys all the benefits of systems like P-Grid or Mercury which support complex non-uniform key distributions and hence non-exact queries (e.g., range or similarity queries) but does not suffer from node in-degree imbalance, while exhibiting comparable lookup performance. We provide a full analysis for the Oscar algorithms with theoretical guarantees for the performance of the resulting networks. The theoretical results are validated with simulations with realistic skewed workloads, heterogeneous peers and network dynamics (churn). Furthermore, we show that the Oscar overlay is capable of dealing with both the heterogeneity observed in the internet, particularly bandwidth and storage resource heterogeneity at peers, as well as non-uniformity observed in data-oriented applications, particularly skewed key distributions as well as skewed access loads. Such skews lead to disproportionate usage of storage and bandwidth at certain peers. Here we explicitly address the issue of realistic skewed key distributions to build the Oscar overlay network potentially comprising heterogeneous peers.

## 5.3. Preliminaries

Before going into the details of our work, we briefly recapitulate the basic concepts and notations introduced in Chapter 3.

**Basic concepts for structured P2P**. A structured overlay network consists of set of peers $\mathcal{P}$ ($N = |\mathcal{P}|$) and set of resources $\mathcal{R}$. There exists an identifier space $\mathcal{I}$ (usually on the unit interval $\mathcal{I} \in [0..1)$, e.g., Chord [Stoica *et al.* 2001], Symphony [Manku *et al.* 2003]) and two mapping functions $F_{\mathcal{P}} : \mathcal{P} \to \mathcal{I}$ and $F_{\mathcal{R}} : \mathcal{R} \to \mathcal{I}$ (e.g., SHA-1). Thus, each peer $p \in \mathcal{P}$ and each resource $r \in \mathcal{R}$ is associated with some identifiers $F_{\mathcal{P}}(p) \in \mathcal{I}$ and $F_{\mathcal{R}}(r) \in \mathcal{I}$, respectively. There exists a distance function $d_{\mathcal{I}}(u, v)$ which indicates the distance between a peer $u$ and a peer $v$ in $\mathcal{I}$. Each peer $p$ has some short-range links $\rho_s(p) \subset \mathcal{P}$ and long-range links $\rho_l(p) \subset \mathcal{P}$ which form a peer's routing table $\rho(p) = \rho_s(p) \cup \rho_l(p)$. There exists a global probability density function $f$ characterizing how peer identifiers are distributed in $\mathcal{I}$. Any resource $r \in \mathcal{R}$ in the P2P system can be located by issuing a query for $F_{\mathcal{R}}(r)$. In structured P2P systems queries are usually routed in a greedy fashion, i.e., always choosing the link $\rho \in \rho(p)$ which minimizes the distance to the target's identifier.

**Complex Distributions.** Using an uniform hash function $F_{\mathcal{R}}$ (e.g., SHA-1) in data-oriented P2P applications is not adequate. It is necessary to deal with order preserving hash

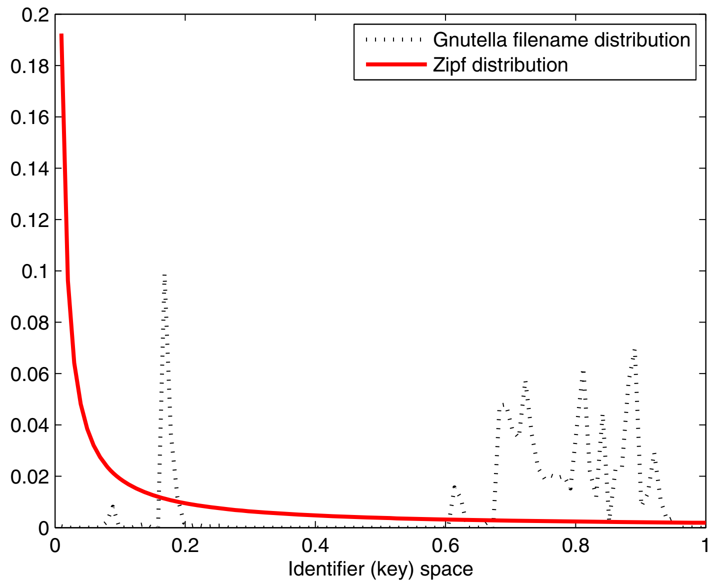


**Figure 5.1.** Probability density functions: *Monotonous Zipf (solid) vs Gnutella filename(dotted)*

functions, which in turn produce highly skewed key distributions.

Let us assume a data-oriented P2P system where the resources are identified and looked-up by filenames. A widely used technique in P2P systems is the following: each peer $p$ and each resource $r$ have identifiers $F_{\mathcal{P}}(p)$ and $F_{\mathcal{R}}(r)$ on a 1-dimensional ring $\mathcal{I} \in [0..1)$. Each peer is responsible for all the resources which map to the identifier range $D(p) \in \Big[F_{\mathcal{P}}(p), F_{\mathcal{P}}(p_{succ})\Big)$, where the peer $p_{succ}$ is the successor of the peer $p$ on the identifier ring $\mathcal{I}$. We cannot use a uniform hash function such as SHA-1, since we want the function $F_{\mathcal{R}}$ to preserve ordering relationships between the resource keys (e.g., enabling the straightforward use of range queries), i.e., $F_{\mathcal{R}}(r_i) > F_{\mathcal{R}}(r_j)$ iff $r_i > r_j$. Such an order preserving hash function will lead to a very skewed distribution of resource identifiers over the identifier ring $\mathcal{I}$. For example, in Figure 5.1 we can see a distribution function of filename identifiers in $\mathcal{I}$ extracted from a Gnutella trace (dotted line) of 20'000 filenames crawled in 2002. Despite the complex skew of key distributions, we would like that each peer $p$ is responsible for a "fair" (or equal) amount of resources and be storage-load balanced, i.e., $|\mathcal{R}_{p_i}| \approx |\mathcal{R}_{p_j}|$ for any $i$ and $j$, where $\mathcal{R}_p \subset \mathcal{R}$ and $\forall r \in \mathcal{R}_p$ $F_{\mathcal{R}}(r) \in D(p)$. In this case the peer identifiers will have to reflect the distribution of resource identifiers. Hence the peer identifier distribution will have a similar shape as the resource identifier distribution. Since in general the resource identifier distributions are usually non-uniform and exhibit complex skews, the resulting peer identifier distribution will have to have a complex skew as well.

**Dealing with skewed spaces.** The seminal work of Kleinberg [Kleinberg 2000] proposes "routing efficient" network on a uniform $d$-dimensional mesh. The follow up works [Barrière *et al.* 2001; Manku *et al.* 2003] showed how to adopt the *Kleinbergian* network construction principles for P2P systems with uniform key distribution. In the previous chapter we have shown that it is indeed possible to construct "routing efficient" Small-World network in the 1-dimensional space even if the peers are non-uniformly distributed on the unit interval. For doing so it was shown that a peer $u$ has to choose a peer $v$ as its long-range neighbor with a probability that is inversely proportional to the integral of the probability density function $f$ between these two nodes, i.e.,

$$P[v \in \rho_l(u)] \propto \frac{1}{|\int_{F_{\mathcal{P}}(u)}^{F_{\mathcal{P}}(v)} f(x)dx|}. \tag{5.1}$$

However, it is non-trivial to apply this technique in practice because it requires at each peer the global knowledge about the data load in the system, hence the key distribution $f$. A simple approach to obtain the distribution is to randomly sample the network and get an approximation of the key distribution, e.g., Mercury [Bharambe *et al.* 2004]. However, the real-world distributions can be totally arbitrary and the only sufficient approximation of the distribution would be gathering in a sample set the complete set of values which, of course, does not scale. In this chapter we show that Mercury (which uses random sampling) fails to build routing efficient networks given arbitrary distribution functions. Moreover, we also show that it is not necessary to know the distribution function over the entire identifier space with uniform "resolution" – it is sufficient to "learn" well the distribution for only some regions of the identifier space while leaving other regions vaguely explored, making it the base idea of Oscar algorithms.

## 5.4. Oscar Overlay

**The Insight.** According to the continuous Kleinberg's approach [Barrière *et al.* 2001; Girdzijauskas *et al.* 2005; Manku *et al.* 2003] for construction of a "routing efficient" network in 1-dimensional space, each node $u$ has to choose two short-range neighbors and one or more long-range neighbors. Short-range neighbors of $u$ are its immediate successor and predecessor on the unit ring. A peer $u$ chooses its long-range neighbor $v$ in the unit interval with the pdf $g(x) = \frac{1}{x \ln N}$ on the range $[\frac{1}{N}, 1]$, where $x = d_{\mathcal{I}}(u, v)$. It has been proven that a network constructed in such a way is a "routing-efficient" network, where a greedy routing algorithm on expectation requires $O(\frac{\log_2^2 N}{l})$ hops, where $l$ is the number of long range links at every peer. This means that a node $u$ will tend to choose a long-range neighbor $v$ rather from its close neighborhood than from the farther away regions. The pdf $g(x)$ according to which the neighbors are chosen also has one nice property when partitioned into equally spaced segments on the logarithmic scale (logarithmic partitions). That is, if we partition the identifier space into $\log_2 N$ partitions $A_1, A_2, ..A_{\log_2 N}$, such that the distance between the peer $u$ and any other peer $v$ in $A_i$ is bounded by $2^{-i} \leq d_{\mathcal{I}}(u, v) < 2^{-i+1}$ the peer $v$ will have equal probability to be chosen from each of the resulting partitions (Figure 5.2). Indeed, the probability that $v$ will

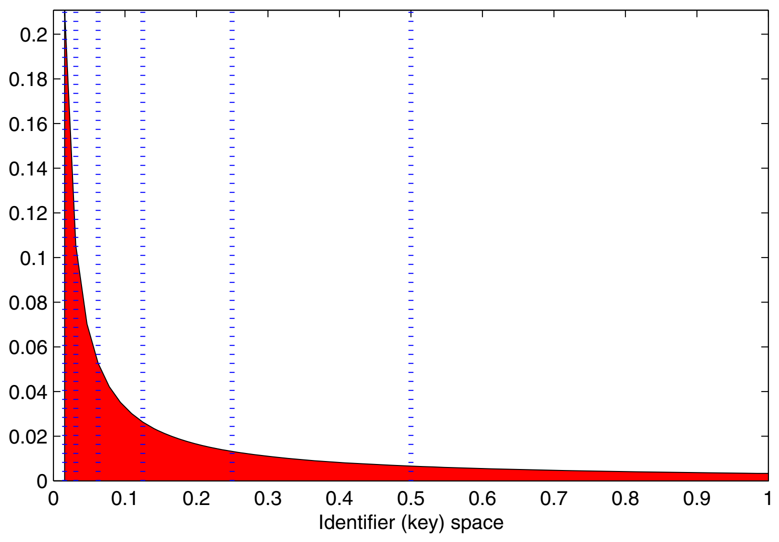


**Figure 5.2.** pdf $g(x)$ (solid bars) and the $A_1, A_2, .., A_{\log_2 N}$ partitions separated by the dotted lines

be chosen by $u$ in some interval $A_i$ is exactly $\frac{1}{\log_2 N}$ and does not depend on $i$:

$$P(F_{\mathcal{P}}(v) \in A_i) = \int_{2^{i-\log_2 N-1}}^{2^{i-\log_2 N}} \frac{1}{x \ln N} dx = \frac{1}{\log_2 N} \tag{5.2}$$

In practice choosing non-uniformly at random but according to some continuous pdf is complicated. Thus, equation 5.2 gives us an insight of how to modify a network construction algorithm, in which the neighbors will be chosen not directly by some continuous pdf $g(x)$, but uniformly at random in certain regions derived from $g(x)$. That is, if each peer $u$ first chooses uniformly at random one logarithmic partition and then within that partition uniformly at random one peer $v$ as a long-neighbor then none of the pdf characteristics will be violated and all the desirable properties of the "routing efficient" network will be preserved. Of course, this approach perfectly fits the case with uniform key-distributions. In such cases the partitions can be recalculated in advance at each peer. However, when assuming skewed key-spaces, it is not straightforward how to define logarithmic partitions, hence how to choose the long-range link.

### 5.4.1. Space Partitioning

In the case of uniformly distributed peer identifiers the expected number of peers within some range of length $d$ is actually equal to $d \cdot N$ assuming a unit length identifier space. Thus, the division of such an identifier space into a logarithmic (base-2) number of partitions is nothing else but recursively halving the peer population. That means a peer $u$ with identifier 0 ($F_{\mathcal{P}}(u) = 0$) will define the partition $A_1$ which will contain half of the peer population, i.e., all the peers which identifiers are bigger than $\frac{1}{2}$, $A_2$ – all the peers which identifiers are bigger than $\frac{1}{4}$ and smaller than $\frac{1}{2}$ and so on. This technique can be easily adapted to the case with any identifier skew, so Oscar uses this intuition in order to build its routing network in a simple and efficient manner. Instead of using the predefined borderlines between the logarithmic partitions we will use the median values of the exponentially decreasing peer populations. That is, an Oscar node $u$ with an identifier $u_{id}$ has to partition the identifier space into logarithmic partitions $A_1, A_2, ..A_{\log_2 N}$. Each border between neighboring partitions is determined by a median value of the peer identifiers in the exponentially decreasing subsets of peer population, i.e., the border between $A_1$ and $A_2$ will be the median $m_1$ of the peer identifiers from the whole peer population $\mathcal{P}$, the border between $A_2$ and $A_3$ will be the median $m_2$ of the identifiers from the subpopulation $\mathcal{P} \setminus A_1$ etc. In general the border value between $A_i$ and $A_{i+1}$ will be the median $m_i$ of peer identifiers from the subpopulation $\mathcal{P} \setminus B_i$, where $B_i = \cup_{j=1}^{i-1} A_j$. Ideally the first partition $A_1$ has to contain $\frac{1}{2}$ of the initial population, $A_2$ has to contain $\frac{1}{4}$ and so on. Since in practice it is not possible to exactly know the precise members of all the partitions, an Oscar node has to approximate the key range for each partition. For finding the median values an Oscar node has to uniformly sample each subpopulation $B_i$ and determine the current median $m_i$ from the acquired sample set. The random sampling technique proposed by Mercury (for sampling the whole population) is employed. To sample the subsets of the population $B_i$ the Oscar nodes use random walkers which do not visit nodes with identifiers that do not belong to the current population $B_i$. Our simulation experiments show that such a technique yields very good results in practice even with very low sample sizes.

### 5.4.2. Oscar Technique

Here we introduce the basis of Oscar's technique – the long-range link acquiring procedure: *each peer $u$ first chooses uniformly at random one logarithmic partition $A_i$ and then within that partition uniformly at random one peer $v$. This peer $v$ will become a long-range neighbor of $u$.* Thus, for successful building of an overlay we only require each peer to have a snapshot of the current key distribution by acquiring the knowledge of the positions and sizes of the corresponding partitions $A_1, A_2, ..A_{\log_2 N}$, i.e., a list of peer keys $F_P(p^{m_1}), F_P(p^{m_2}), .., F_P(p^{m_{\log_2 N}})$, where peers $p^{m_1}, p^{m_1}, .., p^{\log_2 N}$ represent the boundaries between the partitions. Such knowledge can be gained either by actively sampling the network (cf. Section 5.4.3) or by copying a snapshot of a "global view" from a ring neighbor (in this case the snapshot has to be adjusted accordingly by contacting the peers $p^{m_1}, p^{m_1}, .., p^{\log_2 N}$ and requesting the keys of their ring neighbors $F_P(p^{m_1}_{succ}), F_P(p^{m_2}_{succ}), .., F_P(p^{m_{\log_2 N}}_{succ})$, which are then included in the snapshot). Such copying incurs contacting only $O(\log_2 N)$ peers and is very suitable for the propagation of the latest knowledge of the key distribution.

The network remains correctly wired if the global key distribution remains stable over time. However, even if the key distribution is changing the peers can rewire only "on demand", i.e., when a network's performance starts to deteriorate. One of the indications that a network is not healthy is high in-degree [Girdzijauskas *et al.* 2006] of some peers. This is an indication that a majority of the peers which are pointing to an overloaded peer have an out-dated "global view" and wrong distribution estimation. Another indication for out-dated knowledge is an increased average routing time. Therefore, an overloaded peer can trigger the resampling and rewiring processes for itself and for its neighbors. Such actions bring the connectivity of the network to the optimal state. We show in our simulations that the Oscar sampling technique is not expensive and adapts the network well to changing id-space conditions. Moreover, in our analysis (cf. Section 5.5) we prove that the error within the partitions can be relatively large without inflicting considerable damage on the search efficiency, i.e., a network can support quite high variation in the distribution without actively sampling the network.

Since Oscar is an instance of randomized Small World networks – the number of long-range links in Oscar is not restricted and can be assigned individually according to the needs of a particular peer, as long as there exists at least one such link per peer. By allowing different in/out degree at each peer Oscar can easily adapt to heterogeneous environments (workload of a peer or local available resources) still guaranteeing efficient global search as long as there is at least one long-range link maintained per peer. Our simulations show (Section 5.6) that Oscar performs in heterogeneous environment as good as in homogeneous.

As for the correctness of the system, we rely on the already devised self-stabilizing algorithms (e.g., [Angluin *et al.* 2005; Ghodsi; Li *et al.* 2004; Liben-Nowell *et al.* 2002; Shaker and Reeves 2005; Stoica *et al.* 2001]) which maintain the virtual ring (short-range links) under churn. The establishment of short-range links ensures correctness of the greedy routing algorithm[1], while long-range links are specifc to different approaches and serve as "routing optimization" links. Thus, we will focus mainly on the algorithms for establishing long-range links.

### 5.4.3. Oscar Algorithms

Here we will formally describe the *Oscar* network construction and maintenance algorithms.

**The *join* algorithm.** In *Oscar*, as in many other P2P approaches, to join the network a peer $u$ has to know at least one peer already present in the system and to contact it. The joining peer is assigned with some identifier $F_{\mathcal{P}}(u)$ which is usually based on the distribution of the global data in the peer-to-peer system (see the discussion in Section 5.3) and depends on load balancing algorithms which are orthogonal to our work, e.g., in [Ganesan *et al.* 2004; Godfrey *et al.* 2004; Karger and Ruhl 2004; Rao *et al.* 2003]. Upon joining the network $u$ issues a query with its identifer $F_{\mathcal{P}}(u)$ and it inserts itself into the unit ring between the responsible peer for peer $u$'s key $F_{\mathcal{P}}(u)$ peer and its successor. Every peer keeps an estimated state of the global view as a set of pointers to the peers which mark the boundaries of the logarithmic partitions (cf. Subsection 5.4.1). A peer $u$ learns about the current key distribution in the network from its immediate neighbor $u_{successor}$ by copying its snapshot set of the "global

[1] establishment of short-range links results in a virtual ring topology in such a way ensuring the correctness of the greedy routing algorithm (a message *always* can be forwarded closer to a target)

view" pointers. Afterwards the peer $u$ establishes $l$ long-range links using the *longRangeLink* algorithm (Algorithm 5.3).

**Algorithm 5.1** Scalable sampling algorithm for learning the key distribution $scalableSampling(u)$

```
1: $\rho(u) = \emptyset; i = 0;$
2: $range = \Big[F_{\mathcal{P}}(u_{successor}); F_{\mathcal{P}}(u)\Big);$
3: $notEnoughPartitions = true;$
4: while $notEnoughPartitions$ do
5:    $i = i + 1; P_{sample} = \emptyset;$
6:    for j=1 to k do
7:       $P_{sample} = P_{sample} \cup randomBoundedWalk(u, range, TTL)$
8:    end for
9:    $m(i) = medianByF_{\mathcal{P}}(P_{sample})$
10:   if $m(i) = F_{\mathcal{P}}(u_{successor})$ then
11:      $notEnoughPartitions = false;$
12:   end if
13:   if i=1 then
14:      $partitionsStart(u, i) = m(i); partitionsEnd(u, i) = u;$
15:   else
16:      $partitionsStart(u, i) = m(i); partitionsEnd(u, i) = m(i - 1)$
17:   end if
18:   $range = \Big[F_{\mathcal{P}}(u_{successor}); F_{\mathcal{P}}(m(i))\Big);$
19: end while
```

**The *scalableSampling* algorithm.** In case a peer $u$ has an out-dated snapshot of the key distribution it can acquire an up-to-date one by using Oscar's scalable sampling technique. It has to determine $O(\log_2 N)$ logarithmic partitions (ranges) in the identifier space using the *scalableSampling* algorithm (Algorithm 5.1). To find the first partition the algorithm starts by issuing $k$ random walkers within the defined *range* of the identifer space using the *randomBoundedWalk* algorithm (Algorithm 5.2). Initially the defined *range* spans the whole identifier space starting from the identifier of peer $u$'s successor on the identifier ring up to the peer $u$'s identifier itself (Algorithm 5.1, line 2). After the collection of the random samples in the set $P_{sample}$ the peer $u$ finds the median value of all the peer identifiers of the set $P_{sample}$ (line 9). Having the median value the peer $u$ can define the first, furthest, partition $A_1$ which will span the identifier space from the found median value up to the peer $u$'s identifer value (line 14). The *range* value for performing the next random walk within the subgraph $\mathcal{P} \setminus A_1$ is reduced (line 18) and the algorithm continues by repeating the same steps (lines 4- 19) to find the successive partitions $A_2, A_3, ..$ etc. The algorithm stops finding the partitions when the median value is equal to the identifier of the $u$'s successor $F_{\mathcal{P}}(u_{successor})$ (line 10). In such a way the algorithm acquires on expectation $\log_2 N$ partitions.

**The *randomBoundedWalk* algorithm.** For successful usage of the *scalableSampling* algorithm it is necessary to be able to sample not only the whole population of peers $\mathcal{P}$ but also some subpopulation of peers $B$. Therefore, a specific random walk algorithm is needed.

**Algorithm 5.2** Bounded Random Walk Algorithm $[r] = randomBoundedWalk(u, range, TTL)$

```
1: if $TTL > 0$ and $R \neq \oslash$ then
2:     $TTL = TTL - 1$
3:     $R = \{p \in \rho(u) | F_{\mathcal{P}}(p) \in range\}$;
4:     $next = chooseRandomly(R)$
5:     $[r] = randomBoundedWalk(next, range, TTL)$
6: else
7:     $r = u$
8: end if
```

The algorithm will produce random walkers which would be able to "walk" only within a subpopulation of peers $B \subset \mathcal{P}$ restricted by a predefined scope variable *range*, such that $p_B \in B$ iff $F_{\mathcal{P}}(p_B) \in range$. Such a *randomBoundedWalk* algorithm (Algorithm 5.2) is a modified random walker, where the message is forwarded not to any random link of the current message holder $u$, but to a randomly selected link $p$, which satisfies the condition $F_{\mathcal{P}}(p) \in range$ (Algorithm 5.2, line 3). Such link will always exist assuming the underlying ring structure is in place.

**Algorithm 5.3** Long range link construction algorithm $[longRangeNeighbors] = longRangeLink(u, outdegree)$

```
1: for i=1 to outdegree do
2:     $randPartition = \Big[F_{\mathcal{P}}(partitionsStart(u, rand)); F_{\mathcal{P}}(partitionsEnd(u, rand))\Big)$;
3:     $[longRangeNeighbors(i)] = queryToRange(u, randPartition)$
4: end for
```

**The *longRangeLink* algorithm.** To assign the long-range link, the *longRangeLink* algorithm is used (Algorithm 5.3) which chooses uniformly at random one of the partitions $A_i$(line 2) and then assigns the random peer $v$ from that partition using the *queryToRange* algorithm (line 3). The *queryToRange* algorithm is a greedy routing algorithm, which minimizes distance to the given range $A_i$ and terminates whenever the first peer $v$ in that range is reached. Since the algorithm requires $k$ samples per each logarithmic partition, the expected number of needed samples per peer in total is $O(k \log N)$.

Note that the algorithm does not require knowledge or estimation of the total number of nodes in the network. The only place where in principle the estimation of $N$ is needed is the $TTL$ value of a random walk. As explained in [Bharambe *et al.* 2004] the $TTL$ should be set to a value of $\log_2 N$. However, the simulations show that it is sufficient to set the $TTL$ value equal to the number of previously determined partitions (initially, newly joined peers request the information on the partition size from their ring neighbors). In such a way the *Oscar* algorithms are designed to be independent of the estimation of the network size $N$.

Since churn exists in P2P networks and the peers join and leave the system dynamically each peer has to rewire its long range links from time to time. This can be done either periodically or adaptively. As discussed above we use adaptive techniques for performing the sampling algorithms if the key distribution in the network has changed considerably, and as an effect some peers got overloaded and the average routing cost has increased. In practice the sampling is performed rarely since, as we will show in the next section, Oscar partitioning is robust to imprecise measurements and distribution fluctuations. In such a way the Oscar system can self-optimize under dynamically changing network conditions.

In the next Section 5.5 we show that the Oscar overlay is robust to sampling errors and have logarithmic search performance given a logarithmic number of links per node.

## 5.5. Analysis

In this chapter we will consider the case of a skewed key-space $\mathcal{I}$ by "transforming" it to the uniform space $\mathcal{I}'$ and making all the necessary proofs in $\mathcal{I}'$ (Figure 5.3). Such a transformation of the problem is at the heart of using Kleinbergian Small-World principles for non-uniform keyspaces. Let us consider that we have the peer population $\mathcal{P}$ where each peer $p \in \mathcal{P}$ has an identifier $F_{\mathcal{P}}(p) \in \mathcal{I}$. Note that for each peer $p \in \mathcal{P}$ the identifier of a peer $p$ in the space $\mathcal{I}'$ is $F'_{\mathcal{P}}(p) = \int_0^{F_{\mathcal{P}}(p)} f(x)dx$. In such a way the identifier space $\mathcal{I}'$ will have uniformly distributed keys on the unit interval [0..1). The important fact here is that any median $m$ in $\mathcal{I}$ and the analogous median $m'$ of $\mathcal{I}'$ belongs to the same peer $p_m$ such that $m = F_{\mathcal{P}}(p_m)$ and $m' = F'_{\mathcal{P}}(p_m)$. Hence finding the median peer in the uniform space $\mathcal{I}'$ will be equivalent to finding it in the skewed space $\mathcal{I}$.

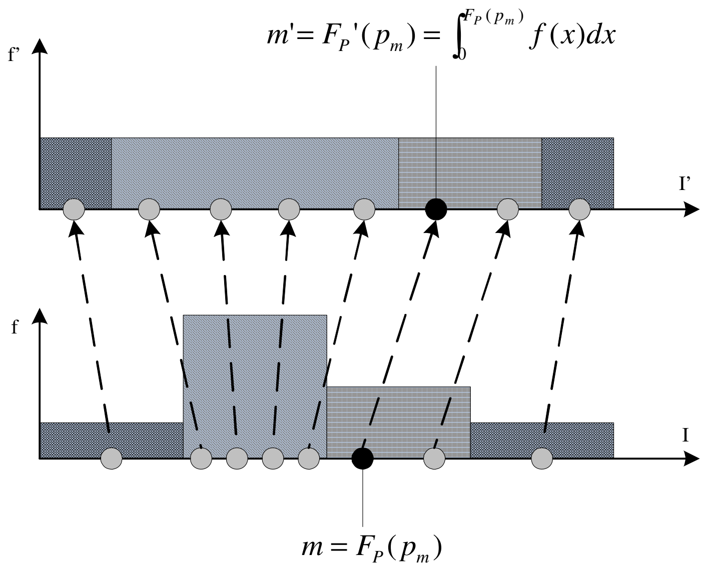


**Figure 5.3.** Transforming skewed key space $\mathcal{I}$ to uniform space $\mathcal{I}'$ is at the heart of using Kleinberg's Small-World principles under real-life workloads.

### 5.5.1. Theoretical Performance of Oscar

Let us assume we want to calculate the expected number of routing hops in an Oscar network constructed on the *base-2* technique with one outgoing long-range link per node. Assume a query from some source node $s$ to some target node $t$. Let us also divide the entire key-space $\mathcal{I}'$ into $\log_2 N$ partitions $B_1, B_2, .., B_{\log_2 N}$, where each partition $B_j$ is populated by the nodes whose distance $d_{\mathcal{I}'}$ from the target node $t$ are bounded by $2^{-\log_2 N+j-1} \leq d_{\mathcal{I}'} < 2^{-\log_2 N+j}$. Each node $u$ which has the query message tries to forward the message towards the target node $t$ greedily, i.e., minimizing the distance $d_{\mathcal{I}'}(u,t)$. After a node forwards the search request to node $s$ we say that the message is at partition $B_j$ if the distance between the current message holder $u$ and the target $t$ is within the range $2^{-\log_2 N+j-1} \leq d_{\mathcal{I}'}(u,t) < 2^{-\log_2 N+j}$. We calculate the probability $P_{next}$ that the current message holder has at least one long-range link to some node $v$ in some partition $B_l$ where $l < j$, i.e., the current message holder can forward the message closer to the target at least by one partition. Assume the node $u$ which belongs to the partition $B_j$ and has the query message is the farthest away from the target $t$, i.e., $d_{\mathcal{I}'}(u,t) = 2^{-\log_2 N+j-1}$. Peer $u$ has its long-range links constructed based on its snapshot view of the global key distribution, i.e., based on the partitions $A_1, A_2, ..A_{\log_2 N}$. In this case, the $j$th partition $A_j$ of $u$ subsumes all the partitions $B_l$ (where $l < j$) which are remaining to be traversed for the message towards target $t$. Since according to the Oscar's wiring technique the probability that peer $u$ will have a long-range link in the partition $A_j$ is $\frac{1}{\log_2 N}$, then probability $P_{next}$ that node $u$ will have a long range link in one of the partitions $B_l$ is also $\frac{1}{\log_2 N}$. Thus, the probability that the message staid in the same partition $B_j$ is $P_{same} = 1 - P_{next} = 1 - \frac{1}{\log_2 N}$.

Let us denote by $X_j$ the total number of hops and by $EX_j$ the expected total number of hops that greedy distance minimizing routing will perform within the partition $B_j$ before jumping into some partition $B_l$ that is closer to the target $t$, i.e., $l < j$. If $N_{B_j}$ is the number of nodes in $B_j$ then we have

$$\begin{aligned} EX_j &= \sum_{i=0}^{N_{A_j}} i Pr[X_j = i] < \sum_{i=0}^{\infty} i (P_{same})^i P_{next} \\ &= \frac{1 - P_{next}}{P_{next}} = \log_2 N - 1. \end{aligned} \tag{5.3}$$

There exist $\log_2 N$ partitions of the key-space and the expected number hops in each of them is less than $\log_2 N - 1$, so the expected total number of hops that the algorithm will need, including the long-range hops is at most $\log_2 N \cdot \log_2 N$. The expected number of nodes that the algorithm will have to visit using neighboring edges from the partition $B_1$ to the target node $t$ is 1 (note that we assume uniform distribution of the peers in $\mathcal{I}'$). Therefore, the algorithm will require polylogarithmic number of hops: $\log_2^2 N + 1$, i.e., $O(\log_2^2 N)$. Similarly it can be proven logarithmic search performance given logarithmic number of long-range links per node.

### 5.5.2. Median Estimation Error

The next problem is to determine how precisely we can estimate a median $m'$ given we have $k$ uniform samples from the population $\mathcal{P}$. For that we will use Hoeffding's inequality [Hoeffding 1963]. Consider a subset $\mathcal{P}_{sub} \subseteq \mathcal{P}$ containing $k$ sample elements drawn uniformly at random from $\mathcal{P}$. Let us assume a set $\mathcal{P}^{m'}_{sub}$ consists of all the key values of population $\mathcal{P}_{sub}$ and $\mathcal{P}^{m'}$ consists of all the key values of population $\mathcal{P}$ in the normalized space $\mathcal{I}'$. In this case, a $m'$-median peer $p_{m'}$ from the sample subset $\mathcal{P}_{sub}$ will have the expected identifier key value $\mathcal{F}_{\mathcal{P}}(p_{m'})$ equal to the expected value of the set $\mathcal{P}^{m'}_{sub}$. Then, according to Hoeffding's inequality, the probability that the measured mean $E(\mathcal{P}^{m'}_{sub})$ and the real mean $E(\mathcal{P}^{m'})$ differs only by some small $\varepsilon$ ($\varepsilon > 0$) is at least $1 - \exp(-2k\varepsilon^2)$, i.e.:

$$P_{sample}[E(\mathcal{P}^{m'}_{sub}) - E(\mathcal{P}^{m'}) > \varepsilon] > 1 - e^{-2k\varepsilon^2} \tag{5.4}$$

Since the distribution of the keys is uniform in $\mathcal{I}'$, the mean value is also the median in $\mathcal{P}^{m'}$. Thus, with probability $P_{sample} > 1 - e^{-2k\varepsilon^2}$, after $k$ samples we will be able to find a peer $p_{m'}$ of which the identifier $\mathcal{F}_{\mathcal{P}}(p_{m'})$ is the median $m'$ of the peer population $\mathcal{P}$ with $\varepsilon$ error. Note that the sampling error $\varepsilon$ is relative to the size of the sampling range in $\mathcal{I}'$. With the decrease of the size of the ranges (as Oscar's sampling algorithm iterates) the sampling error diminishes. E.g., with $k = 30$ samples from the full range [0..1) in $\mathcal{I}'$, it is $P_{sample} > 0.977$ sure that the median is $\mathcal{F}_{\mathcal{P}}(p_{m'}) = E(\mathcal{P}^{m'}) \pm 0.25$. Later in the simulations we will show that Oscar can be successfully constructed having $k$ values as low as 1, resulting in the overall sampling size of $O(\log N)$ (see Section 5.6).

### 5.5.3. Median Estimation Error's influence to the Routing Performance

In practice, each imprecise estimation $m'_{est}$ of the median will result in determining imprecise partitions $A_1, A_2, ..A_{\log_2 N}$. In particular there could be two extreme-case scenarios. In the first case (dense case) the estimation produces more than $\log_2 N$ partitions, and can be modeled as a logarithmic partitioning of the base-$m'^{-1}_{est}$, where $m'_{est} = 0.5 + \varepsilon$ on the range [0..1), i.e., the resulting number of partitions is $\log_{\frac{1}{0.5+\varepsilon}} N > \log_2 N$. In the second case (sparse case) the estimation produces less than $\log_2 N$ partitions, such that $m'_{est} = 0.5 - \varepsilon$ on the range [0..1), and the resulting number of partitions is $\log_{\frac{1}{0.5-\varepsilon}} N < \log_2 N$. The first case scenario tends towards "high clusterisation" of the resulting network, whereas the second one - towards "uniform randomness". This can be interpreted as a shift of the network topology from clustered to random depending on the dimensionality variable $r$ in Kleinberg's work [Kleinberg 2000]. Therefore, by calculating the worst upper bound of the search cost of these two extreme-cases we will provide an upper bound for the performance of the Oscar technique given a partitioning error $\varepsilon$ on the range [0..1).

**Proof for upper bound.** Each peer splits its space $\mathcal{I}'$ into partitions $A'_1, A'_2, ..A'_{\log_a N}$, such that the distance between the peer $u$ and any other peer $v$ in $A'_i$ is bounded by $2^{i-\log_a N-1} \leq$

$d_{\mathcal{I}'}(u,v) < a^{i-\log_a N}$.

Assume a query from some source node $s$ to some target node $t$. Let us also view the entire key-space $\mathcal{I}'$ as $\log_b N$ partitions $B'_1, B'_2, .., B'_{\log_b N}$, where each partition $B'_j$ is populated by the nodes whose distance $d_{\mathcal{I}'}$ from the target node $t$ are bounded by $b^{-\log_b N+j-1} \leq d_{\mathcal{I}'} < b^{-\log_b N+j}$ and $a^{-1} = 1 - b^{-1} \Leftrightarrow b = \frac{a}{a-1}$. Each node $u$ which has the query message tries to forward the message towards the target node $t$ greedily, i.e., minimizing the distance $d_{\mathcal{I}'}(u,t)$. After a node forwards the search request to node $s$ we say that the message is at partition $B'_j$ if the distance between the current message holder $u$ and the target $t$ is within the range $b^{-\log_b N+j-1} \leq d_{\mathcal{I}'}(u,t) < b^{-\log_b N+j}$. We calculate the probability $P_{next}$ that the current message holder has at least one long-range link to some node $v$ in some partition $B'_l$ where $l < j$, i.e., the current message holder can forward the message closer to the target at least by one partition. The probability $P_{next}$ that the node $u$ will have a long range link in $B'_{j-1}$ is at least $\frac{P_{sample}}{\log_a N}$. Thus, the probability that the message staid in the same partition $B'_j$ is $P_{same} = 1 - P_{next} = 1 - \frac{P_{sample}}{\log_a N}$.

Let us denote by $X_j$ the total number of hops and by $EX_j$ the expected total number of hops that greedy distance minimizing routing will make within the partition $B'_j$ before jumping into some partition $B'_l$ that is closer to the target $t$, i.e., $l < j$. If $N_{B'_j}$ is the number of nodes in $B'_j$ then we have

$$\begin{aligned} EX_j &= \sum_{i=0}^{N_{B'_j}} iPr[X_j = i] < \sum_{i=0}^{\infty} i(P_{same})^i P_{next} \\ &= \frac{1-P_{next}}{P_{next}} = \frac{\log_a N - P_{sample}}{P_{sample}}. \end{aligned} \tag{5.5}$$

There exist $O \log_b N$ partitions of the key-space and the expected number hops in each of them is less than $\log_b N$, so the expected total number of hops that the algorithm will need, including the long-range hops is at most $(\frac{\log_a N - P_{sample}}{P_{sample}} + 1) \log_b N$. The expected number of nodes that algorithm will have to visit using neighboring edges from the partition $B_1$ to the target node $t$ is 1 ($\mathcal{I}'$ is uniform). Therefore, the upper bound for expected number of hops is:

$$c \log_2^2 N + 1, \tag{5.6}$$

where $c$ does not depend on $N$ and is given as $c = (P_{sample} \log_2 a \log_2 \frac{a}{a-1})^{-1}$. For example, if we have the sampling error $\varepsilon < 0.25$ (on the range [0..1)) and $P_{sample} > 0.977$ (cf. the example in Section 5.5.2) the routing cost can increase only by a factor $c = 1.23$. The behavior of the coefficient $c$ given different sampling errors and sampling probabilities is depicted in Figure 5.4. Thus, our result shows that a network built by using Oscar's partitioning technique is robust to sampling errors and can sustain reasonable search cost given very high sampling errors. This can be explained by the fact that the absolute error diminishes as the partitions get smaller (closer to a peer), i.e., it confirms the property of Small-World that the network can be efficient even if peers have a very vague estimation of the regions which are far away whereas as long as the close-by areas are well estimated.

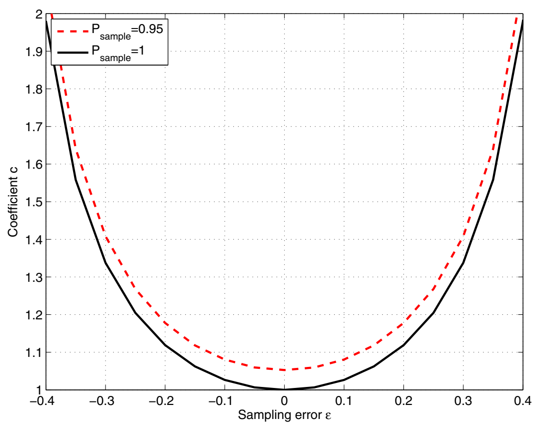


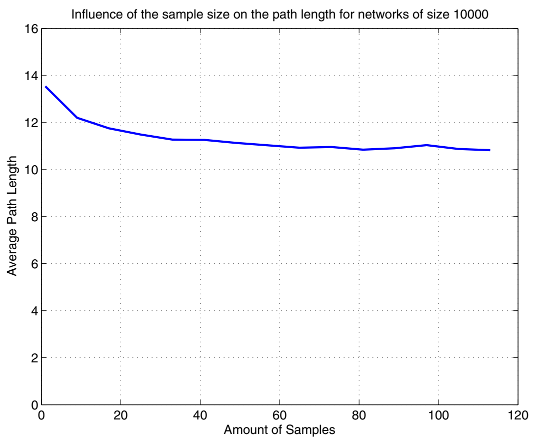
**Figure 5.4.** The behavior of the coefficient $c$ given different sampling errors and sampling probabilities

### 5.5.4. Expected In-Degree

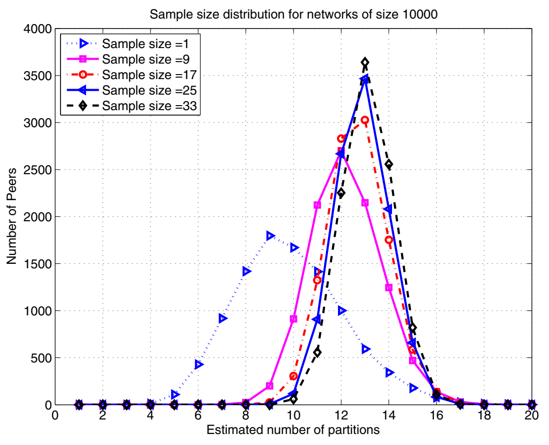
It can be seen from 5.6 that the networks constructed according to our proposed algorithms perform well even with quite high estimation errors. However, the connectivity of the network can be highly affected by the incorrect estimation. Having inaccuracies in measuring the medians implies the possibility of some peers being highly overloaded by incoming links. Let us calculate the expected in-degree of any peer in the case of no inaccuracies and the worst case scenario having some estimation error $\varepsilon$.

**Ideal Case ($\varepsilon = 0$).** Assume a set of peers $C_i$ consisting of $2^{-i}N$ members and a peer $u$, which belongs to partition $A'_i$ of every peer from $C_i$. The probability that the peer $u$ will be chosen as a long range link from any peer $p \in C_i$ is equal $2^{-i}N \cdot \frac{1}{\log_2 N} \cdot \frac{1}{2^{-i}N} = \frac{1}{\log_2 N}$. Since there exist $\log_2 N$ partitions, peer $u$ will have expected in-degree $EX_{in} = 1$.

**Real Case ($\varepsilon \neq 0$).** However, having an imprecise estimation of medians there will be some peers which are more overloaded than others. Let us assume the worst case scenario where peer $u$ is the most overloaded peer. In such scenario $C_i$ set will have $(1+\varepsilon)2^{-i}N$ peers, which will have peer $u$ in their $A'_i$th partitions. Thus, the probability that the peer $u$ will be chosen as a long range link from any peer $p \in C_i$ is at most $(1+\varepsilon)2^{-i}N \cdot \frac{1}{\log_2 N} \cdot \frac{1}{(1-\varepsilon)2^{-i}N} = \frac{1+\varepsilon}{1-\varepsilon} \cdot \frac{1}{\log_2 N}$. Since in the worst case there exist $\log_{2(1+\varepsilon)^{-1}} N$ partitions, the peer $u$ will have an expected upper bound for the in-degree $EX_{in} = \frac{(1+\varepsilon)\ln 2}{(1-\varepsilon)\ln \frac{2}{1+\varepsilon}}$. E.g., as in the example of Section 5.5.2, when the error $\varepsilon = 0.25$ (on the range $[0..1)$) the expected upper bound for the in-degree of the most loaded peer is $EX_{in} = 2.46$.



(a) Search performance of Oscar network given different sampling parameters $k$



(b) The distribution of number of estimated logarithmic partitions at each peer

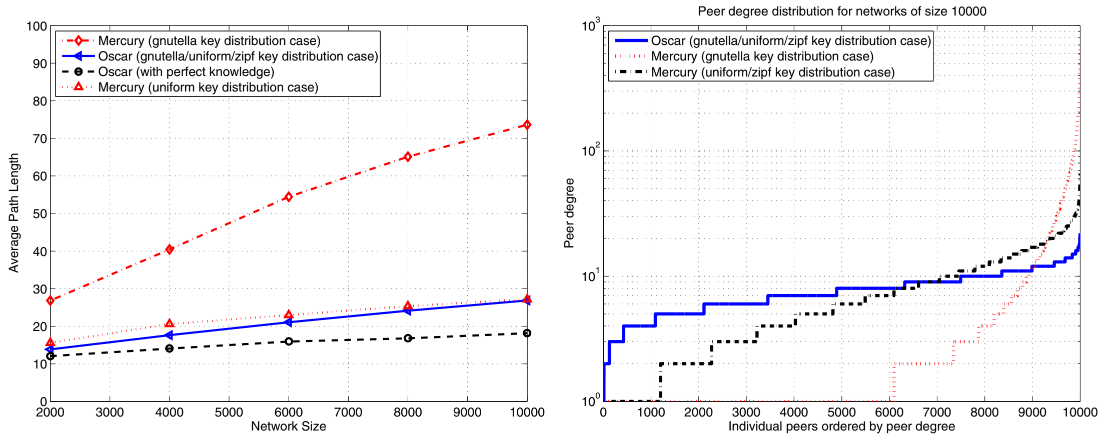
**Figure 5.5.** Performance of the networks with various sample parameters $k$

## 5.6. Simulations

Here we show that the network built according to our proposed technique performs well and does not suffer the drawbacks of existing systems. Similarly as in Mercury [Bharambe *et al.* 2004], we created a discrete-event based network simulator (in Java 1.6) where each application level hop is assigned a unit delay. Using the simulator we simulate an Oscar network with bi-directional links starting from the network bootstrap of 2 nodes and simulating the its growth until it reaches a peer population of 10000. Unless specified otherwise, we set the average node degree to 13 links per peer, Oscar sampling parameter $k$ to 9 and use Gnutella filename trace (cf. Section 5.3) to model the key distribution in the network. We have performed the simulations under various settings, namely varying key and node degree distributions and performed the simulations under churn where the key distribution changes over time. In the following, we will describe the simulation settings in more detail.

### 5.6.1. Oscar's performance with low sample sizes

First we have investigated the optimal parameters for an Oscar overlay. Figure 5.4 suggests that the search performance of Oscar should be sufficiently efficient even with very inaccurate estimations of the medians, i.e., with very small sampling parameters $k$. Hence we measure the effect of the size of the sampling parameter $k$ on Oscar's search performance. We have grown an Oscar network from scratch to 10000 peers with an average node degree of 7 and using different values of $k$. We compared the search performance (Figure 5.5(a)) together with the distribution of the estimated number of partitions by each peer (Figure 5.5(b)). The latter suggests how accurate the measurements are since the "perfect" estimation would result in exactly $\log N$ logarithmic partitions (for $N = 10000$, $\log N \approx 13$). From the experiments we can see that even with very low values of $k$ Oscar peers could estimate well the existing key distribution in the network (in terms of determining the logarithmic partitions), hence the search cost did not deteriorate much even with sampling values as low as $k = 1$. The average search cost given $k = 1$ and $k = 100$ differs by only 2.5 hops. This is a clear indication of the robustness of the Small-World networks which are built using the Oscar technique.



(a) Search performance of Mercury and Oscar given various key distributions (b) Distribution of node degree with Gnutella key distribution

**Figure 5.6.** Simulation Results

### 5.6.2. Oscar vs. Mercury

As suggested in Mercury [Bharambe *et al.* 2004], we have set Mercury's parameters $k_1$ and $k_2$ to $\log_2 N$ for constructing a Mercury network. Each peer in our Mercury simulation constructed a distribution approximation from the sample set of $k_1 \cdot k_2$ random walks. In this way we simulated the exchange of Mercury's distribution estimates in an epidemic manner where each peer issued $k_1$ random walks and each of the selected nodes reported back the $k_2$ most recent estimates. Thus, to sample the network each Mercury peer had to issue $\log_2^2 N$ random walkers. We set the number of random walkers in Mercury to 169 per peer and Oscar's sampling parameter $k$ to 9, which results in the average number of 108 samples per peer for networks of size 10000. With such a setting it is ensured that the sampling parameters in Oscar are not larger than in Mercury, which provides fair simulation conditions.

Each peer $p$ in the network had values $\rho(p)$ and $\rho^{max}(p)$ as the preferred and maximal allowed degree of a peer. During the network construction procedure each peer $p$ was trying to establish $\rho(p)$ bi-directional connections to other peers using long-range links. However, only peers which had less than $\rho^{max}(p)$ degree acknowledged to become peer $p$'s neighbors. This allowed individual peers to autonomously determine their degree and thus the load incurred by them for network maintenance and query traffic. Since both Oscar and Mercury are randomized overlay networks we could employ the power-of-two choice technique [Mitzenmacher *et al.* 2001]

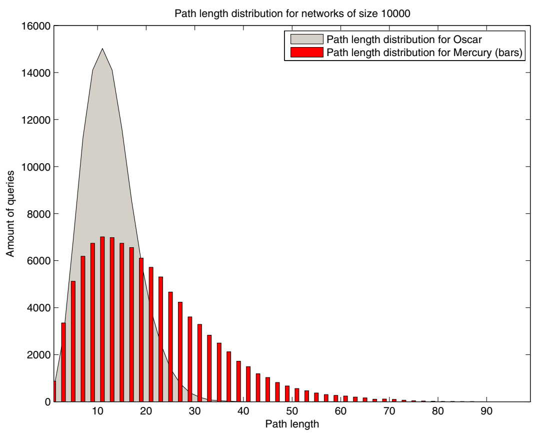


**Figure 5.7.** Distributions of search costs with Gnutella key-distribution

to better load-balance the degree distribution among the peers. During the growth of the networks we were periodically rewiring long-range links of all the peers and measuring the performance of the current network.

**Different Key Distributions.** Since our goal is to show that our proposed technique results in "routing efficient" networks and dealing with churn is an orthogonal issue, we have simulated a fault-free environment, i.e., a system without crashes. We simulated three cases of key distribution: uniform, monotonous Zipf (with parameter $\alpha = 1$) and Gnutella filename (as in Figure 5.1). In each case a peer joining the network was assigned an identifier randomly drawn from the corresponding key distribution. We compare the simulation results also with a "perfect case", i.e., *Kleinbergian* Small-World network [Girdzijauskas *et al.* 2005; Manku *et al.* 2003] built based on perfect knowledge of the global key-distribution. The preferred degree $\rho(p)$ was set to 7 links for every peer. To show the actual capacity of every system to construct routing efficient overlay networks we did not limit the maximum degree value $\rho^{max}(p)$ for any peer. We have measured the performance of the resulting networks; specifically, the average routing cost and the average node degree.

As expected the simulations showed that Mercury performed well given uniform and monotonous skews, but poorly given a complex Gnutella distribution (Figure 5.6(a)). In contrast the Oscar network resulted in a much more efficient network for highly complex key distributions. Figures 5.6(b)and 5.7 indicate that the *Oscar* network had a much better distribution of node degree and lower message cost for the case of Gnutella key distribution. In contrast the Mercury overlay could not cope with the complex distribution of the keys and had significantly higher node degree imbalance, which in turn resulted in poorer lookup performance. As expected the results have shown that Oscar is robust for realistic skews in the key distribution. This is what we ideally want and expect for a system to generate routing efficient Small-World graphs for skewed key-distributions (cf. Chapter 4).

**Different Node Degree Distributions. Heterogenous peers.** We have also shown

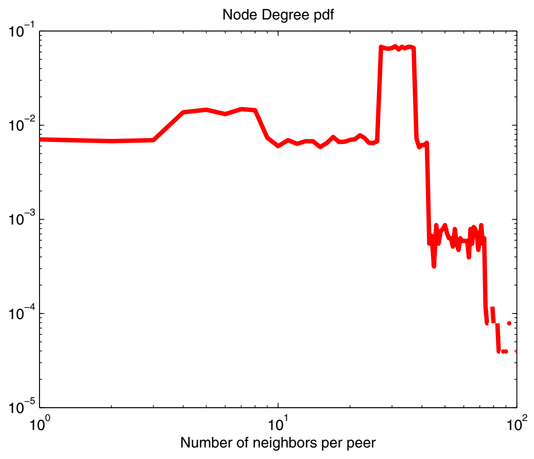
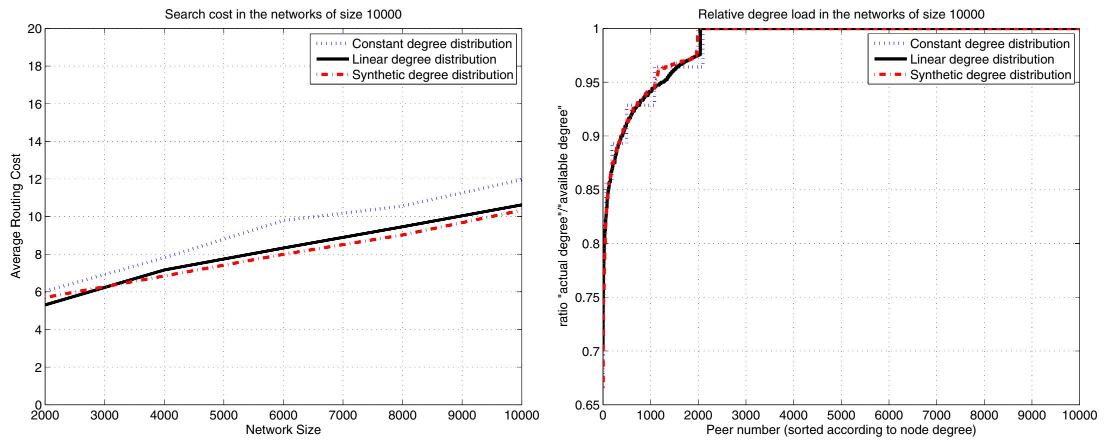


**Figure 5.8.** Synthetic spiky node degree distribution

by simulation that the Oscar technique results in routing efficient networks not only given homogeneous peers but also assuming node-degree heterogeneity. We performed simulations of Oscar given three different node degree distributions: "realistic", "linear" and constant. In the "realistic" node degree distribution case the maximum degree value $\rho^{max}$ of each peer was drawn from a predefined synthetic spiky distribution (Figure 5.8) to emulate the behavior of real P2P systems [Stutzbach *et al.* 2005]. To match the data from [Stutzbach *et al.* 2005] we chose 13 bi-directional links as the mean degree. In the "linear" node degree distribution case, for each peer the $\rho^{max}$ value was drawn uniformly at random from the range of 6 to 20. In the constant degree distribution case, for all peers, $\rho^{max}$ was set to 13. Note that for all the aforementioned cases the average node degree remained 13. The keys for the peers were drawn from the Gnutella filename distribution.

After performing network construction the results showed that Oscar performed almost identically for all the degree distribution cases (Figure 5.9(a)). This shows that Oscar can easily adapt to various degree distributions without any loss in search performance. To measure how well the potential network connectivity is exploited we calculate for each peer $p_i$ the ratio $\frac{\rho(p_i)}{\rho^{max}(p_i)}$ between the actual peer degree and the available (maximal) peer degree. In Figure 5.9(b) we can see that the node degree distribution ratio was very similar in all three cases and exploited around 98% of available degree "volume" $(\frac{100}{N}\sum_i \frac{\rho(p_i)}{\rho^{max}(p_i)})$ in the system of 10000 peers. We also observed in our experiments that in the Mercury network with the same setting and constant node degree distribution only 61% of available degree "volume" were exploited and the Mercury network had an average search cost of 27.3 routing hops per query. Since Mercury could acquire fewer links than Oscar, naturally the search cost in Mercury was higher compared to Oscar.

**Oscar under churn.** Since the data stored on the peers is not static but dynamic, it is expected that because of the storage load balancing the peer key distribution will be changing as well. To investigate the robustness of the Oscar network under churn we performed simulations



(a) Search performance of Oscar given various node degree distributions

(b) Relative degree load in Oscar peers

**Figure 5.9.** Oscar's performance given various key and node degree distributions

of our system in a dynamic environment where the key distribution changes over time. We have modeled a volatile transition from a simple uniform key distribution to Gnutella filename distribution. Our simulation starts with the Oscar overlay of 5000 peers with the uniform key distribution. We model a stable churn rate where every time slot a peer joins or leaves the network with 50% probability. The simulation has two phases. In the first one the arriving peers acquire a key drawn from a uniform distribution. After some time the second phase starts and the new peers start acquiring the keys drawn from the Gnutella filename distribution. We perform the measurements at every time step and measure the average lookup length in the network. Upon joining the network a peer has two options for acquiring the global view of the distribution function: (1) by copying the estimated boundaries of the logarithmic partitions from a ring neighbor; (2) estimating by using sampling. In our simulations a peer chooses with the probability $p$ option (1) and with the probability $(1-p)$ – option (2). We run our simulations with three different settings, where $p = 0$, $p = 0.5$ and $p = 0.9$. In Figure 5.10 the first dotted vertical line marks the starting time of the 2nd phase (the arriving peers start acquiring keys from Gnutella filename distribution). As expected, the network adapts very fast to the dynamic churn when $p = 0$, but even with $p = 0.9$ (the network is sampled only by 10% of the peers) the search cost is still relatively low. This shows that Oscar overlay can sustain efficient routing properties under volatile and dynamic network conditions, and changing load-skews.

## 5.7. Conclusions

In this chapter we have addressed the problem of dealing with skewed key distributions as encountered in data-oriented applications in a realistic P2P environment characterized by churn

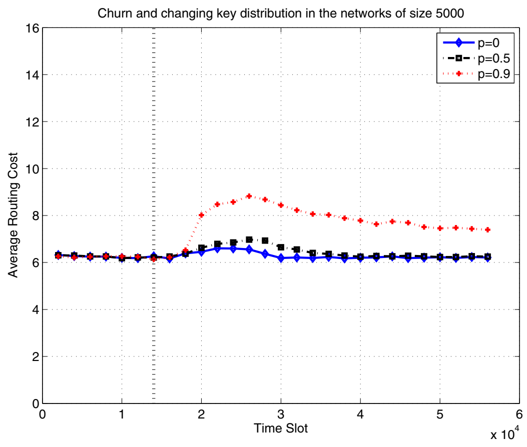


**Figure 5.10.** Routing cost of Oscar under churn

and heterogeneity. We have shown that current approaches cannot cope successfully with such complex workload distributions. Furthermore, most existing structured overlay designs do not deal with peer heterogeneity, and instead assume homogeneity and aim at achieving load-balancing. We take a more pragmatic look at the problem. In deployed (unstructured) overlays, it has been observed that the contribution made by participating peers has large variations, and is decided autonomously by peers subject to their own physical constraints. Load-balancing schemes in presence of peer heterogeneity and autonomy is an impractical ideal. What is pragmatic is instead to respect peers' autonomy to exploit the resources available from each of these peers to fulfill the system's needs adequately and efficiently. External mechanisms based on incentives or punishments to achieve load-balancing in a manner where peers are individually deciding to contribute to the system equally is an orthogonal issue, and will work well in our settings because of the flexible network construction rules. The system design is based on a fundamental understanding of the Small-World networks, and applicability of the principles are validated with simulation experiments. In the actual implementation, our system can borrow a lot of existing design solutions from fine-tuned implementations of ring based overlay networks like Chord, particularly using existing self-stabilization algorithms for ring maintenance and content replication. The only required changes to apply our proposed techniques are the choice of the long range link based on the sampling mechanism.

In the following chapter we will show how to reduce the maintenance cost of Small-World based overlays like Oscar by abandoning one of the main topological features of structured overlays – the ring.

# Chapter 6

# Fuzzynet: Ringless Routing in a Ring-like Structured Overlay

Everything is vague to a degree you do not realize till you have tried to make it precise.

Bertrand Russell

## 6.1. Overview

In the previous chapters we have shown how to use Small-World design principles to construct structured overlays, capable of supporting non-uniform key distributions and exploiting the advantages of such networks for building efficient large-scale systems for heterogeneous environments. In this chapter we show how the Small-World properties can be utilized to reduce the maintenance cost of such systems by discarding the necessity to rely on a ring invariant – a core network connectivity element for most structured overlays. We argue that reliance on the ring structure is a serious impediment for real life deployment and scalability of structured overlays. We propose an overlay called Fuzzynet, which does not rely on the ring invariant, yet has all the functionalities of structured overlays.

This chapter is organized as follows. We discuss ring maintenance challenges in Section 6.2. In Section 6.3 we describe the basic concepts and the design of Fuzzynet, and in Section 6.4 we present and discuss the relevant algorithms. In Section 6.5 we give a theoretical analysis of the approach. In Section 6.6 we validate our design based on simulations and experiments with a Java based Fuzzynet prototype implementation on the PlanetLab[1] testbed. We discuss related systems in Section 6.7 before drawing our conclusions in Section 6.8.

[1] http://www.planet-lab.org/

## 6.2. Motivation

Most commonly, structured overlays are based on enhanced rings, meshes, hypercubes, etc., leveraging on the topological properties of such geometric structures. The ring topology is arguably the simplest and most popular structure used in various overlays [Bharambe *et al.* 2004; Girdzijauskas *et al.* 2007; Manku *et al.* 2003; Rowstron and Druschel 2001; Stoica *et al.* 2001]. In ring based overlays, it is necessary and sufficient to set correctly the successor and the predecessor of each node for correct routing, while additional (long range) links are used to enhance routing efficiency.

Under churn (peer membership dynamics), the ring is both a blessing and a curse. On the one hand, an intact ring is sufficient to guarantee correct routing. Hence, historically, all existing structured overlays have de facto considered it necessary. We argue that it is not only *un*necessary, but also relying on such a ring invariant leads to some undesirable consequences. In certain cases, the existing greedy-routing mechanisms cannot deal with even a single fault/break in the ring on the routing path. On the other hand, in a dynamic environment where peer lifetime is a few minutes for the majority of them, the ring is susceptible to continuous breakages. This in turn incurs high maintenance cost, and despite whatever high maintenance cost, there is at no point any absolute guarantee that the ring is indeed intact. The larger the number of peers, the more likely it is that the ring invariant is violated. This is a serious impediment for scalability and deployment of structured overlays.

Moreover, another well-known challenging issue for the ring invariant is posed by the non-transitive connectivity and the routing anomalies in the overlay networks. It is quite common in real-life networks that some pairs of alive peers cannot directly communicate to each other (e.g., between two firewalled peers); however, it is possible for them to communicate indirectly through a third peer. As it has been shown in [Freedman *et al.* 2005], such non-transitive connectivity may misdirect nodes to wrongly set ring neighbors, thus leading to violation of the ring invariant. This in turn can lead to the disruption of the overlay's functional correctness (e.g., some data items might never be inserted to the peer-to-peer system because of the erroneously assigned responsibility ranges of the participating peers).

The effect of unreachable nodes has been studied in depth by several researchers. Kong et al. [Kong and Roychowdhury 2007] investigated the percolation effect in structured peer-to-peer systems such as Chord [Stoica *et al.* 2001] and Symphony [Manku *et al.* 2003] and measured the size of the reachable network component under failures. In the experiments the authors expose the drawbacks of these systems, specifically showing that up to 4% of the nodes are not reachable (where the network size is $10^6$ peers) even though they belong to the same connected component. Mislove et al [Mislove *et al.* 2006] found routing anomalies in 9% of PlanetLab peer pairs, where the peers could not establish direct connection among themselves. Wang et al [Wang *et al.* 2004a] measured two real P2P systems and found that even up to 36% of the participating peers were residing behind firewalls and Network Address Translators

(NATs), which in many cases made direct communication between these peers impossible. These results show that the inevitable deficiency in the direct communication between any two peers prevents sustaining the ring invariant in real networks. Therefore, it seems that despite the great maintenance cost, in reality structured overlay networks have huge challenges meeting the assumed system invariant. A naive approach to tackle this problem would be to detect and exclude the firewalled peers from the DHT, however, taking out vast quantities of such peers would mean wasting the valuable resources, which in turn would inflict a greater strain on the remaining peers in the system.

The approach presented in this chapter – Fuzzynet, circumvents the need for a ring and the associated problems like non-transitivity and costly maintenance. By introducing the Fuzzynet technique we set a base for a completely *lazy*-maintenance design of P2P systems where the only maintenance action is taken upon peers joining the network. Fuzzynet is based on the connectivity principles of navigable Small-World networks [Kleinberg 2000]. It does not require the ring structure, yet it has all the functionalities of contemporary structured overlay networks. Fuzzynet peers can "mimic" the ring-behavior by contacting the immediate key-neighbors through the neighbor cluster with high probability by exploiting Small-World clusterization. More specifically, the suggested relaxed structure of Fuzzynet has the following differences compared to tightly structured DHTs: i) No explicit ring maintenance. ii) Peers are not deterministically responsible for a particular key section but probabilistically. iii) Data keys are disseminated and replicated in the vicinity of the targeted key. Fuzzynet peers develop their neighbors according to a policy for optimal routing at the joining phase and this effort helps older peers update their stale connections. As it is shown later, this suffices assuming churn rates which have been observed in the deployed applications [Guha *et al.* 2006].

While Fuzzynet is based on loose connectivity and is much more relaxed in its peer-to-key bindings, it is not an unstructured overlay. The peer keys and the stored data keys are highly correlated. The lookup messages in Fuzzynet are never flooded but greedily routed to the targeted area based on the data keys. Even though our system's performance guarantees are probabilistic rather than deterministic, we show that with sufficient amount of neighbors ($O(\log N)$, comparable to traditional overlays), even under high churn the data can be retrieved w.h.p. from our system. In contrast, traditional overlays which rely on a ring provide a deterministic guarantee subject to the condition that the ring invariant is met. However, in reality, this invariant is impossible to meet continuously. As a consequence, systems relying on the ring invariant have poorer performance over time on the average than a probabilistic system, as we observe from the experiments (cf. Section 6.6). Moreover, our suggested solution does not depend on any key distribution, giving Fuzzynet the flexibility to achieve further desired system properties such as load-balancing.

Thus, in contrast to the related literature which tries to improve the ring maintenance mechanisms, we take a complementary approach, where we want to ensure functional correctness (of querying and new data insertions) even in case the ring invariant is violated. Whenever the ring invariant is met, our approach has no message overheads compared to the traditional approaches given similar data replication factor (which is anyway needed for fault tolerance). Therefore, the mechanism can be integrated to work seamlessly in a ring-based P2P network, while avoiding the non-transitivity problems and obviating the need for any aggressive and expensive ring self-stabilization. For the purpose of overall efficiency, a low-cost background self-stabilization mechanism may, however, be employed.

## 6.3. Ringless Overlay

### 6.3.1. The Need For the Ring Structure

To begin the quest of "removing the ring" firstly we have to understand why one needs the ring in the first place. There are plenty of reasons why the ring is an attractive design solution for distributed indexing systems. We will discuss the most important of them.

**Navigability**. First of all, the ring[2] makes the small world easy to navigate, i.e., using decentralized memoryless greedy routing algorithm. The introduction of such a routing technique necessitates the ring structure to assure the correctness of the routing algorithm. Without the ring structure the query messages would not have any guarantees of reaching the desired peer since there will be no assurance for forwarding, i.e., an intermediate peer might not have a "closer" link to the target, thus failing the query.

**Responsibility**. Secondly, and even more importantly for data-oriented P2P systems, the ring structure provides clear responsibility space for every peer. For example, in Chord a peer $p$ is responsible for all data items which hash into the range $\big(F_{\mathcal{P}}(p_{predecessor}), F_{\mathcal{P}}(p)\big]$. With such a knowledge every peer certainly knows which identifer range it is responsible for and which query messages have already "reached the target" and do not need to be forwarded further. The storing/routing/answering decisions can be made because of the certainty that there is no other peer between two successive ring neighbors. Since these storing/routing decisions are basic and essential for any data-oriented P2P system, the ring has to be maintained eagerly (e.g., periodically). Eager maintenance is required even if the churn rate in the network is high and the ring connections (which were established with high cost) have never been used before they are dropped.

**Easy Construction of Long-Range Links**. Thirdly, having the ring as always-in-place concept, it is relatively easy to use it as a bootstrap building block of the long-range links of the P2P system, e.g., Skip graphs with their "multi-dimensional" rings [Aspnes and Shah 2003; Harvey *et al.* March 2003], or the hop count technique [Klemm *et al.* 2007].

Although the ring invariant is a very strong requirement, nevertheless, because of the aforementioned advantages most of the structured overlays employ this idea and impose the

[2] We can generalize the ring to a *Kleinbergian* lattice [Kleinberg 2000] or any other exact, peer key-dependent structure like hypercubes [Schlosser *et al.* 2002], butterfly networks [Malkhi *et al.* 2002], etc.

ring structure in their approaches (e.g., [Aspnes *et al.* 2004; Bharambe *et al.* 2004; Ganesan *et al.* 2004; Manku *et al.* 2003; Stoica *et al.* 2001]).

However, as discussed in the Introduction, even with a high maintenance cost it is practically impossible to meet the ring invariant assuming realistic network conditions, where abundance of participating peers reside behind firewalls and NATs contributing to the frequent routing anomalies, thus making the direct communication between some ring links impossible. Hence, we take a completely different standpoint to design a concept which would not require such a strong assumption as the ring invariant.

### 6.3.2. Fuzzynet Concepts

In order to be able to drop the ring invariant we need to address the aforementioned functional requirements which make the ring an attractive solution in the P2P community.

Firstly, Fuzzynet drops the requirement for every peer having a predefined deterministic responsibility range on the identifier space. Instead, we use a probabilistic responsibility approach, where a data item will be likely stored on a peer whose key on the identifier space is close to the hash value of that data item (data key).

Secondly, we employ a data replication concept in Fuzzynet, by disseminating the data replicas in the vicinity of the data position on the identifier space. Since P2P overlays (Small-World networks) exhibit a high clusterization effect, the data dissemination in the vicinity of the desired position can be performed with relatively low effort. Such dissemination of data replicas does not actually rise the requirements for our system, since all the realistic systems (which use the ring structure) employ replication for fault tolerance and persistence anyway. A useful consequence of such data replication in Fuzzynet is the fact that a simple greedy routing query will find one of the replicas w.h.p. given sufficient network connectivity.

To make the Fuzzynet concept work we need to be able to construct a navigable overlay, i.e., a Small-World network. For that we can use some of the existing approaches, e.g., Oscar's long-range link acquisition technique presented in Chapter 5 designed for skewed key distributions or [Galuba and Aberer 2007] for uniform key distributions.

In the following we will discuss in more detail the above described principles of Fuzzynet, which can be generalized as two types of routing: routing for lookup (read) and routing for storing or publishing (write).

#### 6.3.2.1. Lookup (Read)

Lookup routing will employ a greedy routing algorithm, where messages will be forwarded every time minimizing the distance to the target. The routing terminates if the looked-up data item $D$ is found. However, since there are no ring links and no predefined responsibility ranges, a peer might end up in a situation where it does not have any links which would lead the query closer to the target, nor it holds the requested data. In such a case the lookup query would

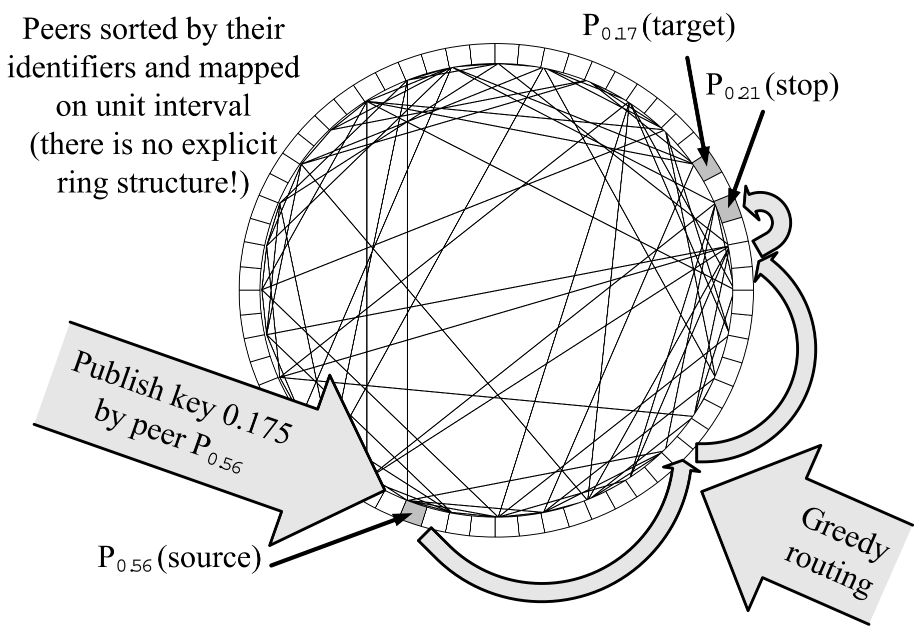

Peers sorted by their identifiers and mapped on unit interval (there is no explicit ring structure!)
P0.17 (target)
P0.21 (stop)
Publish key 0.175 by peer P0.56
P0.56 (source)
Greedy routing

**Figure 6.1.** Greedy-Approach. Routing from the originator peer ($P_{0.56}$) to the *greedy-closest* peer ($P_{0.21}$) where the greedy approach towards the target key 0.175 (*actual-closest* peer $P_{0.17}$) is no further possible.

terminate unsuccessfully. Nevertheless, we will prove later in the analysis and show with the experiments that with realistic parameters w.h.p. data is found if it was published before (e.g., in the networks with $O(\log N)$ degree and typical data replication cost).

#### 6.3.2.2. Publish (Write)

The high guarantees for the lookup lie in the exploitation of a particular Small-World property, namely the clusterization property, during the data writing phase.

The write operation is performed in two stages and stores data $D$ on $r$ peers (replicas) in the vicinity of the data key $F_{\mathcal{R}}(D)$. The first stage is similar to the lookup (read) phase and uses greedy routing to find one of the peers which are close enough to the data key $F_{\mathcal{R}}(D)$. Once the write operation reaches the vicinity of the target key location, the data is seeded in the nearby peers by the self-avoiding multicast (a controlled "Write-Burst"). The underlying idea is to use the clusterization property of the network and to reach as many peers as possible in the $F_{\mathcal{R}}(D)$ vicinity. The multicast has two parameters - $fn$ (fanout) and *depth*. A peer contacts its $fn$ closest neighbors to $F_{\mathcal{R}}(D)$ and requests to store the data item $D$ as well as to continue the multicast process with reduced *depth*. The multicast avoids the peers which have been visited (already store data $D$) and terminates when *depth* reaches zero. Data $D$ is seeded (stored) on all the peers reached by the multicast-burst. An example of the publish (write) procedure is given in Figure 6.1 and Figure 6.2 followed by the example of the successive lookup (read) operation in Figure 6.3.

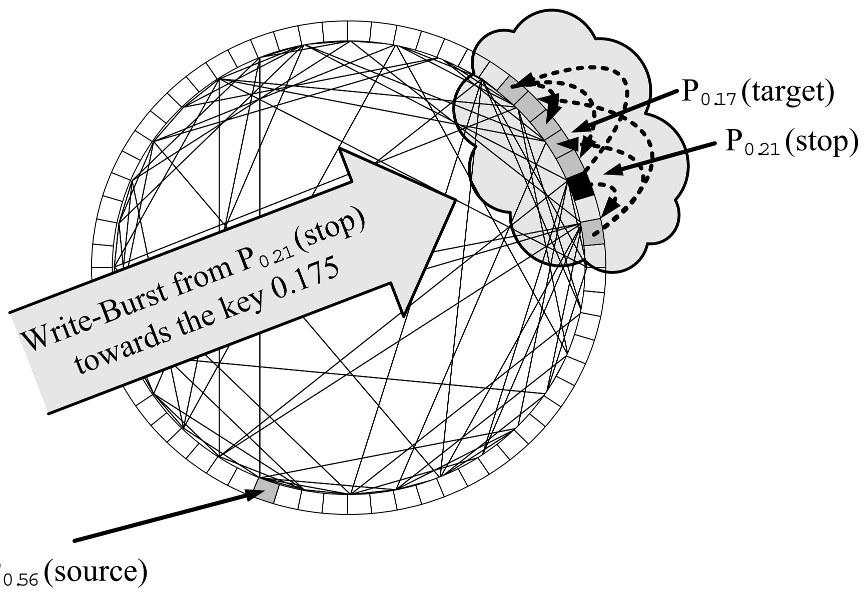


**Figure 6.2.** Write-Burst. The *greedy-closest* peer ($P_{0.21}$) seeds the replicas in the cluster vicinity of the key 0.175 using the Write-Burst.

In contrast to "classical" decentralized data-oriented systems, the replicas in our probabilistic overlay do not need to be globally aware of each other. Although the write procedure is more complex than read (lookup), there are no messages wasted, i.e., only peers which will be storing the data are contacted.

By exploiting the clusterization property, the Write-Burst operations avoid the necessity of the ring and circumvent the non-transitivity problems. In a way, the bursting technique finds the bypasses to the "would-be" ring-neighbors by choosing second or third best neighbors and relying on the fact that in a Small-World network, peers in the same vicinity are highly connected. Instead of keeping the ring structure alive periodically (as in the classical P2P systems) the write procedure in the overlay probabilistically "imitates" the ring behavior only for the storage, whereas the read (lookup) does not actually need the ring if the replication factor is sufficiently large. For all practical purposes, the minimal amount of replication used by current systems purely for the purpose of fault tolerance appears to be sufficient.

It is worth emphasizing that Fuzzynet does not flood the entire network, but affects only a very small neighborhood. By tuning Write-Burst parameters the size of that neighborhood can be adjusted not to exceed a typical replication count of structured overlay networks. It will be shown later in Section 6.6 that it is sufficient to have fanout $fn = 2$ and $depth = 3$ for successful storage and retrieval in a system with $O(\log N)$ average node degree even with network failures.

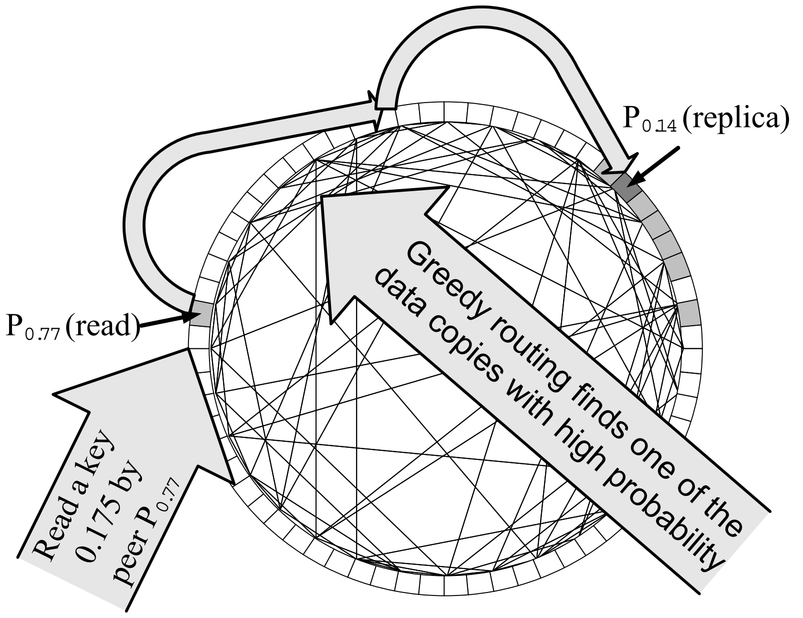

P0.14 (replica)
P0.77 (read)
Read a key 0.175 by peer P0.77
Greedy routing finds one of the data copies with high probability

**Figure 6.3.** After writing the data in the vicinity of the key 0.175, the lookup (read) from any node will have very high chance finding at least one of the data replicas.

#### 6.3.2.3. Updates

Fuzzynet treats data updates in the same way as data insertions, i.e., using the above described "Publish" concept. The updates are made when a new (updated) value $D'$ of the existing data $D$ is published on the same key $F_{\mathcal{R}}(D)$. During publishing, the Write-Burst seeds the updated data items in the vicinity of the key $F_{\mathcal{R}}(D)$, overwriting the existing old data. Although the replica placement in the vicinity of the $F_{\mathcal{R}}(D)$ is non-deterministic, their high degree of clusterization typically ensures that a suitable implicit write quorum is found to overwrite the older version. Such non-deterministic placement of replicas of objects in Fuzzynet makes consistency maintenance subtler, even if not more expensive.

For storage purposes DHTs can be used in two different ways - (i) the DHT routing determining the peers which store actual objects, (ii) the DHT routing determining the peers which store pointers to the actual object, making the storage placement an orthogonal issue.

In the latter case which has more flexibility and is often preferable [Bhagwan *et al.* 2004], maintenance of consistency of the objects is handled by application layer logic. In this case, non-deterministic placement of replicas of pointers does not affect the consistency. However, if some DHT replicas have stale pointers, then they can unnecessarily add to the routing overheads. In general, DHTs store objects with a time to live, and garbage collect stored objects unless they are reinserted [Rhea *et al.* 2005b]. Such a garbage collection mechanism, in conjunction with a push&pull gossip based probabilistic update mechanism [Datta *et al.* 2003] ensures that the pointers that actually stay in the DHT system stay up-to-date. This is also true when the DHT itself is used for storing objects (instead pointers).

The replicas which receive the new updates (with new values or reinsertion of the same old value) will persist in the system. However, if some of the replicas don't receive the updates, they try to pull it from other random replicas regularly. Since most online replicas already have the latest update, such random pull provides the latest update w.h.p. [Datta *et al.* 2003]. The chances of missing updates using the combined push&pull approach is negligible. But in the unlikely case that it happens, the affected replica considers that as the equivalent case of the stored object not being reinforced, and thus simply discards the local copy. As a consequence, in the long run, only the latest version of stored content (or pointer) actually persists in the DHT. Note also, that any other P2P system using replication similarly needs to keep these replicas up-to-date (e.g., the same update mechanism is used in P-Grid [Aberer 2001]), and hence the replica maintenance overheads are comparable, and does not specifically increase Fuzzynet's overheads, but is a price to pay to support mutating content in a P2P network.

#### 6.3.2.4. Maintenance

The basic idea of our approach is to ensure a sufficient performance of the system with no explicit maintenance. We do not have any assumptions on how peers leave, i.e., we deal similarly with both the departures and failures. In this way we can provide much stronger guarantees that our system will be functional given any circumstances. In an environment where the churn rate is stable, i.e., similar numbers of peers join and leave/crash, it is not necessary to perform any maintenance of the overlay except the one triggered by peer arrivals. The churn itself is typically enough to keep the system functioning, i.e., the newcomers with the fresh links will compensate the lost connectivity due to crashed peers [Klemm *et al.* 2007].

Similarly, the churn can keep the residing data alive when the newcomers replicate and republish the data if deemed necessary (cf. Section 6.4.1.3). Every time a new peer joins the network, it checks all the data which is similar to its identifier key. The newcomer can decide to refresh the data by performing a new write if it notices that the replication factor is too small, i.e., a peer does not see enough replicas.

## 6.4. Algorithms

Here we formally describe the algorithms used in our system. Fuzzynet is a general technique that can enhance any P2P overlay which is based on Small-World connectivity but has to rely on the ring integrity for correctness (e.g., [Bharambe *et al.* 2004; Galuba and Aberer 2007; Manku *et al.* 2003], etc). The Fuzzynet algorithms implemented on top of such systems allow to abandon the ring integrity constrain without loosing any of the functions of a regular DHT. In this work, for establishing the underlying network of our system we use the Oscar overlay construction algorithms presented in Chapter 5, which form the Small-World connectivity and can cope with non-uniform (skewed) key distributions of any complexity.

### 6.4.1. Fuzzynet Data Management Algorithms

On the established Small-World network we employ the Fuzzynet algorithms to enable successful data storage and retrieval without the presence of a ring. Here we will describe the core algorithms of our ringless data-oriented overlay.

#### 6.4.1.1. Write-Burst Algorithm

**Algorithm 6.1** Publish (Write) algorithm $publish(data2store)$

```
1: $[burstPeer] = greedyRoute(F_{\mathcal{R}}(D))$
2: $writeBurst(burstPeer, data2store, F_{\mathcal{R}}(D), fanout, depth, \oslash)$
```

To store a data item $D$ a peer initiates a two-phase publish algorithm (Algorithm 6.1). In the first phase it looks-up the closest peer ($currentPeer$) to the data key $F_{\mathcal{R}}(D)$ it can find with greedy routing algorithm, and once found, the "Write-Burst" (Algorithm 6.2) is initiated at the $currentPeer$. The algorithm contacts the closest neighbors to $F_{\mathcal{R}}(D)$ (number of neighbors defined by $fanout$) from the $currentPeer$. Once the closest neighbors are reached the algorithm stores the data $D$ on the visited peers and recursively continues contacting the closest peers, until the maximum allowed $depth$ is reached (i.e., $depth = 0$). Every recursive thread maintains a set of peers which were already visited (Algorithm 6.2, line 3) and avoids visiting these peers later on. With the "Write-Burst" algorithm we exploit the clusterization property of small world networks and recursively visit as many peers in the vicinity of $F_{\mathcal{R}}(D)$ as possible.

**Algorithm 6.2** Write-Burst algorithm $[visited] = writeBurst(p, D, F_{\mathcal{R}}(D), fanout, depth, visited)$

```
1: $depth = depth - 1$
2: **if** $depth \geq 0$ **then**
3:    $visited = visited \cup p$
4:    store $D$ on $p$
5:    $notVisitedNeighbors = getNeighbors(p) \setminus visited$
6:    $fanoutCounter = min(fanout, notVisitedNeighbors)$
7:    **while** $fanoutCounter > 0$ **do**
8:       $CloseLink = chooseClosestToTheKey(notVisitedNeighbors, F_{\mathcal{R}}(D))$
9:       $[visited] = writeBurst(CloseLink, D, F_{\mathcal{R}}(D), fanout, depth, visited)$
10:      $fanoutCounter = fanoutCounter - 1$
11:      $notVisitedNeighbors = notVisitedNeighbors \setminus visited.$
12:   **end while**
13: **end if**
```

#### 6.4.1.2. Lookup (Read) Algorithm

The lookup or read algorithm is fundamentally the same as the traditional greedy routing algorithm. When a lookup request is issued on $F_{\mathcal{R}}(D)$, the query travels greedily towards the target examining at each peer whether it stores a copy of $D$. Once terminated (no more possibility to get closer to the target) the algorithm analyzes the collected information about the data $D$ and with high probability (cf. Section 6.5) a replica of $D$ is found.

#### 6.4.1.3. Peer Join Algorithm

As discussed earlier in this chapter, the join algorithm uses the original Oscar technique for long-range link acquisition to wire the network, i.e., to ensure the connectivity and the desired Small-World properties. Here we will discuss only the second stage of the join process, i.e., the data management after the network connectivity is established.

Once a peer $p$ joins the network and establishes its connections it needs to copy the data which is stored in the vicinity of the peer's key $F_{\mathcal{P}}(p)$ in the key space. For this reason the newcomer peer $p$ performs a Write-Burst algorithm, but instead of writing the data, it collects all the "visible" neighbors in the vicinity of the joining peer's key $F_{\mathcal{P}}(p)$. Then peer $p$ collects the data from the neighborhood peers and analyzes it (Algorithm 6.3, line 3). Peer $p$ copies (becomes a replica) of all the data whose keys are close enough to its own key $F_{\mathcal{P}}(p)$ given the existing replication rate induced by the $fanout$ and *depth* parameters (Algorithm 6.3, line 5). The estimation whether a data item $D$ belongs to the peer's vicinity is easy even for non-uniform key distributions, since a peer knows in advance the boundaries of the logarithmic partitions $A_1, A_2, .., A_{\log N}$ from the overlay building process (cf. Section 5.4) and can estimate the number of peers residing between $F_{\mathcal{R}}(D)$ and $F_{\mathcal{P}}(p)$.

Similarly, a joining peer $p$ could estimate how many replicas of a particular data item $D$ should be visible, taking into account the key of the current data item $F_{\mathcal{R}}(D)$ and peer $p$'s key $F_{\mathcal{P}}(p)$. In case the amount of data items is lower than some predefined threshold, peer $p$ can initiate the write algorithm to reinsert data item $D$. In such a way, if peers leave/crash and arrive independently, it is ensured that the data item will not be lost once it was written in the P2P system.

## 6.5. Analysis

In this section we will give lower bounds for the success probability to retrieve a data item $D$ once it was stored in our system. We will investigate the case of Write-Burst for publishing data with the parameters $fanout = 2$ and *depth* $= 3$. As our simulations suggest, these are the smallest values with which the system performs reasonably well (cf. Figure 6.5) having relatively low average network degree. Although with smaller Write-Burst values the success rate might still be high enough, there will be much fewer replica copies in the system, thus

**Algorithm 6.3** data acquisition algorithm upon join $[targetPeer] = join(p)$

```
1: $clusterNeighbors = \emptyset$
2: $[clusterNeighbors] = writeBurst(p, \oslash, F_{\mathcal{P}}(p), fanout, depth, \oslash)$
3: for $\forall data\ D \in clusterNeighbors$ do
4:   if $estimateIfDataInVicinity(F_{\mathcal{P}}(p), F_{\mathcal{R}}(D), fanout, depth)$ then
5:     store $D$
6:   end if
7:   $estimatedNumberOfDataItems = estimateData(F_{\mathcal{P}}(p), F_{\mathcal{R}}(D), fanout, depth)$
8:   if $estimatedNumberOfDataItems < tresholdNumber$ then
9:     $publish(D)$
10:  end if
11: end for
```

increasing the risk to loose all of them in case of unexpected increase in churn. Therefore, with the aforementioned parameters we will establish the general lower bound of the success probability for acquiring a stored data item in our system.

We will calculate the success probability in three steps. In the first step we will calculate with what probability a read/write message can reach the vicinity of the data key if the ring connectivity is not enforced (i.e., when no more greedy routing is possible). In the second step we will calculate the probability that the Write-Burst will populate the immediate neighbors of the Write-Burst originator. And in the third step we will combine the two and calculate what is the general success probability for a read message to reach a peer which holds a written data item.

### 6.5.1. Step 1. Routing Without Ring-Links

Here we calculate the probability of a message to be delivered from an originator peer $p_o$ to the target peer $p_t$ using only the existing Small-World network links without making the ring connectivity assumption. We assume the worst case scenario when the distance $d(p_o, p_t)$ is maximal, (e.g., equal to 0.5 in the setting of the unit interval where the links between the peers are bi-directional and the peer keys are distributed uniformly). Let us divide the space between peer $p_o$ and peer $p_t$ into logarithmic partitions $B_1, B_2, .., B_i$ where $B_1$ partition contains the farthest $\frac{1}{2}$ of the peers from the $p_t(id)$, residing in the identifier space between $p_o(id)$ and $p_t(id)$, $B_2$ - the next closer $\frac{1}{4}$ of the peers and so on. If the ring links were present and routing tables at each peer are logarithmic in size, the message from the peer $p_o$ to the peer $p_t$ will have to hop over $\log N$ peers on average, at each peer diminishing the distance to the target exponentially, i.e., by at least one logarithmic partition $B_j$ [Girdzijauskas *et al.* 2005]. However, without the ring links, starting from the second hop there exists a non zero probability that there will be no links pointing to the peers which would allow a greedy approach to the target, i.e., the ring neighbor link is missing. The probability of not-getting closer is relatively small in the first hops of the query, but it grows significantly when the message approaches the target. It is because the remaining logarithmic partitions become smaller and smaller, thus the query finds fewer and fewer links which point into the remaining partitions.

Let us denote with $P^j_{hop}$ the probability that a message in the partition $B_j$ will be able to approach greedily to the target, i.e., that at the routing algorithm will be able to find at least one link which can forward the message closer to the target peer. The probability for a Small-World network building process [Girdzijauskas *et al.* 2006] to establish a link from the $j$th partition to one of the remaining partitions $B_{j+1}, .., B_{\log N}$ is $\frac{\log N - j + 1}{\log N}$. Thus, we can calculate the probability $P^j_{greedy}$ that being at the $j$th partition a query can find at least one link pointing into one of the partitions $B_{j+1}, .., B_{\log N}$ is equal to $1 - (\frac{j-1}{\log N})^{\rho(p)}$, where $\rho(p)$ is the average size of the routing table.

Since we assume the worst case scenario, the probability $P^1_{hop}$that the peer will be able to leave the partition $B_1$ and traverse closer to the target is at least $P^1_{hop} \geq 1 - (1 - \frac{\log N - 1}{\log N})^{\rho(p)}$. For every next partition $B_j$ when $j > 1$ we have to calculate $P^j_{hop}$ recursively. In general we obtain $P^j_{hop} = P^{j-1}_{hop} \cdot P^j_{greedy}$. Hence, we calculate the probability $P^j_{stay} = P^{j-1}_{hop} \cdot (1 - P^j_{greedy})$, where $P^j_{stay}$ is the probability of a greedy routing algorithm terminating in the partition $B_j$ without the possibility to advance closer to the target.

## 6.5.2. Step 2. Replication by Write-Burst

When a "write" message for the data item $D$ reaches peer $p_t$ from which no more greedy routing can be performed, the Write-Burst algorithm is initiated to replicate the data in the surrounding area of $p_t(id)$. The peer $p_t$ can immediately store the data, i.e., it will be populated with data $D$ with probability 1; however, the neighboring peers do not have 100% guarantee to get populated. Here we will investigate the probabilities for the peers in the closest partitions $B_{\log N}$ and $B_{\log N-1}$ to get populated by the new data from peer $p_t$.

Firstly, we will calculate the minimal probabilities that $p_t$ will be connected to these closest neighbors by one or two hops. These probabilities will set a lower bound connectivity among these closest neighbors. According to the *Kleinbergian* Small-World network construction principles the probability that a peer will have a direct long range link to one of its immediate neighbors in the partition $B_{\log N}$ is $\frac{1}{\log N}$ and in general the probability $P'_1$ that one of its long range links will have a link to the immediate neighbor is $P'_1 = 1 - (1 - \frac{1}{\log N})^{\rho(p)}$. Similarly, the probability $P'_2$ that a peer will have a direct long range link to one of its immediate neighbors' neighbor ($2^{nd}$ order neighbors in the partition $B_{\log N-1}$) is $P'_2 = 1 - (1 - \frac{1}{2 \log_2 N})^{\rho(p)}$. The probability $P''_1$ that a peer will be connected to one of its immediate neighbors by two hops is at least $P''_1 \geq P'^2_2$. Likewise, the probability $P''_2$ that a peer will be connected by two hops to one of its immediate neighbors' neighbor ($2^{nd}$ order neighbor) is $P''_2 \geq P'_1 \cdot P'_2$. Therefore, the probability that peer $p_t$ can reach and store data on a peer in $B_{\log N}$ partition is $P_1 \geq 1 - (1 - P'_1)(1 - P''_1)$. Similarly, the probability that peer $p_t$ can reach and store data on the peer in the partition $B_{\log N-1}$ is $P_2 \geq 1 - (1 - P'_2)(1 - P''_2)$.

Let us assume that peer $p'_t$ is the closest peer to the data identifier which has to be written. The probability that the write message will stop at that peer is $P_{stay}^{\log N+1}$. Similarly, the probabilities that the message will stop a peer on the partitions $B_{\log N}$ and $B_{\log N-1}$ are $P_{stay}^{\log N}$ and $P_{stay}^{\log N-1}$, respectively. In order for the data to be written on the node $p'_t$, the Write-Burst algorithm had to start either in the $p'_t$ node itself or in its neighborhood. On expectation there exist one node in the partition $B_{\log N}$ and two nodes in the partition $B_{\log N-1}$. We will calculate what is the probability $P_{T0}$ that a data item $D$ was written on node $p'_t$ if the write algorithm stopped on a peer in the partition $B_{\log N}$ or one of the peers in the partition $B_{\log N-1}$. The probability that a Write-Burst algorithm started at a node in $B_{\log N}$ and the data item $D$ was written on the node $p'_t$ is $P_{stay}^{\log N} \cdot P_1$. Likewise, the probability that the Write-Burst algorithm started at a node in $B_{\log N-1}$ partition and the data item $D$ was written on the node $p'_t$ is $P_{stay}^{\log N-1} \cdot P_2$. Combining three different scenarios of data being stored on $p'_t$ (directly or from $B_{\log N}$ or from $B_{\log N-1}$) we get $P_{T0} \geq P_{stay}^{\log N+1} + P_{stay}^{\log N} \cdot P_1 + P_{stay}^{\log N-1} \cdot P_2$. Similarly, we can calculate the probabilities $P_{T1}$ and $P_{T2}$ that the data item $D$ was written on the nodes in the partitions $B_{\log N}$ and $B_{\log N-1}$ respectively, assuming the write burst algorithm started in the vicinity of these nodes. Therefore, $P_{T1} \geq P_{stay}^{\log N} + P_{stay}^{\log N+1} \cdot P_1 + \cdot P_{stay}^{\log N-1} \cdot P_1$ and $P_{T2} \geq P_{stay}^{\log N} \cdot P_2 + 0.5 \cdot P_{stay}^{\log N-1} + P_{stay}^{\log N+1} \cdot P_2$. It is important to note that for the worst-case scenario (lower bound) we assume only the most likely connectivity patterns for calculating $P_{T0}$, $P_{T1}$ and $P_{T2}$.

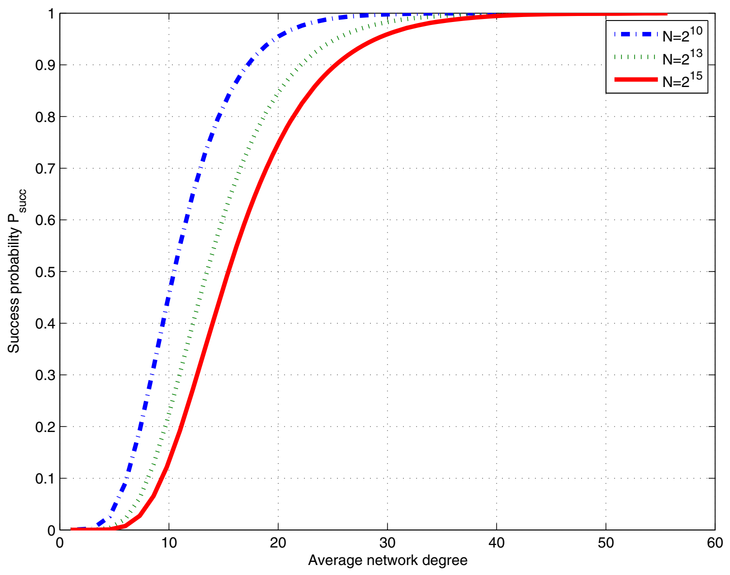


**Figure 6.4.** Lower bound on success probability.

### 6.5.3. Step 3. Lower Bound on Success Probability (Worst Case Scenario)

Having the probabilities that the data item $D$ was written in the vicinity of the identifier of $D$', we can calculate what will be the lower bound of the success probability $P_{succ}$ if a "read" message will be issued at some peer in the system. Since we know what are the probabilities for a greedy routing algorithm to advance towards the target without using the ring-links we can calculate that $P_{succ}$ is at least $P_{succ} \geq P_{T0} \cdot P_{stay}^{\log N+1} + P_{T1} \cdot P_{stay}^{\log N} + P_{T2} \cdot P_{stay}^{\log N-1}$.

In Figure 6.4 we can see the plot of the lower bound success probabilities with different sizes of networks and network's degrees. The above result suggests that given an average network degree of $2\log_2 N$, regardless the network size $N$, the queries in Fuzzynet will be successful w.h.p. even in the worst case scenario. In practice, the success rate in Fuzzynet is even higher (cf. Figure 6.5).

## 6.6. Experimental Results

### 6.6.1. Simulations

In our simulations we have experimented with the Fuzzynet technique built on top of Small-World networks. We investigated what the probabilistic guarantees are to retrieve the data which was stored on the ringless overlays and also compared our approach to ring-based Small-World networks. We have simulated a network environment with routing anomalies where the peers behind firewalls and NATs were not allowed to establish a link directly with each other. Our experiments were carried out on various network connectivity cases, mainly networks with different node degrees. All the experiments were performed including peer churn as described next.

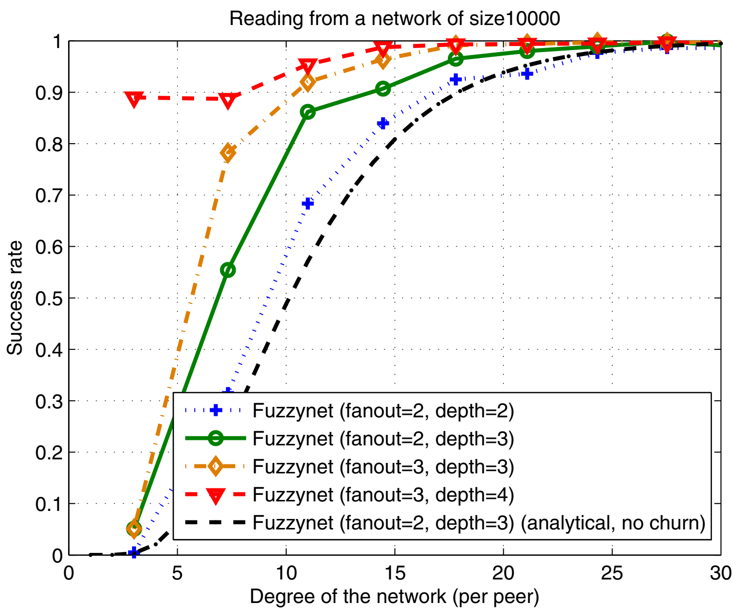


**Figure 6.5.** Query success rate in Fuzzynet under churn.

In the first part of the simulations we have performed tests with a network of 10000 peers on average and a predefined average node degree. For each case of the average degree we have performed separate experiments on a dynamic network where the peer arrivals (joins) and departures (leaves) were generated from the trace of the Skype super-peer network [Guha *et al.* 2006]. Leaving and failing were considered as identical operations, since no action is taken upon a "graceful" leave. The correctness of the system relies only on the actions taken upon new peers joining the network: establishing new links and replicating data from the surrounding areas. We have modeled the behavior of the networks for over 18000 arrival/departure events. At the beginning of the experiment 100 unique data items were inserted into each peer with uniform random keys. During the experiments we have captured the snapshots of the network after every 2000 arrival/departure events. For each snapshot of the network we have performed a *read* query from every peer on the identifiers of the previously written data items. The average success rate and the average search cost (path length) of the queries were measured. Figure 6.5 shows the average query success rates given different fanouts and depths of the Write-Burst algorithm and various average degrees of the networks. We observe that with a relatively low average degree ( 15 links per node) the success rate rises up to almost 100%, with the parameters $fanout = 3$ and $depth = 3$. In Figure 6.6 we show the average search cost for looking up the data.

We experimented with various rates of peer departures superceding the rate of new peers joining, as a result of which the network shrank (i.e., non-equilibrium scenarios). We studied the system's performance until it approximately shrank to half its original size, i.e., 5000 peers from the original 10000. We varied the shrinking rates from relatively slow ones, taking 5000 time slots to halve the network, up to very rapid ones, lasting only 5 time slots until the network reached half of its original size. Fuzzynet proved to be resilient against even such drastic peer population changes. With the average network degree of 20 and with Write-Burst parameters $fanout = 2$ and $depth = 3$, the rate at which the network shrank had a very weak influence on query success rate, which stayed above 96%. Notice that tolerating such a huge membership change within a short interval is essentially equivalent to having a correlated failure of the corresponding number of peers, and hence we conclude from these experiments that Fuzzynet can deal with a massive correlated failure, even without any repair operations, because it does not need the ring invariant for functional correctness.

In the second part of the simulations we have compared the Fuzzynet technique to ring-based approaches performing under the faulty environments. As a ring-based overlay we have implemented Symphony [Manku *et al.* 2003] algorithms, where there is only one peer responsible for a particular data item and the replication of that data (7 replicas). To simulate the effect of peers under firewalls we have labeled some of the peers as "firewalled" peers. During the lifetime of the networks we have forbidden the direct communication among any two labeled peers [Wang *et al.* 2004a]. We have also simulated sporadic routing anomalies by forbidding a communication between randomly selected fraction of existing links [Mislove *et al.* 2006]. In the experiments we have monotonically increased the fraction of firewalled peers and the

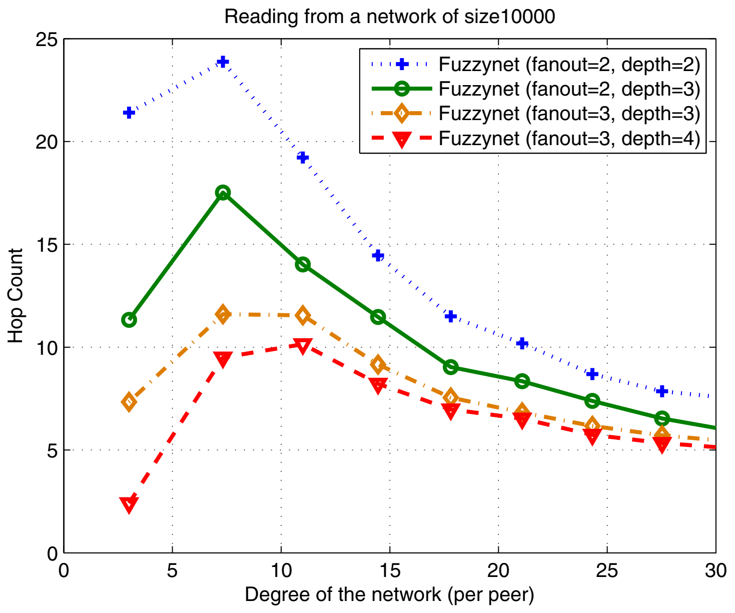


**Figure 6.6.** Total search cost of both: successful and unsuccessful queries.

probability of routing anomalies, such that the total link failure rate was increasing from 0 to 0.3. We have simulated churn as in the first part of the simulations and measured the performance of the networks every 2000 arrival/departure events. The results in Figure 6.7 show that even with high link failure rates the Fuzzynet technique with parameters higher than $fanout = 2$ and $depth = 2$, has much higher success rate than a ring-based approach. With lower values, there are not enough replicas to sustain the data under churn, hence the queries fail to find some of the previously inserted data items, since they disappear from the network. However, with higher values of the these parameters, no data is lost, given the existing replication. Figure 6.8 depicts the data persistence history – the average amount of data replicas residing in the network during the time of the experiment (10000 peers, 20 links on average at each peer).

### 6.6.2. Experiments on PlanetLab

We have also implemented Fuzzynet using the ProtoPeer[3] toolkit. The system is deployed on 330 PlanetLab nodes, all communication between the overlay neighbors is done via TCP and all other communication is done via UDP. For the first 5 minutes we bootstrap the system into a Small-World ring-based topology with Chord-like connectivity. At the 5 minute mark all peers insert 50 random and unique key-value pairs into the overlay. At the 10 minute mark all peers start to periodically look up a randomly chosen key out of the $330 * 50$ inserted. We measure the system-wide failure rate of lookups. A lookup fails if while being routed no greedy

[3] http://protopeer.epfl.ch/

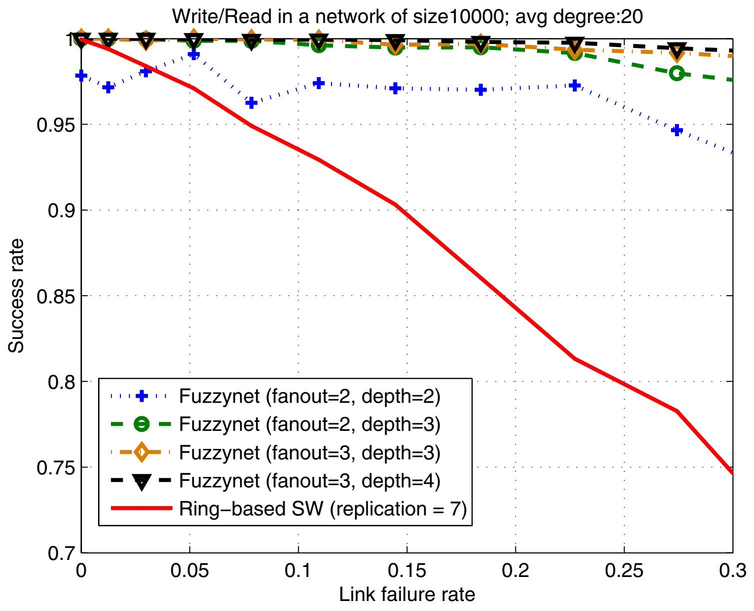


**Figure 6.7.** Fuzzynet success rate as compared to a Ring-Based overlay.

next hop is possible but the current peer does not contain the desired key. We also measure the average length of the lookup paths both for successful and failed lookups. In addition, for each key we count how many times it is replicated, which is summarized as the average, 5th and 95th percentiles of the number of replicas.

We run the experiment in two different setups (Table 6.1): i) A Chord-like write/read algorithms, where there is only one peer responsible for a particular data item and the replication of that data (Chord) and ii) Fuzzynet write/read algorithms with a replication induced by different *depth* and *fanout* values (cf. Section 6.4.1.1). The same experiments are repeated for the network with simulated firewalled peers. Two firewalled (NAT) peers cannot be overlay neighbors since they cannot directly communicate between each other. We set the fraction of firewalled peers to 36%, following the results of the study [Wang *et al.* 2004a] which measured the number of peers behind firewalls and NATs in large-scale public P2P systems.

In our experiments we have observed that, indeed, due to network anomalies even in the absence of NAT a fraction of ring links are missing. The results show that in a real live network deployment the missing ring links cause a considerable number of unsuccessful data insertion operations and, consequently, failed lookups in Chord's case (2.15%). Fuzzynet's write/read algorithms, however, lowers the failure rate by two orders of magnitude (0.01% in the case of D3 F2) only with an average of 6 replicas. Under realistic assumptions on firewalled hosts [Wang *et al.* 2004a], the Chord topology is even more disrupted with 5.2% of failed lookups. The Fuzzynet approach lowers the loss 10-fold, down to 0.47%.

Some of the WriteBurst branches stall and time out, PlanetLab is notorious for its unpredictable delays [Rhea *et al.* 2005a]. We have not implemented any acknowledgements or

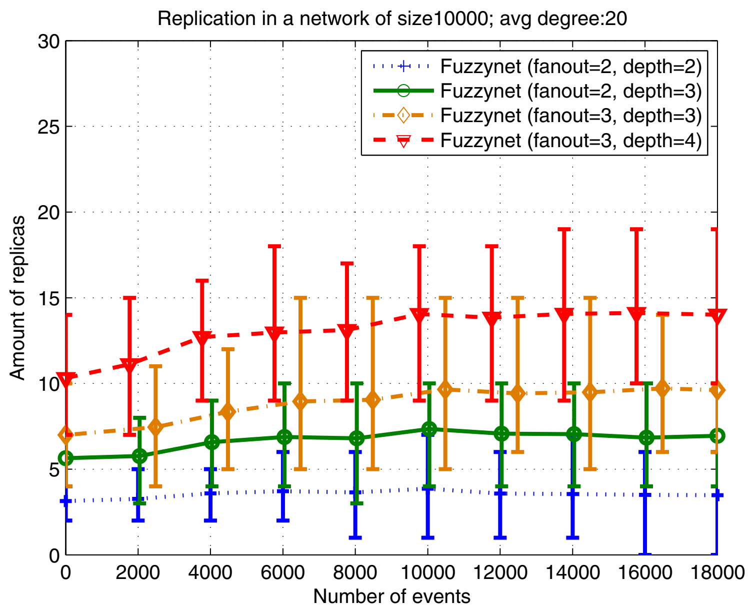


**Figure 6.8.** Data persistence over time. The solid lines represent the average amount of replicas per data item at a particular time slot (the error bars represent the 5th and the 95th percentiles of the replica size distribution).

retries, still our replication scheme is robust under the loss and delay conditions of PlanetLab and enough replicas are created to significantly reduce lookup failures.

## 6.7. Related Systems

The vast majority of structured overlays base their topologies and routing techniques on exact, peer key-dependent core structures like rings [Manku *et al.* 2003; Stoica *et al.* 2001], trees [Aberer 2001; Maymounkov and Mazières 2002], de Bruijn graphs [Ganesan and Pradhan 2003], hypercubes [Schlosser *et al.* 2002], butterfly networks [Malkhi *et al.* 2002], etc. As dis-

| | avg # hops | failed lookups | # Fuzzynet replicas | | |
|---|---|---|---|---|---|
| | | | avg | $5^{th}$ | $95^{th}$ |
| **Chord** | 4.27 | 2.15% | - | - | - |
| **Fuzzynet D2 F2** | 3.70 | 0.15% | 4.38 | 3 | 5 |
| **Fuzzynet D3 F2** | 3.55 | 0.03% | 6.20 | 5 | 7 |
| **Fuzzynet D14 F1** | 3.17 | 0.01% | 14.26 | 8 | 15 |
| **Chord NAT** | 4.44 | 5.20% | - | - | - |
| **Fuzzynet D2 F2 NAT** | 3.79 | 1.81% | 4.64 | 4 | 6 |
| **Fuzzynet D3 F2 NAT** | 3.65 | 0.47% | 6.72 | 5 | 9 |
| **Fuzzynet D14 F1 NAT** | 3.79 | 1.16% | 4.64 | 4 | 6 |

**Table 6.1.** Summary of the PlanetLab results. “D” and “F” represent different depth and fanout values (cf. Section 6.4.1.1)

cussed in the motivation part - all of them to a very high extent lack the flexibility of choosing the neighbors on the close neighborhood level, which makes the maintenance more complicated. A substantial amount of work was devoted to tackle the problem of maintaining the exact structure (e.g., rings) under churn and various stabilization algorithms were developed to keep the core structures alive [Angluin *et al.* 2005; Li *et al.* 2004; Liben-Nowell *et al.* 2002; Rhea *et al.* 2003; Shaker and Reeves 2005]. On the other side, there are unstructured and semi-structured systems [Clarke *et al.* 2001; Ganesan *et al.* 2003; Terpstra *et al.* 2007] based on loose topology which require low maintenance, but are rather inefficient in terms of bandwidth consumption and are suffering from low query recall.

The seminal semi-structured system Freenet [Clarke *et al.* 2001] requires only loose topology and low maintenance cost; however, it has no guarantees that the existing data can be retrieved even in the functioning network. In Fuzzynet data items are placed using the Small-World clusterization, so, even if they are not placed deterministically, it is easy to perform an “update” unlike in Freenet where data items are more widely spread in the overlay.

Another semi-structured system Yappers [Ganesan *et al.* 2003] trades-off between Gnutella-like flooding and DHT routing. In contrast to Fuzzynet's greedy routing - the lookups in Yappers are performed in a broadcast-like fashion. Effectively Yappers is only an improved version of Gnutella network, though it floods smaller fraction of peers than Gnutella itself.

The recently proposed BubbleStorm [Terpstra *et al.* 2007] uses “bubblecast” - a data dissemination and querying strategy based on random walks with flooding over random multigraphs. BubleStrom is designed in such a way that the “data bubble” and the “query bubble” are very likely to intersect. Because of its unstructured nature BubbleStorm requires low maintenance cost; however, the exhaustive flooding-based querying is far too costly compared to Fuzzynet.

To the best of our knowledge Fuzzynet is the first structured overlay based on loose connectivity, which while providing good recall guarantee (in fact, under real networking conditions it is better than the current ring based DHTs) while not requiring any exact-structure maintenance, thus drastically saving on maintenance overheads at a marginal overhead for overlay operations like data read and write.

## 6.8. Conclusions

In this chapter we presented the Fuzzynet technique which allows building structured ringless overlays without requiring any explicit maintenance under churn. Unlike traditional ring-based overlays, Fuzzynet can successfully cope with non-transitivity problems and routing anomalies in realistic networks. Our technique can be employed on loose network topologies, permitting to avoid the eager maintenance strategies of the “heavy”, peer key-dependent structures as e.g., the rings. We show that despite bearing a loose topology, Fuzzynet can still perform efficient publishing (write) and greedy lookups (read) with probabilistic guarantees and surpass ring-based overlays like Chord and Symphony in faulty environments as encountered in real life [Guha *et al.* 2006; Wang *et al.* 2004a]. We have analytically calculated the lower performance bounds of Fuzzynet and evaluated it with simulations, as well as with implementation and deployment of our system on PlanetLab.

We believe that Fuzzynet is ideal for high churn environments, and will successfully fill in the gap between the bandwidth wasting unstructured and high maintenance cost classical-structured overlays that rely on the integrity of the ring, and nevertheless perform worse under real life churn conditions than our probabilistic system based on fuzzy data placement.

Because of the intrinsic flexibility, the same Fuzzynet technique has a potential to address other load-balancing issues, particularly query-load balancing, which can be tackled by changing the replication rate for each individual data item. Also, Fuzzynet can possibly address trust and reputation issues by adapting the "Write-Burst"-like mechanisms to serve as a peer voting technique in an asynchronous environment.

In the following chapter we will show how Small-World based overlays which have an intrinsic property to support key distributions of any complexity can be successfully employed for building efficient publish/subscribe systems.

Chapter 7

# Gravity: An Interest-Aware Publish/Subscribe System

Gravity is a habit that
is hard to shake off.

Terry Pratchett

## 7.1. Overview

In this chapter we consider the problem of efficient data dissemination in a decentralized publish/subscribe (pub/sub) system in the presence of large numbers of topics and arbitrary subscription patterns. Overlay networks are an attractive solution from the scalability standpoint as they provide routing guarantees with limited bandwidth load (fixed node degrees) and only rely on partial membership information for the maintenance. However, the existing overlay construction protocols are typically oblivious to the actual node subscriptions and, as a result, a message published on a certain topic will need to traverse a large number of uninterested peers before reaching all of its subscribers, thus resulting in a high message dissemination cost. In this chapter we introduce Gravity, a pub/sub system that exploits similarity in the individual node subscriptions to build efficient dissemination structures while retaining fixed node degrees. Gravity's topology is based on the Small-World overlay design principles described in Chapter 5. Gravity's key insight is to dynamically cluster the nodes with similar subscriptions by placing them close to each other on the unit ring, thus resulting in a network in which nodes with similar interests are closely connected. The messages are then disseminated over multicast trees that preserve the peer's proximity in the network resulting in a low publication cost. Our thorough experimental study confirms the effectiveness of our system given realistic subscription patterns and shows that Gravity surpasses existing approaches in efficiency by a large margin.

The rest of the chapter is organized as follows. Section 7.2 motivates and describes the current challenges in pub/sub systems. In Section 7.3, we discuss the prior work relevant to Gravity's context. Section 7.4 gives an insight to Gravity's design, while Section 7.6 describes Gravity's main building blocks, which are the node placement strategy and the dissemination tree maintenance algorithms, in more detail. Section 7.7 presents the results of the experimental evaluation. Finally, Section 7.8 concludes the chapter.

## 7.2. Motivation

Publish/Subscribe is a popular communication paradigm [Eugster *et al.* 2003] useful for supporting group-based interaction in distributed settings. A typical pub/sub system supports an abstraction of a logical multicast channel, or *topic*, along with the three basic primitives allowing the users to (1) *subscribe* to a topic of interest, (2) *publish* data on a topic, and (3) receive notifications whenever a new data is posted on some of the topics in their subscription. In this chapter, we introduce *Gravity*, a pub/sub system, specifically designed to support efficient information sharing in large-scale group based applications. In particular, we target two application domains: The first one is the emerging class of massive scale collaborative applications in which large populations of human users collaborate over a large collection of shared artifacts. Examples of such applications include on-line gaming, collaborative editing, live objects [Ostrowski *et al.* 2007], etc. The second one is associated with runtime management of large pools of hardware resources (such as those found in emerging data centers and compute clouds) where a pub/sub system is instrumental to provide the necessary monitoring functionality.

To effectively deal with the problems of scale which are characteristic of the above application domains, Gravity employs a dynamically constructed structured overlay topology as its underlying communication fabric. A novel aspect of Gravity, which sets it apart from the other existing peer-to-peer pub/sub systems [Castro *et al.* 2002; Chockler *et al.* 2007b; Voulgaris *et al.* 2006], is the ability to exploit correlation in the individual peer subscriptions to reduce the communication overhead of the message publication. Intuitively, the necessary pre-requisite for communication efficient message dissemination, is to ensure that the peers interested in the same topics are *well-clustered* in the underlying overlay, that is, separated by a small number of peers not interested in those topics. Ideally, the nodes interested in the same topic $t$ form *strong* clusters, i.e., the subgraph induced by those nodes is connected (in [Chockler *et al.* 2007a], this property is referred as *topic-connectivity*). Clearly, however, strong clustering is impossible to achieve for all possible input subscriptions unless the node degree is unbounded.

In Gravity, we therefore, opt for *best-effort* clustering in which the fixed degree budget is utilized to produce the best possible clustering of the nodes in the overlay. Consequently, the effectiveness of this method for reducing the communication costs is determined by the extent to which the individual peer subscriptions are clustered in the input. As indicated by recent studies of the connectivity patterns exhibited by real-world large-scale pub/sub systems as well as confirmed by our own empirical study of the Gravity performance (cf. Section 7.7), the individual subscriptions do indeed tend to form clusters to the extent sufficient for being effectively exploited even by overlays with relatively small degree budgets. For example, as our experiments show (using Wikipedia editing statistics as the approximation of the subscription patterns), in an Internet-scale pub/sub system show even with a modest network degree of 12, it is possible to achieve 77% improvement in the message dissemination cost compared to a strawman implementation that does not employ similarity-based peer clustering. Hence, the similarity-based clustering could indeed improve performance of peer-to-peer pub/sub systems in realistic settings.

The key idea of Gravity's best-effort clustering algorithm is to ensure that the nodes sharing similar interests will be brought as close together (or "gravitate" towards each other) as it is possible under the given degree constraint. More specifically, we position the nodes subscribed on topic $t$ in a continuous cluster on the ring of structured overlay. Our Gravity technique builds the spanning tree for that topic using greedy routing paths from the root node (which can be any node in the network) to every subscriber in that cluster. Because Gravity's topology is based on a Small-World design we prove (cf. Section 7.5) that for reaching every subscriber greedy routing processes will traverse at most $O(\frac{\log^2 N}{k})$ distinct peers outside the cluster, regardless the size of the cluster itself, where $k$ is the average node degree. Thus, the overhead of disseminating messages in the spanning trees built in such fashion will never exceed $O(\frac{\log^2 N}{k})$ relay peers.

To assess the impact of the subscription similarity-based clustering technique employed by Gravity on the message dissemination cost, we conducted two sets of experiments using a simulated implementation. In both sets of the experiments, our strawman was a pub/sub system similar to [Castro *et al.* 2002] in which the dissemination trees are constructed on top of a ring-based DHT with uniformly chosen peer identifiers.

The goal of the first set of the experiments was to assess the message dissemination cost as a function of various degrees of the subscription correlation in the input. For that, we fed Gravity with synthetically generated subscriptions data using the model described in [Tock *et al.* 2005]. Our experiments show that the improvement in the cost compared to the strawman does indeed depend on the clustering in the input being as high as 10-fold better for strongly clustered subscriptions.

In the second set of the experiments, we used the subscription data collected from two real-world pub/sub systems which we believe are representative of the two application domains being considered. The first data set was the editing statistics for the Wikipedia pages, and the second one, was obtained by logging the subscription requests generated by the clients of the internal monitoring infrastructure employed by the WebSphere[1] application management middleware. The simulations show that Gravity significantly reduces dissemination cost as

[1] www.ibm.com/websphere/

compared to the strawman implementation (cf. Section 7.7.2).

## 7.3. Related Work

The simplest way to cluster subscribers in an overlay network is to maintain a separate communication structure for each individual topic with a non-empty set of subscribers. This approach was realized in [Voulgaris *et al.* 2006] where clustering is achieved by overlaying an unstructured overlay topology with rings spanning the subscribers of each particular topic. The idea of exploiting subscription similarity to reduce the space per node requirements of the subscription-based clustering was introduced in [Chockler *et al.* 2007b] in the context of unstructured overlays. The theoretical feasibility of constructing perfect clustering while minimizing average node degree was studied in [Chockler *et al.* 2007a]. Other pub/sub systems like DPS [Anceaume *et al.* 2006] focus on content filtering which, however, are less relevant for topic-based pub/sub systems, such as Gravity.

Wong et al. [Wong *et al.* 2007] introduce E-llama technique for embedding an edit distance metric into a small world overlay that can potentially be useful for subscription clustering through embedding the topic similarity metric. However, their approach assumes that the distribution of the subscriptions in the input as well as dimensionality thereof can be estimated with a sufficient accuracy ahead of time, which is infeasible in a general purpose pub/sub system.

Topic clustering [Adler *et al.* 2001; Ostrowski *et al.* 2007; Tock *et al.* 2005; Vigfusson *et al.* 2008] looks into amortizing overheads associated with message dissemination in large pub/sub systems by aggregating multiple topics into larger groups (or channels) based on the similarity of their subscriber sets. It was first introduced in [Adler *et al.* 2001] in the context of optimal assignment of multicast groups to multicast addresses, and subsequently generalized to pub/sub in [Tock *et al.* 2005; Vigfusson *et al.* 2008]. The existing solutions to topic clustering rely on approximation techniques (such as K-means [Tock *et al.* 2005]) whose convergence depends on the accurate common knowledge of the current assignment of topics to channels. They are therefore not easily implementable in a decentralized fashion [Shraer *et al.* 2007].

Bayeux [Zhuang *et al.* 2001] and Scribe [Castro *et al.* 2002] are the first multi-group multicast systems based on DHTs. Pastry overlay [Rowstron and Druschel 2001] is used as the underlying topology for Scribe, while Bayeux employs Tapestry [Zhao *et al.* 2004]. Both Bayeux and Scribe are conceptually similar and utilize the existing DHT links to construct and maintain spanning trees on every existing topic for efficient data dissemination. In Bayeux, the spanning trees follow the greedy routing paths from the root node to every subscriber, while in Scribe – from every subscriber to the root node. To balance the bandwidth (node degree) and storage load such pub/sub systems use a uniform hash function to distribute the nodes evenly on the key space. As a consequence, the same interest peers (subscribed to the same topic) are dispersed uniformly on the key space. Such dispersal of similar peers (i.e., the peers which

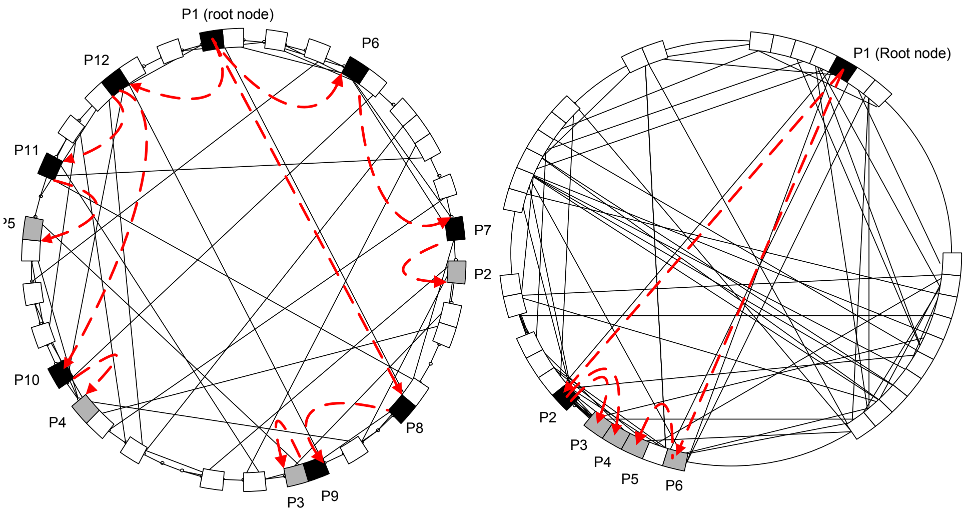


**Figure 7.1.** Spanning trees on uniform (unclustered) and skewed (clustered) structured overlays (black peers represent unsubscribed nodes, grey - subscribed; spanning trees are represented by dashed lines).

exhibit high interest correlation) apart from each other on the key space results in an inclusion of many uninterested nodes to the spanning trees, which in turn leads to a logarithmic factor increase in the message dissemination cost compared to the optimal one. (See Figure 7.1(left) for graphical representation). Since the topic popularity is known to follow a power-law distribution, depending on the distribution parameters and the total number of topics, the low popularity topics can represent as much as 50% of the total topic population[2], which are expected to suffer from high dissemination cost. The dissemination cost can be considerably reduced if the tree-leafs (nodes with the similar interests) are clustered (Figure 7.1(right)). In the following, we will discuss the possibility of employing a peer clustering technique to drastically reduce dissemination cost in pub/sub systems.

## 7.4. Insight to Gravity

Intuitively, the necessary condition for reducing the dissemination cost is to make sure that the peers sharing the same interest form clusters in the overlay. That is, for each topic, any two subscribers to this topic are separated by a small number of peers not interested in this topic. Ideally, the peers interested in the same topic would form a *strong* cluster, that is, the subgraph induced by these nodes is connected. That however, is not always feasible if the nodes have a limited degree budget, which is important for scalability. Our approach to clustering is therefore, based on exploiting similarity in the node subscriptions so that the nodes with closer interests are more likely to be well-clustered than the nodes with far away or disjoint interests. More specifically, we introduce a metric over the node subscription space so that the distance between any two subscription sets is proportional to their intersection size (see Section 7.6.1

[2] E.g., if the topic popularity follows a Zipf distribution with $\alpha = 1$, more than 50% of the topics will have popularity below 10%, if the total number of topics is 1000; and more than 70% for 10000 topics.

for the precise definition of the subscription distance). The subscribers are then organized into a logical ring in a way, that the distance in the subscription space relates to the distance on the ring. More specifically, a newcomer peer is always assigned to become a ring neighbor of the most similar peer in the subscription space. For achieving this, every peer needs the ability to compare its own subscription set to the subscription sets of the peers with overlapping interests, i.e., a sufficient *peer view* has to be available for every peer in the network. There are multiple ways to collect the information on the peer view, e.g., by random sampling, by employing centralized directory approach, etc. We, however, propose an aggregation method where the necessary information on the current peer view is stored in the related spanning trees and the peer view maintenance is performed pro-actively, using the existing tree topology (see Section 7.6.5 for more details). Furthermore, for optimizing peer location on the ring, a number of strategies can be employed; e.g., (i) joining the ring with partial peer view but never changing the location, or (ii) allowing the change of location later on, when more precise information on the peer view is collected. Such a flexible choice of strategies enables Gravity to successfully adapt to many different application scenarios, where different trade-off costs are associated on the topology changes and data dissemination. Our experiments show that Gravity performs reasonably well even with very small peer views (cf. Section 7.7.1.2).

The ring structure is further augmented with a few long range links assigned in a Small-World construction manner so that the probability of creating a link from node $u$ to node $v$ is inversely proportional to the ring-neighbor distance between $u$ and $v$ in the overlay. It is inevitable that peer identifier distributions become arbitrarily skewed due to the fact that the peers are placed on the ring based on their actual subscriptions and not in a uniform fashion. For handling this task we utilize the previously introduced Oscar technique (cf. Chapter 5) for maintaining the underlying topology. As we have shown earlier, Oscar's scalable sampling technique can cope with identifier distributions of any complexity and produce topologies which are proven to belong to the class of routing efficient Small-World networks [Kleinberg 2000]. The topology of Gravity therefore, possesses several important properties of those networks, such as limited degree, logarithmic routing latency, and amenability to a distributed implementation. Furthermore, by construction, the Small-World networks preserve peer clustering, that is, the nodes with closer identifiers are also close in the resulting overlay. We therefore, conclude that the resulting network is also preserving clustering in the interest space, and therefore, satisfies the necessary condition for efficient publication routing.

The only remaining piece is to construct the multicast trees which would preserve the clustering in the overlay. We do that by assigning to each topic a unique *home location*, which is a peer whose identifier is chosen by uniformly hashing the topic name into the node identifier space. The topic home locations can be found efficiently by keeping the nodes in an additional overlay structure on which uniform hashing can be applied (utilizing the same Oscar algorithms). Subsequently, the multicast trees for each topic are rooted in the topic home location, and constructed by following the greedy routing paths from the home location to the topic subscribers. In particular, each new subscriber to a specific topic joins the tree for that topic by following the greedy routing path towards the topic home location until a grafting point in the tree is found, or the topic's home location is reached. We define the grafting point as a node lying on the greedy routing path from the root node to the newcomer node and belonging to the topic tree. At that point, the network is traversed downwards, back to the newcomer peer by following the greedy routing path. Such a tree construction process ensures rapid join and does not overload any single node in the system. The details of the tree construction algorithm are given in Section 7.6.3.

We prove that the trees constructed in this way would indeed preserve the node clustering in the overlay. More specifically we show that the following theorem holds[3]:

**Theorem 7.1.** Given any node $u$ and the continuous range $R$ on the ring of a Small-World based structured overlay, the total amount of the nodes involved in greedy routing processes for reaching every peer in $R$ from $u$, will not exceed $O(r + \frac{\log^2 N}{k})$, where $r = |R|$ and $k$ is the average network degree.

In other words, a spanning tree for topic $t$, consisting of the greedy routing paths from a *home location* to the nodes interested in $t$ and occupying a contiguous range on a ring would include at most $O(\frac{\log^2 N}{k})$ nodes outside of this range, and therefore, dissemination on that tree would involve at most $O(\frac{\log^2 N}{k})$ nodes uninterested in $t$.

This property is especially well suited for spanning tree construction when the applications exhibit strong correlation in the interest subscription patterns allowing large interest clusters to emerge. In reality, the peer clusters might have some gaps in the ranges, which could potentially increase the number of uninterested nodes in the spanning tree's topic, thus increasing the dissemination cost. However, our simulations show that for a realistic setting (cf. Section 7.7) this cost is not significant and Gravity surpasses existing approaches in efficiency by a large margin.

## 7.4.1. Preliminaries and Notation

Before going into the details of our work, we briefly recapitulate the basic concepts and notations introduced in Chapter 3 together with several other notations.

We fix a universe of nodes $P = \{p_1, \ldots, p_n\}$ subscribing to the topics chosen from the set $T = \{t_1, \ldots, t_m\}$. For each topic $t_i$, we write $\lambda_i$ to denote the rate of the traffic being posted on $t_i$. We define an *interest* function $Int$, to be the function mapping $P \times T$ to $\{0, 1\}$ such that $Int(p, t) = 1$ iff $p$ is subscribed to $t$. Likewise, we say that $p$ is *interested* in $t$ iff $Int(p, t) = 1$. Every Gravity peer $p$ bears two keys ($F_{P_{I_{control}}}(p)$ and $F_{P_{I_{skewed}}}(p)$) and belongs to two ring structures: (i) to a control ring in $I_{control}$ space for unambiguously finding

[3] For the proof please refer to Section 7.5.

"home locations" of the topics; and (ii) to a skewed ring in $I_{skewed}$ space for clustering peers according to their interests. Every peer $p$ is responsible for a range on both rings $\mathcal{M}_I^{res}(p) = \big[F_{P_I}(p), F_{P_I}(p_{successor})\big)$, from its own key to the key of its successor on the ring. Every peer $p$ manages all the data items $\zeta$ with identifier $F_R(\zeta) \in \mathcal{M}^{res}(p)$. There exists a distance function $d_{\mathcal{I}}(F_{\mathcal{P}}(u), F_{\mathcal{P}}(v))$ which indicates the distance between a peer $u$ and a peer $v$ in the identifier space $\mathcal{I}$. Furthermore, $p$ maintains a routing table $\rho(p)$ consisting of the links of the $I_{control}$ space and $I_{skewed}$ space, i.e., $\rho(p) = \rho_{control}(p) \bigcup \rho_{skewed}(p)$.

## 7.5. Dissemination Cost on a Range in Small-World networks

Let us name the parallel greedy routing processes from node $p$ reaching all the nodes belonging to a continuous range $R$ a *shower algorithm* on range $R$. The pseudocode of the showering process is given in Algorithm 7.1. Thus, in order to prove Theorem 7.1 we need to show that the number of nodes involved in shower algorithm will not exceed $O(r + \frac{\log^2 N}{k})$.

**Algorithm 7.1** Brodcast showering algorithm in Small-Worlds: $shower(p, R, payload)$

```
 1: if $\mathcal{M}^{res}(p) \cap R \neq \varnothing$ then
 2:     acquire $payload$
 3: end if
 4: $R = R \setminus \mathcal{M}^{res}(p)$
 5: if $R \neq \varnothing$ then
 6:     define $Z(f) \subseteq \mathcal{I}$ such that $\forall f \in \rho(p)$ and $\forall \zeta : F_R(\zeta) \in Z(f), \nexists f' \in \rho(p) \setminus f$ such that
        $d_{\mathcal{I}}(F_{\mathcal{P}}(f'), F_R(\zeta)) < d_{\mathcal{I}}(F_{\mathcal{P}}(f), F_R(\zeta))$
 7:     $Z_R(f) = Z(f) \cap R, \forall f \in \rho(p)$
 8:     define $F \subseteq \rho(p)$ such that $\forall f \in F, \nexists Z_R(f) = \varnothing$
 9:     $shower(f, Z_R(f), payload), \forall f \in F$
10: end if
```

The shower algorithm is executed in the following way. Current showering message holder $p$ acquires the *payload* if it belongs to the range $R$ (line 2). If the peer was responsible for all that range - the algorithm terminates (line 5). Otherwise the algorithm defines the forwarding ranges $Z$ for each routing table entry (finger) $f$, such that greedy routing algorithm issued for identifier $F_R(\zeta)$ would take a path via finger $f$ which is responsible for forwarding into the identifer range $Z(f)$, where $F_R(\zeta) \in Z(f)$ (line 6). If range $R$ does not fall under complete forwarding responsibility of a particular finger then the algorithm partitions range $R$ into smaller chunks $Z_R$, such that every new chunk can be covered by a single forwarding responsibility partition $Z(f)$ of a finger $f$. (line 7). The showering message is then recursively passed to these fingers which are responsible for forwarding the respective $Z_R$ parts of range $R$ (line 9).

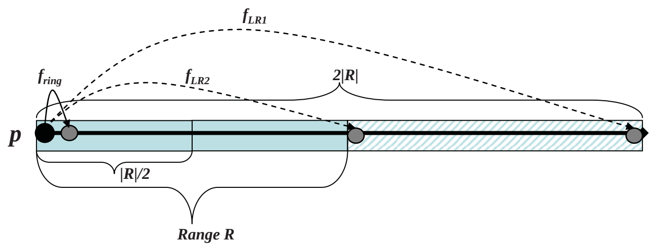


**Figure 7.2.** Visualization of the proof of Theorem 7.1

**Proof of Theorem 7.1** We will calculate the amount of peers involved in the execution process of Algorithm 7.1. Let us assume that peer $p$ issues a range query for a range $R$ of a length $|R|$. Let us also assume that this peer $p$ resides in the range $R$ since because of the navigability property of Small-World networks peer $p$ can be reached in $O(\frac{\log^2 N}{k})$ steps from any node in the network (where $k$ is the average degree of the network).

Let us assume that peer $p$ is on the edge of the range $R$ (see Figure 7.2). Because of the overlay properties peer $p$ will always have at least one link in the range (the ring link $f_{ring}$). The furthest link outside range $R$ which can participate in the next showering phase is $f_{LR1}$, at at most $2|R|$ distance away. The closest link outside range $R$ which can participate in the next showering phase is $f_{LR2}$, at at most $|R|$ distance away. Thus, even if $p$ indeed has a finger $f_{LR2}$ and no more long-range links in the range $R$, $Z_R(f_{ring})$ will have to be at least half of the initial range $R$, i.e., $|Z_R(f_{ring})| \geq |R|/2$.

In the next showering step for the the subrange $Z_R(f_{ring})$ peer $f_{ring}$ is guaranteed not to use any of the existing links outside $|R|$ because this violates the greedy routing principle, since a ring neighbor of $f_{ring}$ would be closer to any point in $Z_R(f_{ring})$ than any possible finger outside $R$. As a result, all the subsequent showering messages for the subpartitions of range $Z_R(f_{ring})$ will not be able to leave the initial range $R$.

Let us denote by $EX$ the expected number of hops that greedy distance minimizing routing will reach partition $Z_{back} = R \backslash Z_R(f_{ring})$ from the current message holder $f_{LR1}$, i.e., the number of hops that takes to halve the distance between $f_{LR1}$ and the furthest peer in range $Z_{back}$. It has been proven in Chapter 4 that $EX$ does not depend on $N$ and is a constant $c$. Thus, the range showering algorithm can leave the range only for a constant number of hops at each showering level (more precisely $2c$, since in the worst case peer $p$ can be in the middle of the range $R$). There are at most $O(\frac{\log^2 N}{k})$ levels, thus the expected number of peers involved in the algorithm, but not belonging to range $R$ will also be only $O(\frac{\log^2 N}{k})$. Together with $r$ peers of range $R$ ($r = |R|$) and $O(\frac{\log^2 N}{k})$ peers which are required to reach the range from the shower originator, the total number of peers that will be involved in the showering process is $O(r + \frac{\log^2 N}{k})$.

*q.e.d.*

## 7.6. Implementation of Gravity

### 7.6.1. The Placement Strategy

In Gravity, the node placement on the ring is determined by pairwise similarity of their subscriptions weighted by the traffic intensity. We capture that formally as follows: Let $p_1$ and $p_2$ be any two nodes. We define the *similarity* between $p_1$ and $p_2$, denoted $sim(p_1, p_2)$, as the weighted size of the intersection of their respective subscriptions normalized by the weighted size of the union thereof. Formally,

$$sim(p_1, p_2) = \frac{\sum_{i=1}^{m}(Int(p_1, t_i) \wedge Int(p_2, t_i))\lambda_i}{\sum_{i=1}^{m}(Int(p_1, t_i) \vee Int(p_2, t_i))\lambda_i} \tag{7.1}$$

Given the above distance function, the node placement algorithm would proceed as follows. Each Gravity node $p$ maintains a variable $view(p) \subseteq Int$, which is a partial view of the nodes' subscriptions known to $p$. By a slight abuse of notation, we write $q \in view(p)$ to denote the fact that $q$ is in the domain of $view(p)$.

The partial views are maintained at the *home locations* of each topic, and are propagated to the other nodes in the system in a lazy fashion using a lightweight gossip protocol. A newly joining node can bootstrap its view using the views available at the nodes through which it is joining the overlay. The view maintenance protocol is discussed in more detail in Section 7.6.5.

**Algorithm 7.2** Peer key acquisition algorithm $F_{P_{I_{skewed}}}(p) = getLocation(p, view(p))$

```
1: if $view(p) \neq \emptyset$ then
2:     find $q$, such that $sim(p, q) = \max\{sim(p, q') : q' \in view(p)\}$
3:     $F_{P_{I_{skewed}}}(p) = mean(F_{P_{I_{skewed}}}(q), F_{P_{I_{skewed}}}(q_{successor}))$
4: else
5:     $F_{P_{I_{skewed}}}(p) = rand$
6: end if
7: return $F_{P_{I_{skewed}}}(p)$
```

Whenever a node $p$ joins Gravity, or changes its interest, it acquires its new location on the *skewed* ring by performing key acquisition algorithm *getLocation* (Algorithm 7.2). The algorithm inspects $view(p)$ to discover a node $q$ such that $sim(p, q) = \max\{sim(p, q') : q' \in view(p)\}$ (if several such nodes $q$ exist, then one of them is chosen uniformly at random). Node $p$ then joins the ring between $q$ and the $q$'s ring successor. If $p$ fails to contact $q$ (e.g., due to a failure), then $q$ is excluded from $view(p)$, and the entire join algorithm is executed once again. If at some point, $view(p)$ becomes $\emptyset$, then $p$ will join the ring at an arbitrary location. If later on, $view(p)$ gets populated, $p$ can attempt to improve its location by re-executing the join protocol.

### 7.6.2. The Overlay Construction

In order to enable efficient routing, in addition to the two ring links, each Gravity node is also maintaining connections (fingers) to a few (usually $O(\log N)$ ) additional long-range neighbors. In our implementation we use the Oscar technique described in Chapter 5 for link creation among peers. However, Gravity is a general technique and can be used on top of any other overlay which can support non-uniform key distributions and which topology exhibits Small-World properties.

### 7.6.3. Spanning Tree Construction

As discussed earlier, in Gravity, a spanning tree for topic $t$ is constructed using the paths which are employed by the showering process on range $R$ (described in Algorithm 7.1), where the peers interested in $t$ belong to one or several ($n$) continuous ranges $R_1, R_2, .., R_n$ on the ring, such that $R = \bigcup_{i=1}^{n} R_i$. In such a way, a spanning tree for topic $t$ involves only a small fraction of peers not interested in $t$ (cf. Theorem 7.1). For building such a dissemination structure, a joining node needs to find a grafting node, from which the spanning tree would be built using simple greedy routing path towards the joining node. The grafting point is either the *home location* of the topic or a least common ancestor responsible for routing to the id of the joining node. Thus, the spanning tree joining protocol consists of three phases (Algorithm 7.3): (i) finding a spanning tree node, (ii) finding a grafting point on the tree and (iii) grafting a new branch on it.

**Algorithm 7.3** Spanning tree join algorithm $joinTree(p_{joining}, t)$

```
1: $p_s = findTreeNode(p_{joining}, t)$;
2: $p_g = findGraftingPoint(p_s, p_{joining}, t)$;
3: $graft(p_g, p_{joining}, t)$;
```

After a newcomer node $p_{joining}$ decides on a key using the above mentioned *getLocation* algorithm (Algorithm 7.2), it needs to join the spanning trees for every topic it is interested in. To find a spanning tree node $p_s$, the joining node $p_{joining}$ issues a look-up request in the control space $I_{control}$ on the hash value of the interested topic $t$ (Algorithm 7.4). The lookup traverses the network towards the "home location", i.e., to a peer responsible for $hash(t)$. On the way towards the target, the lookup either encounters some node belonging to the spanning tree of topic $t$, or it arrives at the root of the tree (i.e., the home location responsible for $hash(t)$ - the target of the lookup request).

Having reached a spanning tree node, the join process enters the phase (ii) by continuing traversing towards the root node on the spanning tree until the grafting point is found. The grafting point is either the most common ancestor which has a spanning tree link responsible for the key of $p_{joining}$ in $I_{skewed}$ or the root node itself (Algorithm 7.5). It is important to mention, that every node in the pub/sub system knows which ranges of the key space each

**Algorithm 7.4** Finding a node on a spanning tree $p_s = findTreeNode(p, t)$;

```
1: if $t \notin topics(p) \wedge hash(t) \notin \mathcal{M}^{res}_{I_{control}}(p)$ then
2:    $p_{next} = p' \in \rho(p) : d(F_{P_{I_{control}}}(p'), hash(t)) < d(F_{P_{I_{control}}}(p''), hash(t)) \forall p'' \in \rho(p) \backslash p'$
3:    $p = findTreeNode(p_{next}, t)$;
4: end if
5: return $p$
```

long-range link (finger) is responsible for, i.e., which link is used if a routing request on a particular key arrives (it is a universal property of structured P2P overlays used for greedy routing). Hence, every node on a spanning tree knows which are the ranges of the key space $I_{skewed}$ which its child nodes of the spanning tree are responsible for.

**Algorithm 7.5** Finding a grafting point on the spanning tree $p_g = findGraftingPoint(p, p_{joining}, t)$;

```
1: $p_{closest} = p' \in \rho(p) : d(F_{P_{I_{skewed}}}(p'), F_{P_{I_{skewed}}})(p_{joining})) <$
   $d(F_{P_{I_{skewed}}}(p''), F_{P_{I_{skewed}}}(p_{joining}) \forall p'' \in \rho(p) \backslash p'$
2: if $(p_{closest} \in spanningTreeNodes(t)) \vee (hash(t) \in \mathcal{M}^{res}_{I_{control}}(p))$ then
3:    return $p$
4: else
5:    $p_{parent} = parent(p, t)$;
6:    $p = findGraftingPoint(p_{parent}, p_{joining}, t)$;
7: end if
```

Once on the grafting point, the join process proceeds to the phase (iii) to graft a new branch to the spanning tree with $p_{joining}$ as a leaf node of that branch. It is done by a greedy route message issued from the grafting node towards the newcomers key in $I_{skewed}$ requesting all the nodes on the path to join the spanning tree (Algorithm 7.6).

**Algorithm 7.6** Grafting a spanning tree branch $graft(p, p_{joining}, t)$;

```
1: $p_{closest} = p' \in \rho(p) : d(F_{P_{I_{skewed}}}(p'), F_{P_{I_{skewed}}}(p_{joining})) <$
   $d(F_{P_{I_{skewed}}}(p''), F_{P_{I_{skewed}}}(p_{joining})) \forall p'' \in \rho(p) \backslash p'$
2: add $p_{closest}$ as a child of $p$ in the spanning tree of $t$;
3: if $p_{closest} \neq p_{joining}$ then
4:    $graft(p_{closest}, p_{joining}, t)$;
5: end if
```

**Spanning tree maintenance.** The spanning tree is maintained with the heartbeat messages periodically propagating in the tree, ensuring the early discovery of inaccessible parent nodes. Once a child node discovers that it cannot access its parent, it issues a new *joinTree* request which reconnects the node to the nearest available branch of the tree.

### 7.6.4. Gravity Join

To join Gravity network a newcomer peer executes *gravityJoin* (Algorithm 7.7) algorithm which utilizes the aforementioned components. Initially a joining peer $p$ joins the control ring at a random location (line 1) and acquires the knowledge on the existing topics (line 2) by updating its peer *view*. According to the *view* the peer then gets a suggestive location of the place to join in the skewed ring (line 3). After joining the skewed ring (line 4), peer $p$ becomes a member of all the spanning trees to which topics it is interested in (line 5).

**Algorithm 7.7** Join algorithm of gravity $gravityJoin(p)$

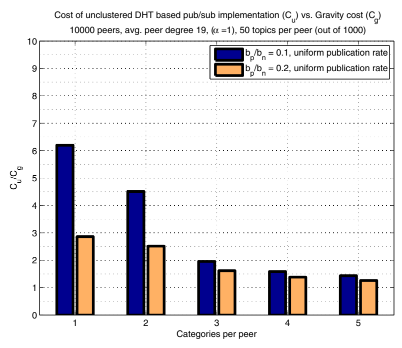
```
1: Join $I_{control}$ ring at random location and establish Oscar connectivity with $\rho_{control}$
   connections.
2: acquire the information about the existing topics $view(p)$
3: $F_{P_{I_{skewed}}}(p) = getLocation(p, view(p))$
4: Join $I_{skewed}$ ring at $F_{P_{I_{skewed}}}(p)$ location and establish Oscar connectivity with $\rho_{skewed}$
   connections.
5: $\forall t \in T(p), joinTree(p,t)$
```

### 7.6.5. Data Aggregation on the Spanning Trees

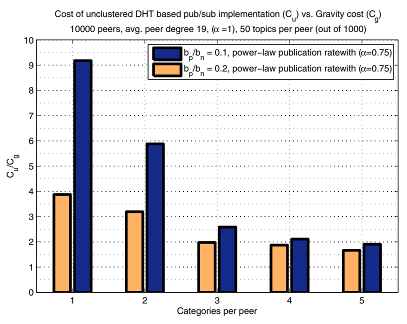
As one of the solutions for peer *view* acquisition we propose for every node on a spanning tree for topic $t$ to bear an information about a subset of peers interested in $t$. Specifically, a leaf node of the tree for topic $t$ propagates towards the root the info about itself and all its interests, while the intermediate nodes propagate upwards only the info on a random subset of $k$ peers $P_k$. Upon receiving this aggregated information about the subscribers, the root propagates it downwards, hence, making every node in the spanning tree to acquire a peer *view* consisting of a set of $k$ random peers of the spanning tree. The propagation of this information can be done in a lazy fashion, and updated periodically with the spanning tree maintenance messages. In such a way no node in the spanning tree is overloaded. Our simulations show that the subset $k$ can be as small as 1 peer (cf. Section 7.7.1.2).

## 7.7. Performance Evaluation

We implemented Gravity in a simulated setting and conducted extensive experimental study to validate our performance and scalability claims. In our simulations Gravity was built on a Oscar-based skewed structured overlay. We compared our system to a strawman – a regular, unclustered DHT based implementation (similar that of Scribe [Castro *et al.* 2002]). In the first part of our experiments we were using synthesized subscription models that have been demonstrated in the prior work [Liu *et al.* 2005; Tock *et al.* 2005] to be a truthful representation of correlated subscription patterns occurring in many real-world scenarios. Meanwhile, in the



(a) Uniform topic publication rate



(b) Power-law topic publication rate ($\alpha = 0.75$)

**Figure 7.3.** Adaptivity to the subscription correlation

second part of the experiments we used actual subscription models, namely Wikipedia users' editing patterns and and pub/sub interest data set collected from IBM WebSphere[4] application management middleware.

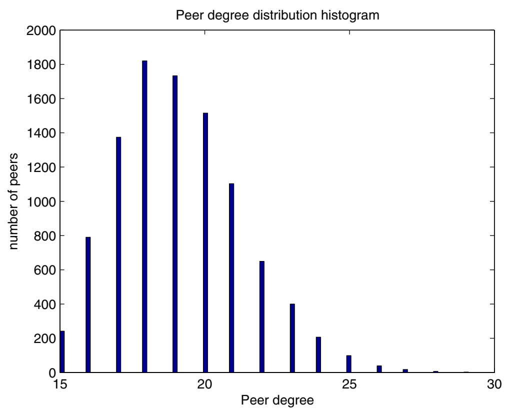
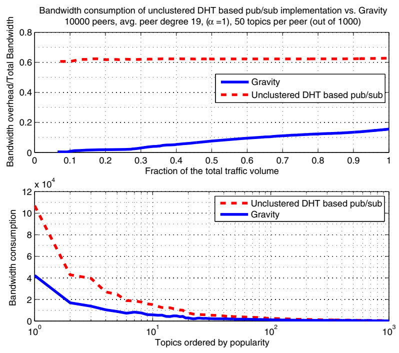


**Figure 7.4.** Peer degree distribution histogram.

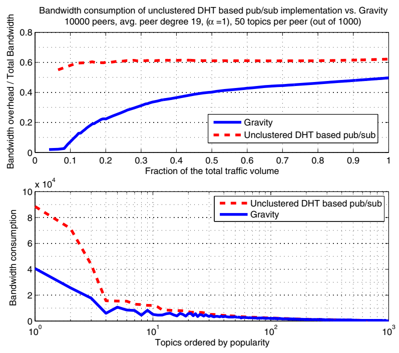
### 7.7.1. Simulations using Synthesized Subscription Models

Unless stated otherwise, for the first part of the simulations we were experimenting with the network, consisting of $10'000$ nodes, where each node had on average 19 links (the peer degree distribution is depicted in Figure 7.4) and Oscar sampling parameter (number of random-walks) was $k = 5$. Every node could subscribe to a subset of 1000 unique topics. The Gravity algorithm used a peer *view* of 10 samples as described in the section on data aggregation on the spanning trees (cf. Section 7.6.5). Throughout the simulations Gravity was consistently outperforming the strawman approach by considerable margins. In the following we will discuss

[4] www.ibm.com/websphere/



(a) Most correlated scenario ($b_p = 1$, $b_n = 10$)



(b) Least correlated scenario ($b_p = 5$, $b_n = 25$)

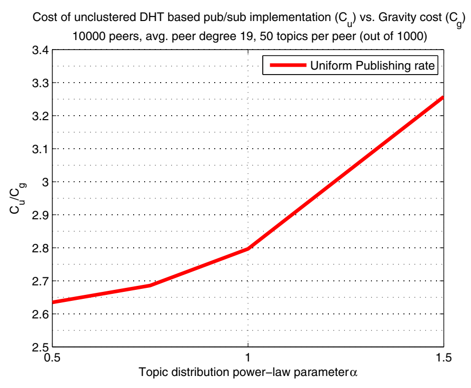
**Figure 7.5.** Bandwidth consumption.

the simulations in more detail.

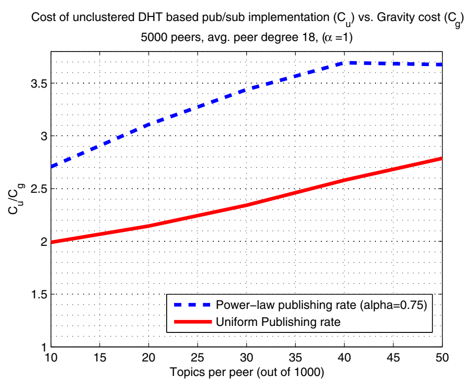
To capture the correlation among the peer subscriptions we utilized the category-model described in [Tock *et al.* 2005]. The overall collection of topics was first partitioned into a fixed number $b_n$ of fixed size categories, and then, every peer had to choose $b_p$ categories u.a.r. out of $b_n$. Within each individual category the topics were assigned based on a power-law distribution. In such a way we capture the peer subscription correlation behavior while avoiding the shortcomings of the simple power-law model, namely the overpopulation of the nodes with the most popular topics. The maximum amount of nodes that the most popular topic can have is $b_p * |b_n|_{max}$, where $|b_n|_{max}$ is the size of the biggest category. The degree of correlation between the peer interests can be adjusted by changing the $b_p$ and $b_n$ parameters while keeping the $b_p/b_n$ ratio intact. That is, the resulting topic frequencies will be the same, however, the correlation between the peer subscriptions will be the highest when $b_p \to 1$ and the lowest (uncorrelated) when $b_p \to b_n$.

### 7.7.1.1. Adaptivity to the Subscription Correlation

We performed extensive simulations to verify whether Gravity can exploit the subscription correlation in the system. We used the above described category model and fixed the ratio of $b_p/b_n$ to 0.1 and 0.2. We were changing the $b_p$ and $b_n$ values from the most correlated (every peer chooses one category out of 10 and 5 categories respectively) to the least correlated (5 categories out of 50 and 25 respectively). Following this model, every peer had been assigned 50 unique topics on average. We have tested our system with different publication rates for each topic. In one experiment every topic was assigned a different publication rate, which was drawn from a power-law distribution with $\alpha = 0.75$, while in the other the publication rate was uniform. Figure 7.3 shows that Gravity outperforms strawman implementation with both publication rate scenarios even with very low subscription correlations. As expected, Gravity performed better with non-uniform publication rate, because the rate is always taken into



(a) Gravity performance for different topic distribution scenarios.



(b) Subscription size influence on the system's performance

**Figure 7.6.** Other experiments.

account by Gravity's peer placement algorithms upon calculating similarity distance among the peers.

Figure 7.5 shows the bandwidth consumption of Gravity as compared to the unclustered DHT based implementation for the most correlated (Figure 7.5(a), $b_p = 1$, $b_n = 10$) and the least correlated case (Figure 7.5(b), $b_p = 5$, $b_n = 25$) with the power-law topic publication rate ($\alpha = 0.75$). The expected bandwidth consumption volume is calculated as $|t|\lambda_t$, i.e., amount of peers involved for publishing on a topic $t$, normalized by the expected publication rate on that topic $\lambda_t$. The graphs reveal almost no overhead for the most popular topics, thus confirming that Gravity is able to do efficient peer placement in the identifier space, which in turn results in compact, bandwidth-efficient spanning trees.

### 7.7.1.2. Adaptivity to Incomplete Knowledge

In our experiments we also show that Gravity is robust enough to deal with the insufficient or partial information on the existing topic information, i.e., not complete peer *view*. We performed two sets of simulations, one with a peer *view* representing a uniform sample of the whole population (e.g., a sample collected with random walkers) and the second a sample representing a random subset of the related topics, as described in Section 7.6.5. Our simulations reveal (Figure 7.7) that Gravity can make the right peer placement decisions with as low as only 1 sample for each related topic per peer.

### 7.7.1.3. Other Results

We have also measured Gravity's performance given different subscription sizes at each peer and the impact of different topic distribution scenarios, where we were changing $\alpha$ parameter of the power-law distribution, which influences the amount of the most popular topics in the system. Figure 7.6(a) shows the performance of Gravity given different $\alpha$ values with the category model

($b_p = 1$ and $b_n = 5$) and uniform publication rate. Naturally, Gravity performs better with the higher $\alpha$ values, since steep power-law function has an inherit correlation quality, i.e., more peers tend to choose the same popular topics, thus increasing the topic correlation among them. Figure 7.6(b) shows the performance of Gravity with uniform publication rate as compared to the non-uniform (power-law with $\alpha = 0.75$) one while changing the peer subscription sizes. The results show that Gravity's algorithms consistently outperforming unclustered DHT based pub/sub systems given any subscription size.

## 7.7.2. Simulations using Wikipedia and WebSphere Subscription Patterns

In the second part of the simulations we have analyzed the behavior of Gravity performance under realistic pub/sub scenarios. We used the trace of Wikipedia edits as the subscription pattern inputs for our simulations. In this experimental setup every entry of the encyclopedia was treated as a topic in our pub/sub system and unique wiki-users handled as Gravity peers. The user edits on the encyclopedia entries were in turn interpreted as the respective peer interests.

For our experiments we have selected 3000 random topics from the whole encyclopedia set, which were edited by nearly $10'000$ unique users. The topic popularity varied from 1 to 348 subscribers per topic, and on average every topic had 5.4 subscribers. Such subscription data was fed to Gravity and to a pub/sub system based on unclustered DHT. Average peer degree for both P2P networks was set to 12. We have measured the cost of publishing for each of the topics in the network. The experiments showed that strawman implementation on average involved 77% more uninterested peers for data publication than Gravity.

We have also obtained the subscription requests generated by the clients of the internal monitoring infrastructure employed by the IBM WebSphere application management middleware. From these WebSphere data traces we had extracted 6145 topics and 79 users. On average every topic had 10.25 subscriptions, however, the divergence was quite extreme – there

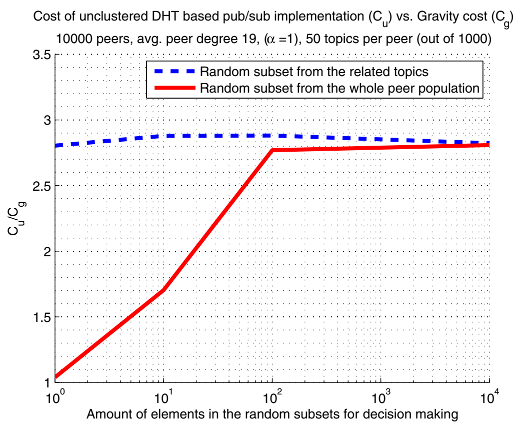


**Figure 7.7.** Adaptivity to incomplete knowledge.

were topics with only one subscriber, as well as the topics populated on all the peers. Average peer degree for Gravity and for our strawman implementation was set to 6. After the publishing on all the topics, the experiments showed that unclustered DHT had on average to route through 37% more uninterested peers than Gravity.

## 7.8. Conclusions

In this chapter we have presented Gravity, a Small-World overlay based infrastructure for scalable topic-based pub/sub. Gravity takes advantage of the existing subscription correlation patterns among the pub/sub users. The proposed technique allows the clusters of similar interest peers to form in the underlying topology which enables the construction of efficient dissemination structures (spanning trees). Since clustering process leads to the non-uniform peer identifer distributions, Gravity employs previously introduced Oscar overlay (cf. Chapter 5) as the underlying topology. Because of the inherent Small-World design, Gravity scales well with the number of nodes, and ensures fixed network degree regardless the number of topics or the size of subscriptions.

In our experimental study we show that Gravity is able to achieve significant reduction in the message dissemination cost, which under certain combination of parameters, can be as high as 10-fold compared to the pub/sub systems based on DHTs with uniform node placement (such as [Castro *et al.* 2002]). Furthermore, we demonstrate that this cost reduction is *adaptive* to both the extent to which the individual node subscriptions correlate, and the amount of information about the other node subscriptions available to each node. In particular, in the worst case scenario, when the subscriptions are completely uncorrelated and/or unknown, the message dissemination cost would not be worse as that of the uniform DHT based systems.

# Chapter 8

# Conclusions

Every solution breeds
new problems.

ARTHUR BLOCH

The main focus of this thesis was to look into the design of the peer-to-peer systems from the perspective of Small-Worlds. We have presented several novel overlay network design solutions which harness the most useful properties of navigable Small-World topologies in order to address many important problems in the field of data-oriented peer-to-peer systems. In this thesis we have mostly concentrated on issues of resource load-balancing, effective data dissemination and reduction of overlay maintenance costs.

To address these problems, we have presented a theoretical model for constructing structured overlays that supports non-uniform hash functions, which builds on the seminal research of Jon Kleinberg [Kleinberg 2000] on navigable Small-World networks. Based on that model, we designed a peer-to-peer system called Oscar, which besides supporting key distributions of any complexity, has also the ability of seamlessly adapting to heterogenous peer environments. We base our Oscar construction algorithms on a novel scalable sampling technique, supported by a rigorous theoretical analysis. The experiments on Oscar validate the theory and show that Oscar deals well with load-balancing issues.

Furthermore, we argue that the utilization of structured overlays which support non-uniform key distributions is not limited to only supporting complex queries in data-oriented peer-to-peer systems. We show that the flexibility of Small-World based overlays like Oscar can be successfully exploited to address the problem of efficient data dissemination. Our novel publish/subscribe system called Gravity is a good example of such design solution and is verified to be highly effective for large-scale data dissemination tasks.

In the thesis we have also approached the largely unexplored issue of abandoning maintenance intensive topology structures (e.g., rings, hypercubes, toruses etc.) in structured overlays. With our Fuzzynet technique, we show how by using the model of Small-Worlds it is possible to drop the reliance on the widely used ring structure, while still retaining all the necessary properties of structured overlays.

## 8.1. Outlook

The overlay network design solutions presented in this thesis have well defined identifier spaces, i.e., it is assumed that every peer has an unambiguous view of the identifier space in which the decentralized routing is performed. However, more and more often, new types of applications emerge where the identifier space is not, or cannot be, defined precisely. For example, for applications based on real-life social networks it becomes increasingly difficult to identify the dimensionalities and the distance functions of the identifier spaces in which the networks are embedded. Designing efficient navigational algorithms that are effective in such type of environments is essential to produce even more flexible peer-to-peer systems which would be well adapted for the upcoming Web 3.0 applications of the future.

There are a number of other aspects which have to be addressed as well. One of them is the locality-awareness problem in building peer-to-peer systems, which becomes increasingly important in many real-life decentralized systems. Addressing this issue can result in a significant boost of the performance of the peer-to-peer systems. Moreover, security and trust are critical issues which should be considered in the design of peer-to-peer systems. Large-scale systems are extremely vulnerable to the behavior of malicious or uncooperative peers, which ranges from innocuous free-riding problem to malevolent Byzantine behavior or Sybil attacks. Failure to address them might render many otherwise effective overlay networks nonoperational.

Although the aforementioned problems were not explicitly touched in this thesis, there are good reasons to believe that the ideas behind Small-World based algorithms which we used to design flexible heterogenous peer-to-peer systems and to reduce their maintenance cost, can be also potentially applied for addressing locality, trust and reputation issues.

# Bibliography

K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001. @pages 4, 9, 14, 16, 83, 93

K. Aberer, A. Datta, and M. Hauswirth. Route maintenance overheads in DHT overlays. In *6th Workshop on Distributed Data and Structures (WDAS)*, 2004. @pages 30

K. Aberer, A. Datta, and M. Hauswirth. Multifaceted Simultaneous Load Balancing in DHT-based P2P systems: A new game with old balls and bins. In *Self-* Properties in Complex Information Systems, "Hot Topics" serices, Lecture Notes in Computer Science*. Springer Verlag(to be published), 2005a. @pages 42, 48

K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 685–696. VLDB Endowment, 2005b. ISBN 1-59593-154-6. @pages 23, 26, 35, 41, 42, 46, 54

M. Adler, Z. Ge, J. F. Kurose, D. F. Towsley, and S. Zabele. Channelization problem in large scale data dissemination. In *ICNP 2001: Proceedings of the Ninth International Conference on Network Protocols*, page 100, Washington, DC, USA, 2001. IEEE Computer Society. @pages 100

L. O. Alima, S. El-Ansary, P. Brand, and S. Haridi. DKS(N,k,f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, 2003. @pages 4, 6, 14, 23, 26, 35

E. Anceaume, M. Gradinariu, A. K. Datta, G. Simon, and A. Virgillito. A semantic overlay for self- peer-to-peer publish/subscribe. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 22, 2006. ISBN 0-7695-2540-7. @pages 100

S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, Dec. 2004. ISSN 0360-0300. @pages 13

D. Angluin, J. Aspnes, J. Chen, Y. Wu, and Y. Yin. Fast construction of overlay networks. In *In 17th ACM Symposium on Parallelism in Algorithms and Architectures(SPAA 2005),Las Vegas, NV, USA*, 2005. @pages 61, 94

J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load balancing and locality in range-queriable data structures. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 115–124, 2004. ISBN 1-58113-802-4. @pages 18, 54, 79

J. Aspnes and G. Shah. Skip graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics. ISBN 0-89871-538-5. @pages 18, 54, 78

F. Banaei-Kashani and C. Shahabi. Swam: A family of access methods for similarity-search in peer-to-peer data networks. In *Information and Knowledge Management (CIKM'04), Washington D.C.*, 2004. @pages 43

D. Barbella, G. Kachergis, D. Liben-Nowell, A. Sallstrom, and B. Sowell. Depth of field and cautious-greedy routing in social networks. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC'07)*, pages 574–586, Dec. 2007. @pages 55

L. Barrière, P. Fraigniaud, E. Kranakis, and D. Krizanc. Efficient routing in networks with long range contacts. In *DISC '01: Proceedings of the 15th International Conference on Distributed Computing*, pages 270–284, London, UK, 2001. Springer-Verlag. ISBN 3-540-42605-1. @pages 57, 58

R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker. Totalrecall: System support for automated availability management. In *The ACM/USENIX Symposium on Networked Systems Design and Implementation*, 2004. @pages 82

A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, 34(4):353–366, 2004. ISSN 0146-4833. @pages 9, 17, 18, 42, 43, 50, 54, 55, 58, 63, 69, 70, 76, 79, 83

S. Bhola, R. E. Strom, S. Bagchi, Y. Zhao, and J. S. Auerbach. Exactly-once delivery in a content-based publish-subscribe system. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 7–16, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1597-5. @pages 7

M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: a large-scale and decentralized application-level multicast infrastructure. *IEEE J. Selected Areas in Comm. (JSAC)*, 20(8):1489–1499, 2002. @pages 7, 14, 98, 99, 100, 109, 114

Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, 2003. ISBN 1-58113-735-4. @pages 15

G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 109–118, 2007a. ISBN 978-1-59593-616-5. @pages 98, 100

G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg. SpiderCast: A Scalable Interest-Aware Overlay for Topic-Based Pub/Sub Communication. In *1th International Conference on Distributed Event-Based Systems (DEBS)*. ACM, 6 2007b. @pages 98, 100

I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, 2001. @pages 4, 16, 23, 35, 94

F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. *SIGOPS Oper. Syst. Rev.*, 35(5):202–215, 2001. ISSN 0163-5980. @pages 54

F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, February 2003. @pages 22

A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003. @pages 82, 83

D. Dolev, E. N. Hoch, and R. van Renesse. Self-stabilizing and byzantine-tolerant overlay network. In *Principles of Distributed Systems, 11th International Conference (OPODIS)*, pages 343–357, 2007. @pages 31

P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003. @pages 7, 98

M. Franceschetti and R. Meester. Navigation in small world networks a scale-free continuum model. *Journal of Applied Probability*, 43(4):1173–1180, 2006. @pages 43

M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica. Non-transitive connectivity and dhts. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, pages 10–10, Berkeley, CA, USA, 2005. USENIX Association. @pages 6, 76

W. Galuba and K. Aberer. Generic emergent overlays in arbitrary peer identifier spaces. In *2nd International Workshop on Self-Organizing Systems (IWSOS 2007)*, volume 4725, pages 88–102, 2007. @pages 79, 83

E. Ganesan and D. K. Pradhan. Wormhole Routing in De Bruijn Networks and Hyper-DeBruijn Networks. In *ISCAS*, 2003. @pages 93

P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 444–455. VLDB Endowment, 2004. ISBN 0-12-088469-0. @pages 18, 48, 54, 61, 79

P. Ganesan, Q. Sun, and H. Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitrary topology. In *IEEE INFOCOM'03, San Francisco, USA*, 2003. @pages 15, 94

A. Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD thesis, KTH—Royal Institute of Technology. @pages 61

A. Ghodsi, L. O. Alima, and S. Haridi. Low-bandwidth topology maintenance for robustness in structured overlay networks. In *Proceedings of the 38st Annual Hawaii International Conference on System Sciences (HICSS)*. IEEE Computer Society Press, 2004. @pages 30

S. Girdzijauskas, A. Datta, and K. Aberer. On small world graphs in non-uniformly distributed key spaces. In *ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops*, page 1187, 2005. ISBN 0-7695-2657-8. @pages 37, 58, 71, 86

S. Girdzijauskas, A. Datta, and K. Aberer. Oscar: Small-world overlay for realistic key distributions. In *The Fourth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), September 11, 2006, Seoul, Korea*, 2006. @pages 55, 60, 87

S. Girdzijauskas, A. Datta, and K. Aberer. Oscar: A data-oriented overlay for heterogeneous environments. In *The 23rd International Conference on Data Engineering (ICDE), April 16-20, 2007, Istanbul, Turkey*, 2007. @pages 76

B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured P2P systems. In *Proc. IEEE INFOCOM*, Hong Kong, 2004. URL `citeseer.ist.psu.edu/surana04load.html`. @pages 61

L. Gong. JXTA: A Network Programming Environment. *IEEE Internet Computing*, 5(3): 88–95, May/June 2001. @pages 22

R. Guerraoui, S. B. Handurukande, K. Huguenin, A.-M. Kermarrec, F. L. Fessant, and E. Riviere. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. *Peer-to-Peer Computing, IEEE International Conference on*, 0:12–22, 2006. @pages 54

S. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer voip system. In *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS06)*, 2006. @pages 77, 90, 95

K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394, 2003. ISBN 1-58113-735-4. @pages 22, 42

N. J. A. Harvey, Jones, M. B., S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USITS 2003, Seattle, WA*, March 2003. @pages 16, 18, 54, 78

W. Hoeffding. Probability inequalities for sums of bounded random variables. In *Journal of the American Statistical Association, 58, 13-30*, 1963. @pages 66

K. Y. Hui, J. C. Lui, and D. K. Yau. Small-world overlay p2p networks: Construction and handling dynamic flash crowd. In *Computer Networks Journal, 50(15)*, October, 2006. @pages 54

F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *In 2nd International Peer To Peer Systems Workshop (IPTPS)*, 2003. @pages 16

D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 36–43, 2004. ISBN 1-58113-840-7. @pages 61

J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, 2000. ISBN 1-58113-184-4. @pages 3, 9, 29, 37, 41, 43, 55, 57, 66, 77, 78, 102, 115

F. Klemm, S. Girdzijauskas, J.-Y. Le Boudec, and K. Aberer. On Routing in Distributed Hash Tables. In *The Seventh IEEE International Conference on Peer-to-Peer Computing*, 2007. URL `http://www.p2p2007.org`. @pages 78, 83

J. S. Kong and V. P. Roychowdhury. Price of structured routing and its mitigation in p2p systems under churn. In *P2P '07: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing, Galway, Ireland*, pages 97–104, 2007. ISBN 0-7695-2986-0. @pages 76

X. Li, J. Misra, and C. G. Plaxton. Active and concurrent topology maintenance. In *In Proc. 18th Ann. Conference on Distributed Computing (DISC*, pages 320–334. Springer, 2004. @pages 61, 94

D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 233–242, 2002. ISBN 1-58113-485-1. @pages 61, 94

W. Litwin, M. anne Neimat, and D. A. Schneider. Lh*-a scalable, distributed data structure. *ACM Trans. on Database Systems*, 21:480–525, 1996. @pages 18

H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *Internet Measurement Conference (IMC), Berkeley*, October 2005. @pages 109

L. Liu, S. Mackin, and N. Antonopoulos. Small world architecture for peer-to-peer networks. In *WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, pages 451–454, 2006. ISBN 0-7695-2749-3. @pages 15

Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, 2002. ISBN 1-58113-483-5. @pages 19

D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002. @pages 16, 78, 93

G. S. Manku. Routing networks for distributed hash tables. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing (PODC)*, pages 133–142. ACM Press, 2003. @pages 42, 50

G. S. Manku, M. Bawa, P. Raghavan, and V. Inc. Symphony: Distributed hashing in a small world. In *In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003. @pages 4, 6, 16, 35, 43, 47, 50, 54, 56, 57, 58, 71, 76, 79, 83, 90, 93

G. S. Manku, M. Naor, and U. Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized p2p networks. In *36th ACM Symposium on Theory of Computing, STOC 2004, p 54-63*, 2004. @pages 50, 55

P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '02: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag. ISBN 3-540-44179-4. @pages 16, 20, 41, 93

S. Milgram. The small world problem. In *Psychology Today 1, 61*, 1967. @pages 42

A. Mislove, A. Post, A. Haeberlen, and P. Druschel. Experiences in building and operating epost, a reliable peer-to-peer application. In *EuroSys '06: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 147–159, 2006. ISBN 1-59593-322-0. @pages 76, 90

M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1:226–251, 2001. @pages 29

M. Mitzenmacher, A. W. Richa, and R. Sitaraman. The power of two random choices: a survey of techniques and results. In *in Handbook of Randomized Computing*, pages 255–312. Kluwer, 2001. @pages 70

K. Ostrowski, K. Birman, and D. Dolev. Live distributed objects: Enabling the active web. *Internet Computing, IEEE*, 11(6):72–78, Nov.-Dec. 2007. @pages 7, 98, 100

K. Ostrowski, K. Birman, and D. Dolev. QuickSilver Scalable Multicast (QSM). In *Proceedings of 7th IEEE International Symposium on Network Computing and Applications (IEEE NCA)*, 2008. @pages 7

I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Scalable Peer-to-Peer Web Retrieval with Highly Discriminative Keys. In *ICDE'07: Proceedings of the 23nd International Conference on Data Engineering*, Istambul, Turkey, 2007. @pages 14

A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003. @pages 61

S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev.*, 31(4):161–172, 2001. ISSN 0146-4833. @pages 16, 22, 23, 26, 42, 54

S. Rhea, B.-G. Chun, J. Kubiatowicz, and S. Shenker. Fixing the embarrassing slowness of opendht on planetlab. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, pages 25–30, Berkeley, CA, USA, 2005a. USENIX Association. @pages 92

S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. Technical Report Technical Report UCB//CSD-03-1299. The University of California, Berkeley, Univ. Paris-Sud, 2003. @pages 94

S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: a public dht service and its uses. In *SIGCOMM '05: Proceedings of the 2005* *conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, 2005b. ISBN 1-59593-009-4. @pages 82

J. Risson and T. Moors. Survey of research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50(17):3485–3521, 2006. @pages 13

A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany*, 2001. @pages 9, 22, 35, 41, 46, 76, 100

M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup – hypercubes, ontologies and efficient search on p2p networks. In *Workshop on Agents and P2P Computing, 2002.*, 2002. @pages 16, 78, 93

T. Schütt, F. Schintke, and A. Reinefeld. Chord#: Structured overlay network for non-uniform load distribution. Technical Report Technical Report ZR-05-40, Zuse Institut Berlin, 2005. @pages 18

A. Shaker and D. S. Reeves. Self-stabilizing structured ring topology p2p systems. In *th IEEE Int. Conference on Peer-to-Peer Computing, 2005*, pages 39–46. IEEE Computer Society, 2005. @pages 31, 61, 94

A. Shraer, G. Chockler, I. Keidar, R. Melamed, Y. Tock, and R. Vitenberg. Local on-line maintenance of scalable pub/sub infrastructure. In *DSN 2007: in the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, page 100, Edinburgh, UK, 2007. @pages 100

G. Skobeltsyn, T. Luu, I. Podnar Žarko, M. Rajman, and K. Aberer. Web Text Retrieval with a P2P Query-Driven Index. In *SIGIR'07: Proceedings of The 30th Annual International ACM SIGIR Conference*, 2007. @pages 14

I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, 2001. ISBN 1-58113-411-8. @pages 4, 6, 16, 17, 19, 22, 23, 26, 27, 35, 41, 46, 54, 56, 61, 76, 79, 93

D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association. @pages 5, 72

G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994. @pages 25

W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. *SIGCOMM Comput. Commun. Rev.*, 37 (4):49–60, 2007. ISSN 0146-4833. @pages 15, 94

Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *17th IASTED International Conference Parallel and Distributed Computing and Systems*, pages 320–327, 2005. @pages 99, 100, 109, 111

Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock. Dr. multicast: Rx for data center communication scalability. Technical Report Technical Report 1813-11587, Cornell University, 2008. @pages 100

S. Voulgaris, E. Rivière, A. marie Kermarrec, and M. V. Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks. In *In the fifth International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006. @pages 7, 98, 100

W. Wang, H. Chang, A. Zeitoun, and S. Jamin. Characterizing guarded hosts in peer-to-peer file sharing systems. In *In Proceedings of IEEE Global Communications Conference*, Global Internet and Next Generation Networks, 2004a. @pages 76, 90, 92, 95

X. Wang, Y. Zhang, X. Li, and D. Loguinov. On zone-balancing of peer-to-peer networks: analysis of random node join. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 211–222, New York, NY, USA, 2004b. ACM. ISBN 1-58113-873-3. @pages 48

D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. In *Nature, vol. 393, pp. 440-442*, 1998. @pages 2, 42

B. Wong, Y. Vigfússon, and E. G. Sirer. Hyperspaces for object clustering and approximate matching in peer-to-peer overlays. In *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 1–6, Berkeley, CA, USA, 2007. USENIX Association. @pages 100

Z. Xu, R. Min, and Y. Hu. Hieras: a dht based hierarchical p2p routing algorithm. *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 187–194, Oct. 2003. ISSN 0190-3918. @pages 16

H. Zhang, A. Goel, and R. Govindan. Using the small-world model to improve freenet performance. *SIGCOMM Comput. Commun. Rev.*, 32(1):79–79, 2002. ISSN 0146-4833. @pages 43

H. Zhang, A. Goel, and R. Govindan. Incrementally improving lookup latency in distributed hash table systems. In *In ACM Sigmetrics*, pages 114–125, 2003. @pages 42

B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, Jan. 2004. ISSN 0733-8716. @pages 9, 22, 41, 100

B. Y. Zhao, Y. Duan, L. Huang, A. D. Joseph, and J. D. Kubiatowicz. Brocade: Landmark routing on overlay networks. In *In Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 34–44, 2002. @pages 18

S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20, New York, NY, USA, 2001. ACM. ISBN 1-58113-370-7. @pages 100

# Curriculum Vitæ

## Personal Data

| | |
|---|---|
| Name: | **Šarūnas Girdzijauskas** |
| Date of birth: | December 01, 1978 |
| Place of birth: | Kaunas, Lithuania |
| Nationality: | Lithuanian |
| Languages: | Lithuanian (mother tongue), English (fluent), Russian (fluent), French (intermediate), Serbian (basics), Polish (basics) |
| Email: | sarunas.girdzijauskas@epfl.ch |
| Web: | http://www.girdzijauskas.lt/ |

## Education

**PhD** in Computer Science **2009**

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
Distributed Information Systems Laboratory, School of Computer and Communication Sciences.

**Doctoral School** in Computer Science **2003**

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
School of Computer and Communication Sciences.

**M.S.** in Computer Science **2002**

Kaunas University of Technology, Faculty of Informatics, Lithuania.
With honors, GPA 9.33/10 (top 3%)

**B.S.** in Computer Science **2000**

Kaunas University of Technology, Faculty of Informatics, Lithuania.
With honors, GPA 9.85/10 (top 4%).

## Work Experience

| | |
|---|---|
| Dec. 2003 – present | **Research Assistant** at EPFL, Lausanne (Switzerland). Distributed Information Systems Laboratory. Teaching assistantship (5 semesters), student projects supervision. |
| Jan. 2008 – Apr.2008 | **Intern** at IBM Haifa Research labs, Israel. Patent under submission. |
| 2001 – 2002 | **IT Engineer** at Hansabank, Lithuania. |
| 1999 – 2001 | **IT Engineer** at Lithuanian Savings Bank, Lithuania. |

## Publications

### Journal Papers and Book Chapters

1. Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *Structured Overlay For Heterogeneous Environments: Design and Evaluation of Oscar.* To be published in ACM Transactions on Autonomous and Adaptive Systems.

2. Sarunas Girdzijauskas, Wojciech Galuba, Vasilios Darlagiannis, Anwitaman Datta, Karl Aberer: *Fuzzynet: Ringless Routing in a Ring-like Structured Overlay* To be published in Peer-to-Peer Networking and Applications Journal, Springer.

3. Wojciech Galuba, Sarunas Girdzijauskas: *Peer to Peer Overlay Networks: Structure, Routing and Maintenance* Entry in Springer's upcoming Encyclopedia of Database Systems.

### Conference and Workshop Papers

1. Sarunas Girdzijauskas, Gregory Chockler, Roie Melamed, Yoav Tock: *Gravity: An Interest-Aware Publish/Subscribe System Based on Structured Overlays (short paper).* The 2nd International Conference on Distributed Event-Based Systems (DEBS), July 1-4, 2008, Rome, Italy.

2. Fabius Klemm, Sarunas Girdzijauskas, Jean-Yves Le Boudec, Karl Aberer: *On Routing in Distributed Hash Tables.* The 7th IEEE International Conference on Peer-to-Peer Computing (P2P), September 2-5 2007, Galway, Ireland.

3. Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *Oscar: A Data-Oriented Overlay For Heterogeneous Environments (short paper).* The 23rd International Conference on

Data Engineering (ICDE), April 16-20, 2007, Istanbul, Turkey.

4. Mirko Steinle, Karl Aberer, Sarunas Girdzijauskas, Christian Lovis: *Mapping Moving Landscapes by Mining Mountains of Logs: Novel Techniques for Dependency Model Generation.* The 32nd International Conference on Very Large Data Bases (VLDB), September 12-15, 2006, Seoul, Korea.

5. Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *Oscar: Small-world Overlay for Realistic Key Distributions.* The Fourth International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), September 11, 2006, Seoul, Korea.

6. Karl Aberer, Luc Onana Alima, Ali Ghodsi, Sarunas Girdzijauskas, Manfred Hauswirth, Seif Haridi: *The Essence of P2P: A Reference Architecture for Overlay Networks.* The 5th IEEE International Conference on Peer-to-Peer Computing (P2P), August 31-September 2 2005, Konstanz, Germany.

7. Sarunas Girdzijauskas, Anwitaman Datta, Karl Aberer: *On Small World Graphs in Non-uniformly Distributed Key Spaces.* The 1st IEEE International Workshop on Networking Meets Databases (NetDB), April 8-9 2005 Tokyo, Japan.

8. Anwitaman Datta, Sarunas Girdzijauskas, Karl Aberer *On de Bruijn routing in distributed hash tables: There and back again.* The 4th IEEE International Conference on Peer-to-Peer Computing (P2P), August 25-27 2004, Zurich, Switzerland.

---

Lausanne, December 22, 2008

✠