A Hierarchical Framework for Peer-to-Peer Systems: Design and Optimizations

Un Marc de Treball Jeràrquic per a Sistemes d'Igual-a-Igual:
Disseny i Optimitzacions

(subtitulat en català)

Marc Sánchez Artigas

TESI DOCTORAL UPF / ANY 2008

DIRECTOR DE LA TESI: Dr. Pedro García López
Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili

DIRECTOR DE LA TESI: Dr. Miquel Oliver Riera
Departament de Tecnologies de la Informació i les Comunicacions
Universitat Pompeu Fabra

UNIVERSITAT POMPEU FABRA

To all those that have contributed to
this work with either their support,
their knowledge, their friendship and love.

Acknowledgments

This dissertation has been written during my time at the Laboratory of Architecture and Telematic Services (ATS) at Universitat Rovira i Virgili.

First, I would like to express my deepest appreciation to my advisor Dr. Pedro García for his unconditional support over the years. He gave freedom to me, to fully develop my ideas, while providing insightful research and advice, which was critical to the successful completion of this thesis. I would never forget our stimulating and enjoyable discussions about distributed systems. His support at certain points have been even more than incalculable.

My thanks also go to Dr. Antonio Skarmeta, who have supported and encouraged me when needed.

I would like to acknowledge the members of the Laboratory ATS at Universitat Rovira i Virgili for their support and friendship, that have made life what it is, sometimes surprisingly, but all the time the most interesting experience I have ever been involved in. Without their help, this thesis would not have materialized. Part of this dissertation is theirs. I also want to thank my colleague and friend Jordi Pujol for his suggestions, discussions and support, as well as my friend Toni Tur. Further, I would like to acknowledge Lluis Pàmies for being a valued friend during the final stage of my PhD study. Without their parties and dinners, I would have forgotten that working on a thesis is more than publishing a collection of papers.

My most important appreciation goes to my family who have been the source of my strength and optimism over the years. I would like to specially thank Caterina, my parents and my brother Jordi for their love and support. Their patience and perseverance encouraged me to look ahead in the moments where giving up was really more than a temptation.

Finally, I would like to thank Dr. Miquel Oliver and Dr. Josep Blat for giving me the chance to defend this dissertation at Universitat Pompeu Fabra.

I appreciate you all very deeply

Marc Sánchez Artigas
Reus, October 2008

ABSTRACT

A peer-to-peer (P2P) overlay network is a logical network, built on top of an underlying physical network, which facilitate the location of distributed resources without centralized control. These systems have emerged on the edges of the Internet thanks to the generalized growth of broadband Internet connections.

There exist two main families of P2P systems: unstructured and structured. In the former, the distribution of resources across the network and the overlay structure are independent. They are more easily set up, but resource location requires exhaustive search over the network. In the latter, the distribution of resources and the network structure are strongly coupled and hence, searches for resources are more efficient. However, their construction and maintenance are more complex than in unstructured overlay networks.

On the contrary, structured overlays, also called Distributed Hash Tables (DHTs), overcome the scaling problems of unstructured systems. Essentially, DHTs provide the same functionality of a hash table — the standard `Put(key, value)` and `Get(key)` interface, but associating key-value pairs with users rather than hash buckets.

Due to its salient scalability, DHT abstraction has received a lot of attention in the last years. However, many of the existing proposals have been conceived under the assumption that logical proximity in the overlay does not need to match physical proximity in the Internet. This mismatch has become a large barrier for the deployment and performance optimization of P2P applications. In addition, many of the existing designs have assumed that communication is uniform, while, in practice, users communicate more frequently with other users in the same administrative domain.

The traditional approach to tackle these flaws revolves around the organization of users into a hierarchy of domains. By partitioning the network into smaller domains, complexity reduces too. As typical examples, DNS or group multicast achieve scalability via a hierarchical design.

The basic problem is that most of the existing DHTs have been devised as flat structures and hence, they cannot enjoy the scalability of a hierarchical design. In this dissertation, we attempt to address this issue from three different perspectives:

Seduced by the scalability of hierarchical designs, we start by presenting a hierarchical framework for DHTs. The main aim of our framework is to provide a generic methodology to transform a flat DHT into a hierarchical one made up of telescoping clusters — clusters of ... of clusters of peers. Our idea is to exploit the recursive structure that many DHTs have to embed a hierarchy of domains. This strategy has an important benefit: our hierarchical construction of a DHT inherits the homogeneity of load and functionality of the original design but with the advantages of having a hierarchical structure. Further, we provide the hierarchical version of Chord and give some hints to transform another six DHTs. With our hierarchical construction of Chord, we investigate the potential reduction in query latency offered by our framework.

Then, we proceed to study an interesting question: what makes our hierarchical constructions better than existing hierarchical DHTs? To address this issue, we introduce an analytic cost model. Our idea is to provide the means for identifying the optimal hierarchical design in terms of communication cost. In an effort to show the lack of a universally optimal design, we analyze the two main families of hierarchical designs: the superpeer design and the homogeneous design. Our hierarchical constructions belong to the last family.

In general, our hierarchical designs have many interesting applications. For example, the introduction of multiple separate domains in a flat structure can be used to improve its performance. One example is latency optimization. By adapting the clusters to the physical network, the search latency could be diminish within each cluster and obtain important savings. The problem is how to divide peers into low-latency clusters in a decentralized and accurate manner.

To answer this question, we finally introduce a clustering algorithm that attempts to organize peers into clusters, so that peers in a cluster are closer — in terms of round-trip-time — to each other than peers not in their cluster. To assess the quality of our clusterings, we propose a novel metric called false clustering rate. This metric measures the proportion of falsely clustered peers in a clustering. Falsely clustered peers are distant nodes that have been clustered together. Using our metric, we provide enough evidence that our algorithm can achieve important improvements.

Keywords: Distributed Systems, Peer-to-Peer Overlay Networks, Distributed Hash Tables, Hierarchical Designs, Proximity Clustering.

Resum

En el darrers anys, Internet ha experimentat una forta expansió gràcies a l'aparició d'un conjunt d'aplicacions d'àmbit global que han esdevingut pràcticament imprescindibles per la majoria d'usuaris. Aquest grup d'aplicacions, que inclouen des de serveis de missatgeria instantània fins aplicacions d'intercanvi de fitxers com BitTorrent o eMule, s'acostumen a construir sobre substrats peer-to-peer (P2P),també anomenats d'*igual-a-igual*. Aquestes substrats es contitueixen en forma de xarxes overlay o de *recobriment*. Les xarxes overlay o de *recobriment* es poden definir com a xarxes punt a punt que interconnecten usuaris de manera lògica i desacoblada de la topologia física, i que proporcionen un servei descentralitzat de cerca de recursos.

Existeixen dues grans famílies de xarxes P2P descentralitzades: les *xarxes P2P desestructurades* i les *xarxes P2P estructurades*. En les xarxes desestructurades, la distribució dels recursos és independent de la topologia de la xarxa overlay, mentre que en les homòlogues estructurades, la distribució dels recursos i la topologia es troben fortament acoblades. La conseqüencia d'aquest acoblament es tradueix en una localització de recursos més eficient, tot i que la seva construcció i manteniment siguin més costosos que en els sistemes desestructurats.

Des del punt de vista funcional, les xarxes estructurades reben el nom de *Taules de Hash Distribuïdes* (DHTs). Bàsicament, les DHTs proporcionen la mateixa funcionalitat de les taules de hash tradicional — la interfície estàndard `Put(clau, valor)` i `Get(clau)`, però associant els parells clau-valor amb usuaris de la DHT.

Degut a les seva excel·lent escalabilitat, les DHT han generat una gran expectació en el darrers anys. Tanmateix, la seva adopció com a eina generalitzada de comunicació és encara lenta degut a una sèrie d'incovenients. El primer inconvenient és que l'estructura lògica de les DHTs no es correspon amb la topologia física d'Internet. En altres paraules, un usuari pot tenir com a veïns a d'altres participants que en realitat es trobin molt allunyats — en termes de latència — d'ell. Per aplicacions en què el retard extrem-a-extrem ha de ser necessàriament baix, aquesta manca de correspondència és un gran obstacle. D'altra banda, molts dels dissenys actuals assumeixen que la comunicació és uniforme, mentre que en la pràctica els usuaris es comuniquen de manera més freqüent amb els usuaris que pertanyen el mateix domini administratiu, comparteixen els mateixos interessos etc.

Per resoldre aquest tipus de deficiències, tradicionalment s'ha recorregut a l'organizació dels usuaris en dominis jeràrquics. Exemples arquetípics d'aquesta estratègia inclouen el sistema DNS i els sistemes de distribució i gestió de contingut multimèdia d'alta qualitat.

El problema bàsic és que la majoria de DHTs s'han dissenyat seguint estructures planes i per tant, no poden gaudir dels avantatges de les jerarquies. En aquesta dissertació, hem procurat solucionar aquesta mancança de la forma següent:

Seduïts per l'escabilitat del dissenys jeràrquics, en la primera part de la nostra discussió, desenvolupem un framework o *marc de treball* jeràrquic per a DHTs. L'objectiu principal d'aquest framework es proporcionar una metodologia genèrica per a transformar una DHT qualsevol en una DHT jeràrquica constituïda per *grups* o clusters telescòpics — clusters de clusters de ... de clusters d'usuaris. L'idea bàsica consisteix en explotar, si és possible, la seva estructura recursiva. En cas afirmatiu, la construcció jeràrquica hereta la homogeneïtat en càrrega i funcionalitat del disseny original, però amb els avantatges addicionals derivats d'una estructura jeràrquica. Per il·lustrar l'utilitat del nostre framework, proporcionen la versió jeràrquica de Chord i un conjunt d'indicacions per a poder transformar sis DHTs de manera senzilla. Concloem aquesta part amb l'estudi de la millora en el rendiment experimentable pels nostres dissenys.

En la segona part d'aquesta dissertació, responem a una qüestió que hom hauria de tenir molt present a fi de poder valorar objectivament l'utilitat del nostre framework: *En quins aspectes les nostres construccions jeràrquiques són superiors a les existents?* Per a donar una resposta satisfactòria a aquesta pregunta, introduïm un model genèric basat en costos. La nostra contribució és proporcionar un mecanisme imparcial que permeti identificar quin model jeràrquic és òptim en termes de cost de comunicació. Per assolir aquest objectiu, analitzem les dues famílies principals de dissenys jeràrquics: el model basat en *superusuaris* i el disseny *homogeni*, i arribem a la conclusió que no hi ha un disseny clarament superior a l'altre. Cal esmentar que les nostres construccions pertanyen a la família homogènia.

En general, els nostres dissenys jeràrquics ofereixen un ampli ventall d'aplicacions relacionades amb l'explotació de múltiples dominis. Una exemple representatiu pot ser l'optimització del rendiment. Si la comunicació és freqüent entre usuaris d'un mateix domini, l'adaptació dels dominis a la xarxa física permetrà reduir el temps de cerca mitjà del sistema. El problema bàsic és com organitzar els usuaris en clusters de baixa latència, de forma descentralitzada i escalable.

Per a satisfer aquesta qüestió, l'última part d'aquesta tesi introdueix un nou algoritme de clustering o d'*agrupament*. La funció d'aquest algoritme és organitzar els usuaris en múltiples clusters de manera que els usuaris dins d'un cluster estiguin mútuament més propers — en termes de latència — que usuaris pertanyents a clusters diferents. Per mesurar la qualitat de la nostra solució, proposem una nova mètrica anomenada *false clustering rate*. Aquesta mètrica mesura la proporció d'usuaris *falsament agrupats* dins del sistema. Per usuaris falsament agrupats ens referim a usuaris llunyans que han estat erròniament agrupats dins d'un mateix cluster. Finalment, demostrem empíricament com el nostre algoritme permet assolir millores significatives respecte les tècniques existents.

Paraules Clau: Sistemes Distribuïts, Sistemes d'Igual-a-Igual, Xarxes overlay d'Igual-a-Igual, Taules de Hash Distribuïdes, Disseny Jeràrquic, Clustering per Proximitat.

Contents

1 INTRODUCTION **1**
1.1 General Introduction 2
1.1.1 Distributed Hash Tables 2
1.2 Topics and Motivation of this Thesis 4
1.3 Contributions of this Thesis 6
1.4 Outline of this Dissertation 7
1.5 Selected Publications 8

2 BACKGROUND **11**
2.1 Peer-To-Peer 12
2.1.1 Unstructured P2P Overlay Networks 12
2.1.2 Flat Structured P2P Overlay Networks 12
2.1.3 Hierarchical Structured P2P Overlay Networks 17

3 CYCLONE: A FRAMEWORK FOR DESIGNING HIERARCHICAL DHTS **25**
3.1 Introduction 26
3.1.1 Goals 26
3.2 Related Work 30
3.3 Background 31
3.3.1 Graph-theoretic and Group-theoretic Notions 31
3.3.2 Chord 33
3.4 Cyclone Framework 33
3.4.1 Overview 33
3.4.2 Determining Recursive Structures: Cayley DHTs 35
3.4.3 Mapping a Hierarchy 42
3.5 Adaptability to Other DHTs 52
3.6 Performance Evaluation 54
3.6.1 Basic Properties: Degree and Average Path Length 54
3.6.2 Query Latency 55
3.6.3 Load Balancing 58
3.7 Conclusions 58

4 A COMPARATIVE STUDY OF HIERARCHICAL DHT SYSTEMS **63**
4.1 Introduction 64
4.2 Related Work 65

4.3 Hierarchical Architectures 66
4.3.1 Homogeneous Design 67
4.3.2 Superpeer Design 68
4.4 Locality Analysis 70
4.4.1 Homogeneous Design 71
4.4.2 Superpeer Design 71
4.4.3 Discussion of Results 72
4.5 Cost-Based Analysis 73
4.5.1 Homogeneous Design 73
4.5.2 Superpeer Design 75
4.5.3 Discussion of Results 80
4.6 Conclusions and Future Research 82

5 False Clustering on Peer-to-Peer Networks 83
5.1 Introduction 84
5.2 Related Work 86
5.3 False Clustering on Proximity Techniques 87
5.3.1 Clustering Model 88
5.3.2 False Clustering Rate 88
5.3.3 Case Study: Proximity Neighbor Selection 89
5.4 TR-Clustering: Principles 95
5.4.1 Internet Routers and Betweenness Centrality 96
5.4.2 Sampling Bias in Traceroute Exploration 97
5.5 TR-Clustering: Algorithms 98
5.5.1 Landmark Selection 98
5.5.2 Clustering Nodes 101
5.5.3 Putting It All Together: a Clustering Example 102
5.5.4 Traceroute: Implementation Issues 103
5.6 Experimental Results 104
5.6.1 Comparison of TR-Clustering against Global Clustering 106
5.6.2 Impact of the Density Threshold 108
5.6.3 Performance Evaluation on PlanetLab 110
5.6.4 Performance Evaluation on DIMES 114
5.7 Summary and Future Directions 117

6 Conclusions and perspectives 119
6.1 Conclusion 120
6.1.1 Summary of Contributions 121
6.2 Directions for future research 122

References 123

List of Figures

1.1 *Routing example on a Distributed Hash Table.* 3

2.1 *Chord's graph embedding: the neighborhood (finger table) for node 8 is shown in the figure.* 15

2.2 *An example of a hierarchical organization in Garces-Erice et al.* 19

2.3 *An example of Coral routing. Higher-tier, smaller-diameter domains are naturally sparser. For a query on identifier id, the querying node a first routes on its highest domain. The routing fails on that tier if the node closest to id on the identifier space, node t_2, does not store the object. If this occurs, Coral continues routing on a lower tier domain, having already traversed the identifier space up to t_2.* 23

3.1 *Coral hierarchical routing. Nodes use the same ids in each of their clusters; higher-level, smaller-diameter clusters are naturally sparser. For a lookup on key k, the querying node a first applies Path-Constrained routing on its highest cluster. The routing fails on that level if the node closest to key k on the identifier space, node t_2, does not store the key. If this occurs, Coral continues its lookup on a lower level cluster, having already traversed the identifier space up to t_2.* 27

3.2 *Recursive decomposition of Q_4. Edges belonging to the 3-dimensional hypercubes are shown in bold style. Interconnection edges are drawn in thin style.* 34

3.3 *Recursive definition of $CH_3 = Cay(\mathbb{Z}_8, \{\pm4, \pm2, \pm1\})$. All vertices are represented in base 2. Intra-cluster edges of the two copies of CH_2 are shown in bold style while inter-cluster edges are drawn in thin style. All nodes belonging to the same copy share the last bit of their $nodeIds$.* 38

3.4 *An example of a Whirl construction.* 44

3.5 *Routing example in Whirl. Node 0 routes a query for key $k = 7$. Within each cluster, clockwise* GREEDY *routing is performed until the node responsible for the key is found.* 48

3.6 *Comparison between the average path length of Chord and Whirl for $\alpha = 16$ and $m = 10$. The value of γ is varied from 0.0 to 1.0* 51

3.7 *Out-degree comparison between Chord and Whirl.* 55

3.8 *Routing comparison between Chord and Whirl.* 56

3.9 *Improvements in query time achieved by Whirl as a function of communication locality, γ.* 57

3.10 *Load balancing comparison between Chord and Whirl.* 58

4.1 *Two-tier Whirl architecture.* 67

4.2 *Two-tier superpeer Chord-like architecture.* 69

4.3 *Comparison between homogeneous (HD) and superpeer (SD) designs' average path length for $D = 32$ and $d = 10$. The value of γ is varied from 0 to 1 (a). γ is fixed at 0.5 and d is varied from 1 to 32 (b).* 72
4.4 *Chord's spanning binomial tree (d = 4).* 77
4.5 *Traffic comparison between homogeneous and superpeer designs for D = 32 and d = 10. (a) illustrates PMF of $R_{p,IN}$. The relative total cost, namely C_{SD}/C_{HD}, is shown in (b) when γ is varied from 0 to 1. Similarly, we plot the relative cost between the highest loaded peers (HLPs) of both designs. (c) Varies locality γ. (d) Varies node lifetime (γ is set to 0.5).* 81

5.1 *An illustration of the backbone of the star-like graph data set.* 92
5.2 *CDF of the absolute relative error for GNP over $100,000$ random node-pairs.* 94
5.3 2*-dimensional plot of the resultant clustering for GNP. Each cluster uses a distinct mark.* 94
5.4 *CDFs of the average* fcr *for Landmark Ordering and GNP.* 94
5.5 agr *for Ideal C-PNS, Landmark Ordering and GNP.* 94
5.6 *CDF of lookup latency for Landmark Ordering, GNP and Ideal C-PNS over Chord. Plain Chord (labeled Chord in the figure) is used as baseline for comparison.* 95
5.7 *A step-by-step clustering example* 103
5.8 *CDF of the RTT over all pairs of PlanetLab nodes.* 106
5.9 *Comparison between TR-Clustering and K-Means on GT-ITM: Evolution of the average* fcr *with an increasing number of clusters* 108
5.10 *Comparison between TR-Clustering and K-Means on GT-ITM: Evolution of the average gain ratio with an increasing number of clusters* 108
5.11 *TR-Clustering vs. K-Means on GT-ITM: Evolution of Davies-Bouldin's (*DB*) index with an increasing number of clusters* 108
5.12 *Resultant clustering for TS-*45 *topology. For instance, nodes S2.2/1.0 and S2.2/0.0 are in the same cluster.* 109
5.13 *Landmark selection step-by-step in TS-*45 *topology. Black arrows indicate the landmarks chosen at next iteration.* 109
5.14 *Effect of the density threshold on* fcr*. Each group of two bars illustrates the* fcr *experienced on six models for thresholds $\rho = \lfloor 0.01N \rfloor$ and $\rho = \lfloor 0.1N \rfloor$, respectively. As suggested by this figure, TR-Clustering appears no particularly sensitive to the current value of ρ.* 110
5.15 Sampling bias (power-law behavior) in RF-1221 dataset after first iteration. 110
5.16 *Average* fcr *of TR-Clustering on PlanetLab with alias and without alias resolution.* 111
5.17 *CDFs of* RE *for Vivaldi and GNP on PlanetLab.* 112
5.18 *Evolution of the average* fcr *as a function of k for TR-Clustering, GNP and Vivaldi on PlanetLab no-alias dataset.* 113
5.19 2*-dimensional plot of the resultant clustering for Vivaldi. $k = 2$.* 113
5.20 *Evolution of the Davies-Bouldin's Index as a function of k for TR-Clustering, GNP and Vivaldi on PlanetLab no-alias dataset* 114
5.21 *CDF of* RE *for Vivaldi over all pairs of DIMES hosts.* 115
5.22 *CDF of cluster size when $k = 450$.* 115
5.23 *Average* fcr *as a function of k for TR-Clustering and Vivaldi.* 116
5.24 *Davies-Bouldin's index as a function of k for TR-Clustering and Vivaldi.* 117

List of Tables

5.1 Average cluster′s gain ratio for GNP and Ideal C-PNS. 95

List of Abbreviations

To improve the readability of the text, the meaning of an abbreviation or an acronym is often given only once at its first appearance in the text of a chapter. The following list includes the most often used abbreviations for the readers reference.

AS: Autonomous System (plural: ASes)

BGP: Border Gateway Protocol

CCDF: Complementary Cumulative Distribution Function

CDF: Cumulative Distribution Function

CDN: Content Distribution Network

DHT: Distributed Hash Table (plural: DHTs)

DNS: Domain Name System

GIS: Geographic Information Services

ID: Identifier (plural: IDs)

IP: Internet Protocol

ISP: Internet Service Provider (plural: ISPs)

LAN: Local Area Network (plural: LANs)

NC: Network Coordinate System (plural: NCs)

P2P: Peer-to-Peer

POP: Point-of-Presence (plural: POPs)

RTT: Round Trip Time (plural: RTTs)

TCP: Transport Control Protocol

XOR: eXclusive OR (bit-wise operation)

1 Introduction

1.1 General Introduction

In May 1999, an 18-year old college student, Shawn Fanning, devised Napster. Shawn developed Napster to help music lovers to find an exchange MP3 files on the Internet. Unlike search engines of the day, Napster indexed the files the users wanted to share on a central server that the other users could access. As a result, Napster users could directly download MP3 files from other users, bypassing the use of a central server.

Despite being shut down by USA government [1], Napster made the term peer-to-peer (P2P) popular. Inspired by Napster, a new generation of P2P systems soon appeared that led a revolution on file sharing and other type of content delivery applications. But, *what exactly a P2P system is?* In order to clarify it, we propose our own definition, as follows:

P2P systems are a class of distributed applications that exploit the resources — storage space, CPU cycles, content, bandwidth — provided by the users at the edges of the Internet to their benefit.

Compared with the traditional client-server architecture, a pure P2P system can be described as a decentralized network system in which all participant computers (also known as peers) have symmetric duties and responsibilities. In simplistic terms, all participating computers act as both clients and servers to one another, leading to a large pool of information sources and computing power. Key features of P2P systems are decentralization, self-organization, dynamism, and fault-tolerance, all of which make P2P paradigm very attractive for information storage and retrieval.

In terms of network structure, P2P systems are roughly classified into two categories: unstructured and structured. The first wave of P2P systems implemented unstructured P2P overlays, in which data placement was random and no global structure was maintained (e.g., Gnutella [2]). To look for a file, unstructured systems used message flooding to propagate queries. Although such systems were fault-tolerant and resilient to users joining and leaving the network (phenomenon commonly known as *churn*), their search mechanism did not scale [3]. Structured overlays [4] [5] [6], which implement a *Distributed Hash Table* (DHT) data structure, were proposed to increase the scalability of unstructured systems. This is the main reason why the focus of this thesis is on DHTs, and not on unstructured systems in which searches are less efficient.

1.1.1 Distributed Hash Tables

DHTs provide the same functionality of a traditional hash table — the standard `Put(key, value)` and `Get(key)` interface— but associating key-value mappings with participating nodes rather than hash buckets. The location of an object is thus determined by the hash-value of its name. For instance, cryptographic secure hash-functions such as MD5 [7] or SHA-1 [8] map arbitrary strings to 128-bit or 160-bit hash-values, respectively. These functions can be used to map arbitrary object names into n-bit hash-values, where n depends upon the hash function being used. This defines an identifier space $\mathcal{I} = [0, 2^n - 1]$, where objects are assigned to.

In a DHT, each participating computer is also assigned an identifier in space $\mathcal{I}$. At any instant, the current set of identifiers determines the set of partitions (hash buckets) that the hash-table has been divided into. Each host is the manager of a distinct partition (hash bucket), being responsible for all objects whose names hash into that partition.

With this abstraction, in order to insert or retrieve an object, one first computes the hash-value of its name, and then locates the peer whose partition contains that hash-value. In order to contact

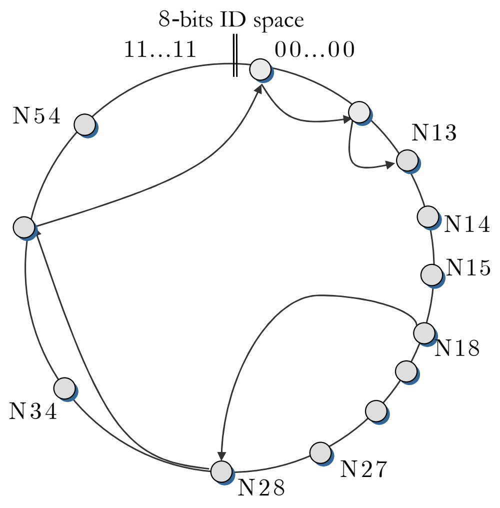


Figure 1.1: *Routing example on a Distributed Hash Table.*

this peer, knowledge of its physical address (IP address and port) is needed. The management of this knowledge arises a key question in DHTs:

Does each peer need to know the complete mapping between the current set of partitions and the physical addresses of their corresponding managers?

The answer might be affirmative if there would be a relatively small and static set of peers in the network. However, decentralization forces such a mapping neither to be maintained as global knowledge nor to be available at some central site. The adopted common solution is to allow each participating peer to establish links only to a few neighbors. Taken together, these links constitute what is known as an overlay network. As a consequence, a request to insert or retrieve an object is injected into the overlay which routes the request to its appropriate manager. Each peer along the route selects one of its out-going links to take the request increasingly closer to the manager. This establishes a logical path between the source and the manager, which is made up of all the intermediate peers through which the request was routed.

While there are significant implementation differences between DHTs, all achieve efficiency by organizing the participant nodes into a well-defined structure, so that queries can be resolved within $\mathcal{O}(\log N)$ number of hops, where N is the number of peers in the network.

The main features of DHTs can be summarized as:

- *Scalability:* there is not a universal definition of scalability. However, in peer-to-peer context, scalability means that a DHT should work efficiently for overlay networks of arbitrary size. This implies handling node arrival and departures in a scalable fashion. A node arrival and departure requires to redistribute the affected objects over the neighbors and reestablish the affected links.

- *Small degree:* this property is consequence of the above property. It means that there should be a reasonable number of neighbors per node; otherwise, link maintenance overhead might congest the entire network.

- *Small diameter:* the manager of an object should be reachable through a short path. At most, the diameter should be of $\mathcal{O}(\log N)$ routing hops. The higher the number of hops, the larger the latency for `Get` and `Put` operations.

- *Load Balancing:* load should be well-balanced across all peers in the network. In a call to `Put` and `Get`, the communication load should be equitably distributed among all participating peers. This includes preserving the randomness guarantee of hashing and the distributional properties of in-degree, which are very relevant for load balancing in message forwarding.

- *Fault Tolerance:* the structure of the overlay should be preserved by a maintenance protocol. Due to unpredictable peer and link failures, a node could suddenly become unreachable. To provide reliable operation, a DHT should be resilient against such failures. Furthermore, it should exhibit a maintenance complexity as much low as possible.

Examples of DHTs are Chord [4], CAN [9], Pastry [5], Tapestry [10], Symphony [11], P-Grid [12] and Kademlia [6].

1.2 Topics and Motivation of this Thesis

The academic community has implemented a number of DHTs as efficient, scalable, and robust P2P infrastructures. However, most of these proposed DHTs make two assumptions which do not correspond to reality. These assumptions are as follows:

- *Uniform communication:* DHTs assume that communication between peers is uniform, that is, all peers have an equal probability of processing a request. However, at any given time, a peer is usually interested in only a few topics and tends to issue queries and share collections of objects about those topics he/she is interested in. This makes communication to be non-uniform and to bother users with queries he/she is not interested in at all. For example, two users from the same organization cannot prevent to route their queries through users not in their organization. Further, applications cannot ensure that an object is stored in a given domain. This lack of control is consequence of the de-clustering nature of the hash functions used to assign peers and objects into the identifier space. In summary, lack of control over object placement and routing paths raises concerns over autonomy, administrative control and accountability of participating organizations.

- *Independence of the underlying physical network:* DHTs have traditionally considered the physical IP-level network as a transparent layer, transmitting data from one point of the overlay to another. This assumption could make sense in small LAN where all users are close to each other and network delays are low. In the Internet, however, this transparency has negative repercussions on scalability. Because in a DHT `Get` and `Put` operations take $\Theta(\log N)$ hops on average, the routing latency between two nodes on the overlay can be very different from the unicast latency between those two peers in the Internet. Put another way, a single logical hop on the overlay could incur several hops in the underlying IP-level network. This clearly results in poor routing performance, and can adversely affect the performance of the applications and services that could use DHTs as communication infrastructures.

To drop these assumptions, we have adopted the hierarchy as a method of organization. The reason is that hierarchies are ideal to accommodate growth and to reflect administrative domains. Our idea is to group the peers within a domain into the same cluster, and represent the hierarchical relationships among domains as a hierarchical DHT. As signaled by [13], the use of hierarchical organization provides two important routing properties:

- *Path locality:* the route between two nodes never leaves the lowest-tier domain that contains both nodes.
- *Path convergence:* the paths from two distinct peers in a domain D but heading to a common destination outside this domain converge at the same node in D

Path locality provides fault isolation, since interactions between two nodes in a domain cannot be interfered with by, or affected by the failure of, nodes outside that domain. On other hand, path convergence enables effective caching, as the answers to the queries for a given hash-value (`key`) can be cached at the peer to which all routing paths converge before leaving each domain.

This thesis improves on the current state-of-the-art by proposing a framework for constructing hierarchical DHTs. Traditionally, hierarchical design has been characterized by the differentiation of roles and responsibilities between the elements of a system. This view has given rise to what is known as superpeer systems.

Superpeer systems classify peers into two or more classes based on their individual capacities, such as CPU, bandwidth and storage space. The idea behind superpeer systems is to assign more responsibilities to high-capacity peers in order to improve the overall performance of flat systems. Typically, high-capacity peers, referred to as superpeers, provide routing services to other peers. Each superpeer becomes responsible for a subset of regular peers, which means that it maintains the necessary knowledge to resolve `Get` an `Put` requests targeted at, and coming from, the regular peers in its group. The main benefit of this structure is that, if properly maintained, provides `Get` an `Put` operations incurring a small number of hops.

However, these systems have a major shortcoming. They change load balancing for scalability, and fault tolerance for single points of failure, two changes that are questionable. For example, in the case of fault tolerance, the failure of a superpeer disconnects all its regular peers from the rest of the structure.

In stark contrast, the design of our framework assumes that all peers are homogeneous. Our idea is to preserve the load balancing and fault tolerance properties of flat DHTs but incorporating the advantages of hierarchical design, such as path locality and path convergence. Conceptually, our designs can be visualized as hierarchy in which peers share the same "horizontal position" of power and authority, each playing a theoretically equal role. Architecturally, this means that each peer maintains connections at all tiers of the hierarchy, instead of one connection to its superpeer or parent node. That is, a peer can be connected to other peers without needing to go through or get permission from some other peer in the hierarchy. This property has some important benefits: it balances routing load and increases resilience to failures, as no peer is a subordinate of a single other peer.

In summary, our contribution is the development of the first generic framework to construct hierarchical structured overlays in which any peer occupies a more prominent position than the rest. Our constructions have many interesting applications that go beyond latency optimization.

For example, we have applied our design to provide a secure routing primitive for DHTs [14] and a Geographic Information Service (GIS) in [15]. In general, our design can be used to implement any distributed application in which separate domains can improve its performance.

On the other hand, it is obvious that path locality could reduce search latency if clusters were optimized based on network latency. However, this task is not trivial in a decentralized environment. Ideally, every peer should be able to identify its cluster with little intervention of the other peers in the system. How to organize participating peers into low-latency clusters in a completely distributed and accurate manner is the challenge to address. By decentralization and accuracy we mean the following:

- *Decentralization:* each peer should identify its cluster without relying on global information and based on a relatively small set of network latency measurements.
- *Accuracy:* clusters should be generated with *minimum* false clustering rate. Falsely clustered peers are distant computers in the underlying physical network that have been erroneously clustered together.

Until now, locality algorithms have mainly attempted to adapt the overlay connections to the underlying network. But they have not considered that for many applications such as application-level multicast, it is more interesting to detect clusters of nearby peers than adapting their logical links.

To fill this gap, the second major contribution of this thesis is the introduction of a clustering algorithm to create clusters that minimize false clustering. In particular, we view clustering as the process of having a set of peers independently self-organize into disjoint clusters, so that peers within each cluster are closer to one another than peers not in their cluster. Clearly, this clustering model combined with path locality can help to improve the performance by avoiding logical links over high latency IP hops.

1.3 Contributions of this Thesis

In this thesis, we describe a framework for constructing and optimizing hierarchical DHTs. The design of our framework is modular. It consists of three main modules, which are the following:

- *The construction module* is responsible for constructing hierarchical structured overlays from their flat counterparts. The hierarchical constructions must fulfill several requirements: load balancing, fault isolation, small number of links per peer, routes shorter than in flat overlays when communication is between arbitrary peers in the same domain, low-latency routes, and resilience to failures. To implement this module, we make the following contributions:
  - **Our first contribution** is a generic framework which preserves all the properties of the flat overlays such as degree, load balancing and fault-tolerance, but incorporates all the advantages of hierarchical design. Our hierarchical construction strategy is based on the recursive structure of DHTs that we elucidate by means of their Cayley graph [16] representation. This makes our framework applicable to a large subset of the existing DHT designs.

  - **Our second contribution** is a novel hierarchical construction for Chord, which inherits its logarithmic complexity in construction, maintenance and degree while enjoying all the benefits of hierarchical design such as path locality. Further, we give some indicative hints of how to transform another six DHTs into hierarchical overlays. We verify, theoretically and through simulations, that our construction of Chord is equivalent to Chord, and in addition, it can obtain significant savings in search latency.

- *The comparative module* is responsible for identifying the features that make our hierarchical constructions better than others, and defining *What does better mean?* To shed light on these questions, we make the following contributions:

  - **Our third contribution** is an analytic cost model to calibrate the potential performance of hierarchical designs. Our strategy is not only to look at the graph-theoretic properties of each design, but also to determine the overall communication costs as a function of several parameters. Among them, locality in communication is very helpful to measure the improvement on search latency brought by path locality.
  - **Our fourth contribution** is the comparison between the two main families of hierarchical designs: the homogeneous design, in which all peers assume equal roles, against the superpeer design, in which a relatively small set of peers behave as proxies for the rest of peers in the overlay. For the comparison, we use our hierarchical construction of Chord as a representative of the homogeneous design. Our comparison shows that there exists no a universally better design: homogeneous designs may be worse than superpeer designs and vice versa, depending on the communication locality, lifetime and query traffic.

- *The clustering module* is responsible for dividing peers into low-latency clusters so that routing performance can be improved by exploiting path locality. We make the following two contributions:

  - **Our fifth contribution** is the definition of the false clustering problem. False clustering occurs when two distant nodes are clustered together. We study the interplay between false clustering and the performance of hierarchical overlays. For this purpose, we first propose a new metric to measure the quality of clusterings. Using this metric, we then show, through a simple example, the limitations of the existing clustering tools.
  - **Our sixth contribution** is an innovative clustering algorithm that is deliberately aimed at minimizing false clustering. Our algorithm reduces the rate of falsely clustered peers by recursively forming new clusters with peers much closer to each other than were in the previous clusters. Our simulation results show that our algorithm is superior to the existing tools in the literature, with a false clustering rate inferior to 5%.

1.4 Outline of this Dissertation

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

Chapter 2: Background. In general terms, this Chapter presents a conceptual model, notations and definitions for both flat and hierarchical DHTs, followed by a discussion of related work. We provide an extensive description of Chord [4], the DHT we use to demonstrate the utility of our framework.

Chapter 3: Cyclone: a Framework for Designing Hierarchical DHTs. This Chapter is central to this dissertation. In this Chapter, we describe our framework to construct hierarchical DHTs and our hierarchical construction of Chord. To conclude this Chapter, we include some simulations results to verify the effectiveness of our framework.

Chapter 4: A Comparative Study of Hierarchical DHT Systems. In this Chapter, we describe the analytic cost model that implements the comparative module. In addition, we compare the two main hierarchical designs: the homogeneous design and the superpeer design. Our analysis reveals that the costs incurred by the superpeer design are not necessarily minimized.

Chapter 5: False Clustering on Peer-to-Peer Networks. In this Chapter, we study the false clustering problem followed by the description of our clustering algorithm. To evaluate the performance of our algorithm, we use real datasets and PlanetLab test bed [17].

Chapter 6: Conclusions. This Chapter presents the conclusions that ensue from this work and a variety of possible future research lines.

1.5 Selected Publications

This thesis is based on the following publications:

- **Marc Sánchez-Artigas**, Pedro García López, and Antonio F. G. Skarmeta, *Cyclone: A Novel Design Schema For Hierarchical DHTs*. In Proceedings of the 5th Intl. IEEE Conference on Peer-to-Peer Computing (P2P'05), Konstanz, Germany 2005.

- **Marc Sánchez-Artigas**, Pedro García López, and Antonio F. G. Skarmeta, *Cyclone: A Novel Methodology For Constructing Secure Multipath Overlays*. IEEE Internet Computing. November/December 2005.

- **Marc Sánchez-Artigas**, Pedro García López, and Antonio F. G. Skarmeta, *DECA: A Hierarchical Framework for DECentralized Aggregation in DHTs*. In Proceedings of 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management Large Scale Management. Dublin, Ireland, October 2006.

- **Marc Sánchez-Artigas**, Pedro García López, and Antonio F. G. Skarmeta, *A comparative study of Hierarchical DHTs Systems*. In Proceedings of the 32th IEEE Conference on Local Computer Networks (LCN'07). **Conference Best Paper Award**. Dublin, Ireland, October 2007.

- **Marc Sánchez-Artigas**, Pedro García López, and Antonio F. G. Skarmeta, *On the Feasibility of Dynamic Superpeer Ratio Maintenance*. In Proceedings of the 8th Intl. IEEE Conference on Peer-to-Peer Computing, (P2P'08), Aachen, Germany, September, 2008.
- **Marc Sánchez-Artigas**, Pedro García López, and Antonio F. G. Skarmeta, *TR-Clustering: Alleviating the Impact of False Clustering on P2P Overlay Networks*. Accepted for publication in The International Journal of Computer and Telecommunications Networking (Computer Networks, Elsevier)
- Jordi Pujol Ahulló, Pedro García López, **Marc Sánchez-Artigas** and Antonio F.G. Skarmeta, *Supporting Geographical Queries onto DHTs*. In Proceedings of the 33rd IEEE Conference on Local Computer Networks (LCN'08). Montreal, Canada, October 2008.

2
Background

2.1 Peer-To-Peer

We distinguish between unstructured and structured P2P systems. First, we will focus on structured P2P systems and introduce Chord. Then, we will give an overview of the two main hierarchical designs for structured P2P systems.

2.1.1 Unstructured P2P Overlay Networks

In unstructured P2P overlays, such as Gnutella [2], peers use flooding or random walks to resolve queries. These routing techniques can be used for complex searches since they are not limited to indexed data in the network. The main problem of these systems is that search cost does not scale well, as it grows linearly with the size of the network. Since one of our main goals is to improve even more search scalability, we will not further consider unstructured P2P systems in this thesis. Nevertheless, it must be noted that unstructured P2P systems are superior to structured systems in maintenance complexity. Since each peer can possibly select any other peer as a neighbor, the goal of the maintenance strategy is to keep the number of neighbors equal to a pre-specified value, which is economic to do in terms of communication.

For example, in Gnutella, after a pre-specified time period, each peer initiates a broadcast message with a certain time-to-live to discover new online peers. If a peer that receives the message is still active, it should confirm its active presence by sending a reply back which is called a pong message. The peer sender waits for a specified time period for the pong messages to arrive. From the set of active peers, it selects new neighbors only in the case that some of its current neighbors have gone offline.

2.1.2 Flat Structured P2P Overlay Networks

We start this section by first introducing an abstract model for flat structured overlay networks. In general terms, a flat structured network, also called Distributed Hash Table (DHT), can be defined by a a set of peers $\mathcal{P}$ that cooperate together to provide access to a set of objects $\mathcal{O}$. The access is provided by means of an identifier space $\mathcal{I}$ and mappings: $F_{\mathcal{P}} : \mathcal{P} \longrightarrow \mathcal{I}$ and $F_{\mathcal{O}} : \mathcal{O} \longrightarrow \mathcal{I}$. These mappings establish the assignment of objects to peers.

The mapping $F_{\mathcal{P}} : \mathcal{P} \longrightarrow \mathcal{I}$ splits $\mathcal{I}$ into disjoint partitions. At any instant, the current set of identifiers defines the current set of partitions in which the identifier space has been divided into. Each peer is the manager of a distinct partition. By manager we mean that each peer is responsible for all objects whose identifier is mapped into that partition according to $F_{\mathcal{O}} : \mathcal{O} \longrightarrow \mathcal{I}$.

The distributional properties of $F_{\mathcal{P}}$ and $F_{\mathcal{O}}$ have a critical impact on load balancing. For this reason, our framework focuses on the effects that these two mappings have upon our hierarchical designs.

To maximize load balancing, structured P2P overlays use a hash function to insert both peers and objects into the identifier space. Cryptographic hash-functions like MD5 [7] or SHA1 [8] are usually used to map arbitrary strings to 128-bit or 160-bit keys, respectively.

Associated to the identifier space, structured P2P systems define a distance metric $d : \mathcal{I} \times \mathcal{I} \longrightarrow \mathbb{R}$, which is used to route greedily towards the manager of a key. Distance metric d should satisfy the following three properties:

i. $\forall u, v \in \mathcal{I} : d(u, v) \geq 0$.

ii. $\forall u \in \mathcal{I} : d(u, u) = 0$.

iii. $\forall u, v \in \mathcal{I} : d(u, v) = 0 \Rightarrow u = v$.

Very frequently, d satisfies the triangle inequality: $\forall u, v, z \in \mathcal{I} : d(u, z) \leq d(u, v) + d(v, z)$. To illustrate, examples of distance metrics are the Euclidean distance adopted in CAN [9], the length of the common prefix adopted in Pastry [5], Tapestry [10], and Kademlia [6] (also known as XOR metric in the latter), or the clockwise distance between two ids on the circle $[0, 2^n - 1]$ modulo 2^n. Clockwise distance is the distance metric specified in Chord [4].

In addition to $F_{\mathcal{P}}$, $F_{\mathcal{O}}$ and d, three other elements characterize a structured overlay. The first is the management protocol of the identifier space. The second one is the graph topology embedded into the identifier space and finally, the routing algorithm for locating an object.

Management of the identifier space: The management protocol is responsible for durability and availability of objects. More formally, it can be determined by the mapping $F_{\mathcal{M}} : \mathcal{I} \longrightarrow 2^{\mathcal{P}}$ that associates to every object name in $\mathcal{I}$, the set of managers of this object. To provide fault-tolerance, the returned set by mapping $F_{\mathcal{M}}$ typically includes more than one peer, i.e., more than one peer is responsible for managing each object. In some systems, the number of managers for each object is pre-defined. In other systems, the number of managers is variable and it is the result of dynamic load balancing protocols. Independently of the cardinality of the resulting set, the common way to define $F_{\mathcal{M}}$ is based on the distance metric of $\mathcal{I}$. Specifically, the set of M managers for an identifier id in $\mathcal{I}$ is defined as: $\mathcal{M} = \Big\{ m_1 = \text{argmin}_{p \in \mathcal{P}} d(F_{\mathcal{P}}(p), id), m_2 = \text{argmin}_{p \in \mathcal{P} - \{m_1\}} d(F_{\mathcal{P}}(p), id), \ldots, m_M = \text{argmin}_{p \in \mathcal{P} - \{m_1, m_2, \ldots, m_{(M-1)}\}} d(F_{\mathcal{P}}(p), id) \Big\}$.

In general, the cardinality of set $\mathcal{M}$ specifies the maximum degree of replication. Two schemes for replication have been proposed:

i. Replication of the contents of manager m_1 at each of the $r \leq M$ other closest managers of id along the identifier space.

ii. Erasure codes for breaking an object into r' smaller-sized objects with the property that any $r < r'$ of the small-sized objects are sufficient to reconstruct the original object, where r and r' are small integers.

See Weatherspoon and Kubiatowicz [18] for comparison between the two schemes.

Graph embedding: Generally, a structured P2P overlay network can be modeled as a (un)directed graph $G(\mathcal{P}, E)$, embedded into the identifier space $\mathcal{I}$, where $\mathcal{P}$ is the set of vertices (peers) and E is the set of edges. This graph is time-dependent, that is, it depends on the set of participating peers $\mathcal{P}$ at each instant of time. This causes peers to change its neighbors in order to preserve the topological properties of the graph, i.e., small diameter, routing load-balancing etc. Denote by $\mathcal{P}(t)$ the set of participating users at time t. Then, the mapping $F_{\mathcal{N}} : \mathcal{P}(t) \longrightarrow 2^{\mathcal{P}(t)}$ defines the neighborhood of each peer in G. That is, for a peer p, $F_{\mathcal{N}}(p)$ represents the set of peers to which p maintains a physical connection with. $F_{\mathcal{N}}(p)$ is a subset of the edges in G.

The neighborhood of a node could be either *deterministic* or *randomized*. In deterministic overlays like Chord [4], for a given set of participants $\mathcal{P}$, $F_{\mathcal{N}}$ establishes only one possible neighborhood for each node. In randomized overlays like Symphony [11], there exist multiple graphs for the same $\mathcal{P}$ and $F_{\mathcal{N}}$. Moreover, neighborhood can be classified into two types with respect to the distance metric:

i. *Short-range neighborhood,* which refers to the immediate neighbors of a node with respect to d. An example could be: $\forall p, q \in \mathcal{P} : d(F_{\mathcal{P}}(p), F_{\mathcal{P}}(q)) \leq d_{min} \Rightarrow q \in F_{\mathcal{N}}(p)$.

ii. *Long-range neighborhood:,* which refers to the relatively distant neighbors of each peer along $\mathcal{I}$. One example are small world networks [19]. These graphs are constructed in such a way that long range connections satisfy the condition: $\Pr[q \in F_{\mathcal{N}}(p)] \propto \frac{1}{d(F_{\mathcal{P}}(p), F_{\mathcal{P}}(q))^D}$, where D is the dimensionality of the identifier space.

The philosophy of separating the short- and the long-range connections stems from the following fact: the short-range links enforce *correctness* on routing, i.e., they ensure that all managers are connected at all times and are able to communicate with each other; whereas the long-range links stand for *efficiency*. They ensure that the path between two nodes requires few hops, at most $\mathcal{O}(\log N)$ hops. The choice of long-range links is particularly conditioned on the trade-off between the number of links per node and the average number of hops. To achieve optimal trade-off, long-range links are selected so that the resulting graph emulates a family of graphs with small degree and small diameter. Examples include hypercubes and small-world networks [19].

This emulation has an important consequence. In order to provide a small diameter, the connectivity of the graph should be preserved at all times. However, due to node arrivals and departures, the graph structure can be deteriorated. For this reason, each graph embedding is accompanied by a maintenance algorithm that repairs its structure in the face of network dynamics.

Routing: routing is the aspect of a structured overlay that mostly characterizes its performance. Each overlay network has its own routing strategy. In general, a routing strategy can be described as follows:

Definition 1. *Let $\mathcal{I}$ be the identifier space of a structured overlay network. For a peer p, let $F_{\mathcal{N}}(p)$ be the neighbors of p in this overlay. Let id be an identifier in $\mathcal{I}$. A routing strategy, $R : \mathcal{P} \times \mathcal{I} \longrightarrow 2^{\mathcal{P}}$, selects for each peer p in this overlay the next peer $q \in F_{\mathcal{N}}(p)$ to which a request for id will be forwarded.*

For structured overlays, the most representative routing strategy is GREEDY routing. In GREEDY routing, each peer forwards a requests to the neighbor that minimizes the remaining distance to the destination with respect to distance metric d.

With the concepts presented above, we are ready to provide a simple definition that describes what a flat overlay is. Although not all aspects are considered in our definition, we believe that it is illustrative enough to characterize *any* DHT design.

Definition 2. *A flat, structured overlay design, D, can be mainly described by the tuple*

$$D := \langle \mathcal{I}, d, F_{\mathcal{P}}, F_{\mathcal{O}}, F_{\mathcal{M}}, F_{\mathcal{N}}, R \rangle$$

where

- *$\mathcal{I}$ denotes the identifier space of this design;*
- *d is the distance metric associated with $\mathcal{I}$;*

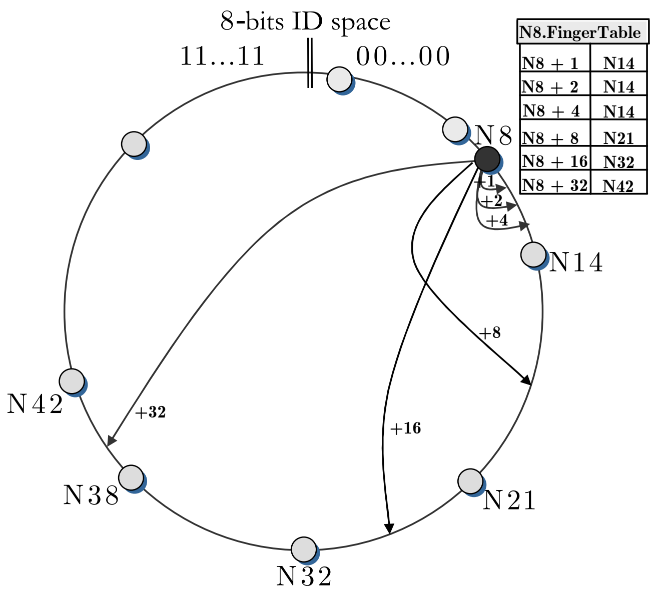


Figure 2.1: *Chord's graph embedding: the neighborhood (finger table) for node* 8 *is shown in the figure.*

- $F_{\mathcal{P}}$ *is the mapping of the participating peers into* $\mathcal{I}$*;*
- $F_{\mathcal{O}}$ *denotes the mapping of the objects into* $\mathcal{I}$*;*
- $F_{\mathcal{M}}$ *returns the set of managers for each identifier in* $\mathcal{I}$*;*
- $F_{\mathcal{N}}$ *determines the out-going links for each participating peer; and*
- R *is the routing strategy associated with design* D*;*

Chord

Now we describe Chord [4], one of the earliest DHTs that were developed. In our description, we instantiate all the aspects included in Definition 2.

The identifier space $\mathcal{I}$ of Chord consists of finite strings on a binary alphabet. The length of the identifiers is exactly $n = 160$ bits and they are ordered on an identifier circle modulo 2^n, labeled from 0 to $2^n - 1$.

The mapping $F_{\mathcal{P}} : \mathcal{P} \longrightarrow \mathcal{I}$, which assigns each peer an identifier in $\mathcal{I}$, is implemented using secure hashing function SHA-1 [8]. Consequently, random partitions are assigned to each joining peer, without taking into account the current partitions and loads imposed on participating nodes.

The mapping $F_{\mathcal{O}} : \mathcal{O} \longrightarrow \mathcal{I}$, which maps an object to an identifier (also called *key*) in $\mathcal{I}$, is also realized through SHA-1. The main benefit of this choice is that objects are uniformly distributed along the identifier space. As illustrated in [4], for any set of $|\mathcal{P}|$ peers and $|\mathcal{O}|$ objects, the use of SHA-1 provides the following two guarantees with high probability:

i. Each node is responsible for at most $(1 + \epsilon)|\mathcal{O}|/|\mathcal{P}|$ keys (for small ϵ).

ii. When an $(|\mathcal{P}| + 1)^{st}$ peer joins or leaves the network, responsibility for $\mathcal{O}(\frac{|\mathcal{O}|}{|\mathcal{P}|})$ keys changes hands (and only to or from the joining or leaving node).

The distance metric defined on $\mathcal{I}$ is the *clockwise distance*. More formally, given two peers with identifiers u and v, the clockwise distance, $d_{clockwise}(u,v)$, between them is

$$d_{clockwise}(u,v) = \begin{cases} v-u & \text{if } v \geq u \\ 2^n + v - u & \text{otherwise} \end{cases}$$

Specifically, distance metric $d_{clockwise}(u,v)$ gives actually the number of points (ids) found on the circle when moving in clockwise direction from the identifier of u to v. Most remarkably is the fact that Chord's identifier space is not symmetric with respect to clockwise distance, which means that $d_{clockwise}(u,v) \neq d_{clockwise}(v,u)$ (except for the case $d_{clockwise}(u,v) = 2^{n-1}$). More precisely, $d_{clockwise}(u,v) = 2^n - d_{clockwise}(v,u)$.

Management to the identifier space: We start by giving some definitions necessary for presenting the concept of management of the identifier space in Chord.

Definition 3. *A peer p is said to the predecessor of an identifier id in the Chord circle, denoted as $pred(id)$, if $d_{clockwise}(F_{\mathcal{P}}(p), id) = \min_{q\in\mathcal{P}} d_{clockwise}(F_{\mathcal{P}}(q), id)$. A peer p is said to be the predecessor of another peer q if $p = pred(F_{\mathcal{P}}(q))$.*

Analogously,

Definition 4. *A peer p is said to the successor of an identifier id in the Chord circle, denoted as $succ(id)$, if $d_{clockwise}(id, F_{\mathcal{P}}(p)) = \min_{q\in\mathcal{P}} d_{clockwise}(id, F_{\mathcal{P}}(q))$. A peer p is said to be the successor of another peer q if $p = succ(F_{\mathcal{P}}(q))$.*

The mapping $F_{\mathcal{M}} : \mathcal{I} \longrightarrow 2^{\mathcal{P}}$ is defined as follows: a peer p is responsible for all objects whose identifiers are hashed onto the arc from $F_{\mathcal{P}}(p)$ to the identifier of the closest node that precedes p on the circle. Formally, a peer $p \in F_{\mathcal{M}}(id)$ for all the identifiers in the range $[F_{\mathcal{P}}(pred(p)), F_{\mathcal{P}}(p))$. Moreover, Chord replicates all the objects whose hash-name falls into range $[F_{\mathcal{P}}(pred(p)), F_{\mathcal{P}}(p))$ to the $r = \Omega(\log N)$ successors of p, i.e., $F_{\mathcal{M}}(id) = \{p, succ(p), succ(succ(p)), \ldots, succ^r(p)\}$, where $succ^r(p)$ is the result of applying recursively function $succ()$ r times starting at p.

Graph embedding: the neighborhood set, $F_{\mathcal{N}}(p)$, of a peer p consists of the $pred(p)$, the $succ(p)$ and the set of long-range contacts, $Fingers(p)$, also known as the finger table of p. Hence,

$$F_{\mathcal{N}}(p) = \{pred(p)\} \cup \{succ(p)\} \cup Fingers(p)$$

$Fingers(p)$ contains at most n fingers and equals to $\{succ(F_{\mathcal{P}}(p) + 2^{i-1} \pmod{2^n})\}_{1\leq i\leq n}$. Informally, each peer partitions the identifier space by means of the fingers into at most n partitions. The starting point of the i^{th} partition is the id of $succ(F_{\mathcal{P}}(p) + 2^{i-2} \ (mod\ 2^n))$ while the end point is the id of $succ(F_{\mathcal{P}}(p) + 2^{i-1} \ (mod\ 2^n))$, respectively. Fig. 2.1 shows the fingers of peer 8. The first finger of peer 8 points to peer 14, since peer 14 is the first peer that succeeds $(8+2^0)\ (mod\ 2^6) = 9$. Similarly, the last finger of peer 8 is peer 42, since it is the first peer going clockwise that succeeds $(8+2^5)\ (mod\ 2^6) = 40$.

The maintenance strategy for Chord consists of two algorithms:

i. *The stabilization algorithm:* each peer periodically runs the stabilization algorithm to maintain up to date the information about its immediate successor on the circle. Each time a peer runs this protocol, it asks its successor about which peer is its predecessor and learns if there is a new peer that should become its successor.

ii. *The fix_fingers algorithm:* each peer periodically calls *fix_fingers*. In every call to this function, *fix_fingers* refreshes exactly one finger. To repair all fingers, *fix_fingers* remembers the index i of the last finger refreshed, and in a new call, it repairs the $(i+1)^{th}$ finger.

In addition, each node executes an algorithm for periodically checking its predecessor.

Routing: the routing strategy of Chord is a natural consequence of how its identifier space is partitioned. Its functionality is to find the successor of a key k when trying to locate an object with this key. The standard Chord routing protocol works as follows: upon receiving a message for a key k, a node p forwards the query to the neighbor (finger) whose id precedes most immediately (or is equal to) key k. Formally, it can be described as follows:

Definition 5 (Clockwise GREEDY Routing). *In a Chord graph with distance function $d_{clockwise}$, GREEDY routing entails the following decision: Given a key k, a node p with neighbors $F_{\mathcal{N}}(p)$ forwards a message to its neighbor $q \in F_{\mathcal{N}}(p)$ such that $d_{clockwise}(q,k) = \min_{x \in F_{\mathcal{N}}(p)} d_{clockwise}(x,k)$.*

2.1.3 Hierarchical Structured P2P Overlay Networks

Shortly after the appearance of flat overlay networks, it became obvious that uniformity prevents flat overlays to improve its performance. To address this issue, several hierarchical designs were rapidly proposed as basis for dropping such an assumption. Sometimes, the common characteristic of these designs was only the organization of peers into two or more tiers.

In this section, we provide a conceptual description of existing hierarchical designs. In general terms, they can be classified into to major groups: superpeer systems and homogeneous systems. This classification is based upon whether a hierarchical design prefers to optimize load balancing or to assign responsibility in proportion to the capacity of peers.

In what follows, we describe the two families in conjunction with an example of each category. We start with the description of superpeer systems.

Superpeer Systems

Generally, superpeer systems divide peers into two tiers, mostly because of the simplicity in their design. Their main characteristic is that they attempt to assign responsibility in proportion to the capacity of a peer. As a result, they classify peers into two tiers: the supertier, where high-capacity peers, referred to as superpeers, process requests coming from low-capacity peers at the regular tier.

Since there is no a generic design for superpeer systems, no formal definition can be provided without the risk to blur the singular aspects of each design. Consequently, we describe the general aspects of superpeer systems and provide a tentative definition.

Let $\mathcal{S}$ be the subset of high-capacity peers or superpeers. Then, a first definition of a superpeer system is:

Definition 6. *A superpeer system $\mathcal{H}$ can be described by the quadruplet $H := \langle \mathcal{D}_{\mathcal{S}}, \mathcal{D}_{\mathcal{P}}, F_{\mathcal{N}_{\mathcal{S}}}, F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}} \rangle$, where $\mathcal{D}_{\mathcal{S}}$ is the overlay design at the supertier, $\mathcal{D}_{\mathcal{P}}$ is the overlay design at the regular tier, the mapping*

$F_{\mathcal{N}_\mathcal{P}} : \mathcal{P} - \mathcal{S} \longrightarrow 2^{\mathcal{S}}$ defines the connections from peers to superpeers, and the mapping $F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}} : \mathcal{S} \longrightarrow 2^{\mathcal{P}-\mathcal{S}}$ defines the connections from superpeers to peers.

Usually, peers provide cooperative access to a set of objects $\mathcal{O}$ by virtue of functions $\mathcal{F}_\mathcal{P}$ and $\mathcal{F}_\mathcal{O}$. However, since each overlay can maintain a separate identifier space, $\mathcal{F}_\mathcal{P}$ and $\mathcal{F}_\mathcal{O}$ should be adapted to each layer. Analogously, the management of the identifier space should be considered at two levels. At the regular tier, objects are assigned to peers as in flat DHTs. At the supertier, objects are assigned to superpeers. The basic idea is that any superpeer can find any object routing through the supertier. This mapping is analogous to the mapping $\mathcal{F}_\mathcal{M}$ defined for flat overlays.

Graph embedding: in addition to neighborhood mappings $F_{\mathcal{N}_\mathcal{P}}$ and $F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}}$, there are two additional neighborhood relationships:

i. The mapping $F_{\mathcal{N}_\mathcal{S}} : \mathcal{S} \longrightarrow 2^{\mathcal{S}}$, which establishes the connections among superpeers; and

ii. The mapping $F_\mathcal{N} : \mathcal{P} \longrightarrow 2^{\mathcal{P}-\mathcal{S}}$, which defines the links among regular peers.

If all these mappings are defined, superpeer design leads to the following out-degrees. For a regular peer p, the out-degree is $|F_{\mathcal{N}_\mathcal{P}}(p)| + |F_\mathcal{N}(p)|$. The out-degree of a superpeer s is $|F_{\mathcal{N}_\mathcal{S}}(s)| + |F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}}(s)|$.

Routing: the implementation of the basic communication primitives `Get` and `Put` depends on the type of the peer. Most superpeer systems have been designed in such a way that superpeers only assume the responsibility for routing, although there are other systems in which superpeers are responsible for storing objects as well. As for flat overlays, the routing strategy can be defined by function $R : \mathcal{P} \times \mathcal{I} \longrightarrow 2^{\mathcal{P}}$, where $\mathcal{P}$ also includes the set of superpeers.

For a superpeer s, this function specifies to which neighbor s should forward a request. More specifically, s decides whether to forward to a superpeer from $F_{\mathcal{N}_\mathcal{S}}(s)$ or to a peer from $F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}}(s)$. This decision is specific to the overlay design.

For a regular peer p, the decision is also whether to forward to a superpeer or to a regular peer. Technically, this means to select from $|F_{\mathcal{N}_\mathcal{P}}(p)|$ or from $|F_\mathcal{N}(p)|$. The way this decision is made is specific for each overlay network.

Examples of superpeer systems that match this definition are Brocade [20] and the hierarchical design of Mizrak et. al. [21]. The core idea behind these designs is to build a superpeer overlay on top of a flat DHT to improve routing performance. However, there are other superpeer systems which do not adhere to this definition, as their intention is to reflect administrative domains rather than improving routing performance. Since these designs do not completely match Definition 6, we now provide a detailed description of one example to complete our discussion on superpeer systems.

Garcés-Erice et. al. Hierarchical Design

Garcés-Erice et. al. proposed a superpeer-based model in [22], attracted by the idea of hierarchical routing in the Internet.

To support multiple domains, this model divides peers into several groups where each group can form its own structured overlay: Chord, Tapestry etc. In each group one or more superpeers

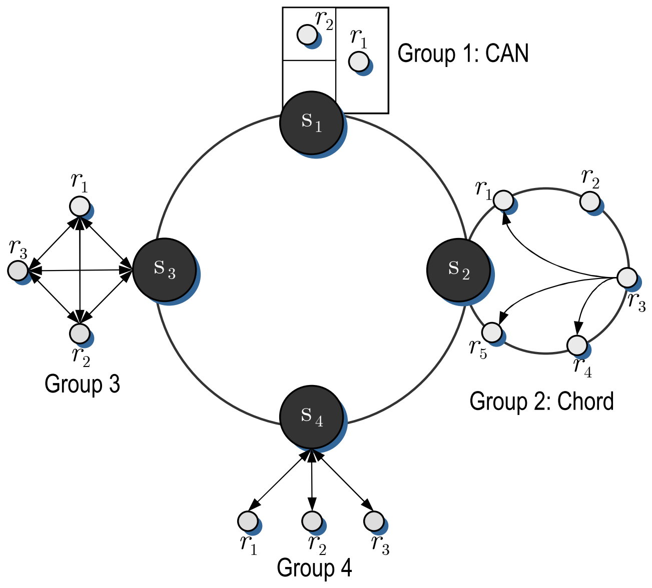


Figure 2.2: *An example of a hierarchical organization in Garces-Erice et al.*

are chosen to participate in a superpeer overlay, where superpeers behave as proxies sending and receiving messages on behalf of the peers in their groups.

In this discussion, we assume that the superpeer overlay is a Chord DHT, as described in [22]. As a result, the diameter in this system will be characterized by $\mathcal{O}(\log|\mathcal{S}|)$ hops, where $|\mathcal{S}|$ denotes the number of superpeers. An illustration of this two-level hierarchy is given in Fig. 2.2

In general, the identifier space $\mathcal{I}$ of this model can be viewed as the union of g subsets $\mathcal{I}'_i$ and $\mathcal{I}_s$, the identifier space of the superpeer overlay. More formally, $\mathcal{I} = \mathcal{I}_s \cup [\bigcup_{i=1}^{g} \mathcal{I}'_i]$, where each $\mathcal{I}'_i$ corresponds to the identifier space of each group. Since each group is free to form an independent overlay, this hierarchical design cannot assume two global mappings for $F_{\mathcal{P}}$ and $F_{\mathcal{O}}$. A reasonable decision, however, would be to assume that $F_{\mathcal{P}}$ and $F_{\mathcal{O}}$ are realized via a secure hash function in all groups, including the superpeer overlay.

Another feature of this design is that there could be $g+1$ distance metrics: one for the superpeer overlay, d_s, and g intra-group distance metrics, $d_1, d_2, \ldots, d_g$. For the superpeer overlay, we assume that $d_s = d_{clockwise}$. Recall that Chord is the overlay used in the supertier.

Management of identifier space: in each intra-group, objects are managed by peers according to the specific structured overlay chosen for each group. Hence, the mapping $F_{\mathcal{M}}$ can be viewed as a complex function defined separately on each subdomain.

Let $\mathcal{S}$ denote the set of superpeers in the system. For ease of explanation, we assume that the mapping $F_{\mathcal{S}} : \mathcal{I} \longrightarrow 2^{\mathcal{S}}$, which defines how the identifier space is managed by superpeers, equals to the mapping $F_{\mathcal{M}}$ of Chord. In practice, intra-groups are assumed to be administrative domains such as university campuses.

Graph embedding: the whole set of peers $\mathcal{P}$ is divided into g disjoint subsets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_g$. A possibly different overlay network is defined on each subset. Those overlay networks are referred to as intra-groups.

To ease exposition, we assume the existence of a mapping $F_{\mathcal{G}} : \mathcal{P} \longrightarrow \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_g\}$, which returns the intra-group for each participating peer. We note that this function is not part of the model defined in [22]. We use it to simplify notation.

Each regular peer p creates its neighbor set, $F_{\mathcal{N}}(p)$, according to the linking rules of the specific intra-group overlay with the following condition: $\forall q \in F_{\mathcal{N}}(p) : q \in F_{\mathcal{G}}(p)$, where $F_{\mathcal{G}}(p)$ denotes the intra-group of p.

Obviously, one or more peers from each intra-group are promoted to the supertier; they are referred as superpeers. The function $F_{\mathcal{N}_{\mathcal{S}}} : \mathcal{S} \longrightarrow 2^{\mathcal{S}}$ defines the neighborhood sets at the supertier. The general model in [22] allows the use of any possible structured overlay as the superoverlay. However, as defined in [22], the authors assume that the overlay at the supertier is Chord. For a superpeer s, this means that $F_{\mathcal{N}_{\mathcal{S}}}(s) = \{pred_{\mathcal{S}}(s)\} \cup \{succ_{\mathcal{S}}(s)\} \cup Fingers_{\mathcal{S}}(s)$, where the subscript $\mathcal{S}$ remarks that the predecessor, the successor and fingers of s are superpeers.

Each superpeer s needs also to maintain a reference to some peers in its intra-group. As before, its intra-group neighborhood, $F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}}(s)$, depends on the specific intra-group overlay design. For instance, a superpeer s in a Chord like intra-group will maintain a set $F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}}(s)$ of $\mathcal{O}(\log |F_{\mathcal{G}}(s)|)$ peers. In a star like intra-group, s will maintain connections to $|F_{\mathcal{G}}(s)| - 1$ peers.

Routing: when a regular peer r receives or initiates a request for an object identified by id, then it must distinguish between two cases:

i. if id is managed by a peer from its intra-group, then r can either forward directly or through a superpeer. In Group 3 in Fig. 2.2, each peer knows about all peers in its intra-group and can therefore immediately contact the target peer. In Group 2, which organizes peers into a Chord overlay, peer r can route the request using its neighborhood set, $F_{\mathcal{N}}(r)$. Recall that in Chord, a peer r forwards a message to its neighbor $q \in F_{\mathcal{N}}(r)$ such that $d_{clockwise}(q, id) = \min_{x \in F_{\mathcal{N}}(r)} d_{clockwise}(x, id)$. Group 4 has a star-like organization and in this case, r forwards each request to its superpeer.

ii. Otherwise, the request is routed through a superpeer in r's intra-group. When a superpeer s receives a request for a identifier id, it first decides upon whether this request is for its own intra-group or belongs to a distinct group. Depending on this decision, s forwards either to a peer in its intra-group or to another superpeer. Let $R_{\mathcal{S}}(s, id)$ be the routing decision taken at superpeer s when receiving a request for identifier id. Then, $R_{\mathcal{S}}(s, id)$ can be formulated as follows:

$$R_{\mathcal{S}}(s, id) = \begin{cases} r' \quad |d_{F_{\mathcal{G}}(s)}(r', id) = \min_{x \in F_{\mathcal{N}_{\mathcal{P}-\mathcal{S}}}(s)} d_{F_{\mathcal{G}}(s)}(x, id) & \text{if } d_s(s, id) = \\ & = \min_{x \in F_{\mathcal{N}_{\mathcal{S}}}(s)} d_s(x, id) \\ s' \quad |d_s(s', id) = \min_{x \in F_{\mathcal{N}_{\mathcal{S}}}(s)} d_s(x, id) & \text{otherwise} \end{cases}$$

where r' is the peer to which s will forward the request within its intra-group, and s' is the superpeer to which it will forward the request in the supertier.

Homogeneous Systems

In contrast to superpeer systems, the main characteristic of homogeneous designs is that all peers are considered as equal. Assumptions on equality are made regarding peers characteristics, such as available bandwidth and storage capacity. As a consequence of this assumption, homogeneous systems can exploit the uniformity of load and functionality that characterizes flat DHTs such as Chord [4], Pastry [5] and Kademlia [6].

Generally, the main feature of homogeneous systems is the use of a single identifier space that is shared by the all peers in the system. This property has three important consequences:

i. there is a single distance metric d in the system;

ii. the mappings $F_{\mathcal{P}}$ and $F_{\mathcal{O}}$ are unique; and

iii. there is only one structured overlay which is virtually shared by all peers.

The difference with flat structured overlays is the use of hierarchical organization to provide additional features such as fault isolation and path locality. Their hierarchical structure, however, is completely opposite to the hierarchical structure of superpeer systems. Mainly, the difference is that each peer is contained in a collection of telescoping domains, rather than in a single separate domain. More formally, their hierarchical organization is typically as follows (a similar definition is provided in [23]):

Let $\mathcal{H}$ be a collection of sets $D \subseteq \mathcal{P}$. Clearly, each set $D \in \mathcal{H}$ is a subset of peers. As usual, we refer to D as a domain. Any domain that does not contain another domain in $\mathcal{H}$ is said to be a leaf domain. Then, the hierarchical structure of homogeneous designs have the following properties:

- $\mathcal{P} \in \mathcal{H}$.
- For any pair of domains in $\mathcal{H}$, the two domains are either disjoint, or one domain is a proper subset of the other.
- For each domain $D \in \mathcal{H}$, if D is not a leaf domain, then D is the union of a finite number of sets in $\mathcal{H}$.

By the above definition, one can easily see that relation '$\subset$' defines a partial ordering over the domains in $\mathcal{H}$, generating a partial-order tree with multiple tiers. All the participating peers are included at tier-0, the highest tier. At tier-1, the single domain $D = \mathcal{P}$ at tier-0 is split into a finite number of domains, $D_1, D_2, \ldots, D_m$, such that for all $i = 1 \ldots m$, $D_i \subset D$. The rest of tiers can be recursively defined in the same manner.

The main consequence of this form of organization is that each peer has the same identifier in all domains to which it belongs; as discussed in [24], one can view a peer as projecting its presence to the same location in each of its domains. Further, it conditions the following aspects:

Management of the identifier space: in any homogeneous overlay, ids are associated to managers according to the managing rule of the original structured overlay. Although the managing policy is the same for all domains, the set of managers for a specific identifier varies at each tier. This is caused by the telescoping property of non-leaf domains. Because each non-leaf domain comprises a finite number of lower-tier domains, the partitioning of the identifier space as established by $F_{\mathcal{P}}$ in that domain differs from those in the lower-tier domains. As a result of this repartitioning, the identifiers are reassigned to managers. In this sense, the mapping $F_{\mathcal{M}}$ can be viewed as a complex function defined separately for each domain.

Graph embedding: from a structural viewpoint, homogeneous design can be viewed as a single structured overlay made up of smaller structured overlays. Consequently, $F_{\mathcal{N}}$ must be considered separately for each domain, as each domain forms its own overlay. The unique constraint is that $F_{\mathcal{N}}$ must adopt exclusively the linking rules of the chosen structured overlay in all domains.

Routing: let h be the number of tiers in a homogeneous system. Generally, routing operation proceeds in h loops. In the first loop, a query for an identifier id starts at the lowest domain. Upon the reception of the query, each peer uses the routing strategy of the flat structured overlay to route on that domain. Inside that domain, routing continues until the peer p that satisfies $id \in F_{\mathcal{M}}^{-1}(q)$ is found. If peer p does not store the object, the routing operation continues in the next higher tier. The same operation is repeated until a manager holding id is found. At the last loop, the routing procedure is executed on the largest domain which spans all peers in the system. Definitely, the routing procedure stops at one manager of id, irrespective if this manager stores the object whose identifier is id.

Examples of homogeneous designs are HIERAS [25], Coral [24], Canon [13] and TOPLUS [23]. In the following, we provide a description of Coral in order to clarify concepts.

Coral

Coral is a P2P content distribution network that was proposed in [24]. While attempting to reduce the latency of `Get` and `Put` operations, Coral inadvertently proposed the earliest homogeneous hierarchical design. In this section, we characterize this design.

Since it belongs to the family of homogeneous designs, all domains share a unique identifier space $\mathcal{I}$. In Coral, this space consists of strings on a binary alphabet. The length of the identifiers is $n = 160$ bits, such that the whole identifier space consists of 2^{160} binary strings.

As discussed above, the uniqueness of the identifier space enables the following simplification: the adoption of two global mappings for $F_{\mathcal{P}}$ and $F_{\mathcal{O}}$. Specifically, $F_{\mathcal{P}}$ and $F_{\mathcal{O}}$ are implemented via a hash function. This implies that:

i. Each peer p has the same identifier $F_{\mathcal{P}}(p)$ in all tiers; and

ii. Each object o has the same hash-name $F_{\mathcal{O}}(o)$ in all levels.

In addition, the use of a single identifier space restricts the number of distance metrics to one. That is, there is a single distance metric d defined on $\mathcal{I}$ for all domains. Specifically, d corresponds to XOR metric, d_{XOR}. This metric defines the distance between two identifiers u and v as

$$d_{XOR}(u, v) = \sum_{i=0}^{n-1} |u_i - v_i| 2^i$$

where u_i and v_i denote the i^{th} bit of identifiers u and v, respectively. d_{XOR} has the following properties:

i. $\forall u, v \in \mathcal{I} : d_{XOR}(u, w) = d(v, w)$ for all $w \in \mathcal{I} \Rightarrow u = v$.

ii. $\forall u, v \in \mathcal{I} : d_{XOR}(u, v) \leq 2^n - 1$.

iii. Let $\pi(u, v)$ be the number of bits in the common prefix of u and v. $\forall u, v \in \mathcal{I} : \pi(u, v) = m \Rightarrow d_{XOR}(u, v) \leq 2^{n-m} - 1$.

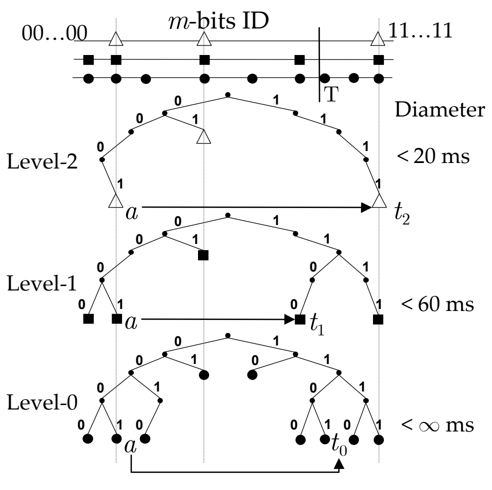


Figure 2.3: *An example of Coral routing. Higher-tier, smaller-diameter domains are naturally sparser. For a query on identifier id, the querying node a first routes on its highest domain. The routing fails on that tier if the node closest to id on the identifier space, node t_2, does not store the object. If this occurs, Coral continues routing on a lower tier domain, having already traversed the identifier space up to t_2.*

iv. $\forall u, v, w \in \mathcal{I} : d_{XOR}(u, w) \leq d_{XOR}(v, w) \Rightarrow \pi(u, w) \geq \pi(v, w)$.

Observe that d_{XOR} is a refinement of longest-prefix matching. If u is the unique longest-prefix match with v, then identifier u is the closest to v in terms of the metric. Further, if two peers share the longest matching prefix, the metric will break the tie.

Management of the identifier space: in Coral, objects are managed by peers according to Kademlia [6], the specific structured overlay chosen to represent the hierarchy. As a result, $F_{\mathcal{M}}$ is based on the XOR metric, d_{XOR}. Inside each domain, the manager of an identifier id is the peer p in that domain that minimizes $d_{XOR}(F_{\mathcal{P}}(p), id)$.

Graph embedding: for locality-optimized routing, Coral divides the entire set of participants $\mathcal{P}$ into three tiers of domains. Each peer participates in at least one domain from each tier. Domains are characterized by a threshold on round-trip-time. At the highest tier, the peers are the *physically* closest. At each successive tier, the average round-trip-time between all pairs of peers gradually increases.

Since each peer projects its presence to the same location at each tier, the mapping $F_{\mathcal{N}} : \mathcal{P} \longrightarrow 2^{\mathcal{P}}$ can be viewed as a complex function defined separately for each domain. For a peer p, denote by $F_{\mathcal{N}}(p, i)$ the neighbor set of p at tier-i. Then, $F_{\mathcal{N}}(p)$ can be defined as $\bigcup_{i=0}^{h} F_{\mathcal{N}}(p, i)$, where h is the height of the hierarchy. In the original Coral work [24], $h = 3$, although there is no explicit reason to restrict the number of tiers to 3. Each subset $F_{\mathcal{N}}(p, i)$ is determined according to d_{XOR}.

Routing: as in all homogeneous designs, routing is recursive. It starts at the highest tier in order to take advantage of network locality. If the object is not found in that domain, search operation reaches the peer in that domain that is closest to the target identifier w.r.t. d_{XOR}. Then the process continues from that peer in the next lower level. This process is repeated until a manager holding the object is found.

Technically, this means that the routing strategy is the same as in Kademlia, but successively applied at each tier. Without loss of generality, let us consider that a query is routed through a tier-

i domain. For a peer p, the next hop is the neighbor q s.t. $d_{XOR}(q, id) = \min_{x \in F_{\mathcal{N}}(p,j)} d_{XOR}(x, id)$, where

$$j = \begin{cases} i - 1 & \text{if } p \text{ is s.t. } d_{XOR}(p, id) = \min_{x \in F_{\mathcal{N}}(p,i)} d_{XOR}(x, id) \text{ and } p \text{ does not store } id \\ i & \text{otherwise} \end{cases}$$

An example of Coral routing is shown in Fig. 2.3.

3 CYCLONE: A FRAMEWORK FOR DESIGNING HIERARCHICAL DHTs

In this Chapter, we present a hierarchical model for DHTs which benefits from the recursive structure that many DHT topologies inadvertently own. We devise a novel group-theoretic technique based on Cayley graph model for converting a flat DHT into a hierarchical DHT almost at no cost. While retaining the uniformity, scalability and load balancing of the original designs, our framework increases their scalability through a hierarchical design. The reason is that hierarchies constitute a nice method to accommodate growth and isolate faults. We show how Chord can be converted into a hierarchical DHT using our methodology. Furthermore, we give some indicative hints of how six different DHTs — Randomized Chord, Symphony, Kademlia, P-Grid, Tapestry and Pastry — can be transformed into their hierarchical versions. Simulations results verify the effectiveness of our framework.

3.1 Introduction

"The trees that are slow to grow bear the best fruit"'

Moliere (1622-1673)

Distributed Hash Tables (DHTs) such as Chord [4], Kademlia [6], Pastry [5] and Tapestry [10], have recently arisen as general infrastructures for building large-scale distributed systems. DHTs have many appealing virtues such as decentralization, self-organization, and fault-tolerance. However, yet, it is unclear how to implement certain type of applications and services using DHTs. For instance, DHTs have poor locality. Although some DHTs such as Pastry make an effort to route queries through nodes with low network latency, the last few hops in any query are essentially random. This is of particular concern for content delivery networks (CDNs), which require the end-to-end latency from the source to each receiver to be low.

The traditional approach to tackle the above flaw revolves around the hope to organize the nodes hierarchically. By dividing the network into smaller groups, the system complexity reduces as well. As typical examples, DNS or group multicast achieve scalability via a hierarchical design. In the peer-to-peer context, hierarchical designs provide new forms of routing that can potentially achieve significant savings in query time. One of these forms of routing is *Path-Constrained routing*. Path-Constrained routing can be described as follows:

Definition 7 (Path-Constrained Routing)**.** *Given a query for a key k originated at a node belonging to cluster C, Path-Constrained Routing refers to the routing in which each hop is constrained within cluster C and terminates at the node responsible for each k in this cluster.*

This form of routing has been adopted by hierarchical systems such as Coral [24] and Canon [13], albeit they do not refer to this form of routing with this denomination.

As a representative example, we describe Coral [24], a decentralized, peer-to-peer web-content distribution network. Coral exploits DNS redirection, together with a "sloppy" DHT abstraction to redirect each client to a close copy of the requested web page, hosted on some volunteer node.

To attain this goal, Coral uses three levels of clusters and allows a set of nodes to form a cluster only if their average, pairwise round-trip-times (RTTs) are below some threshold. More precisely, Coral specifies thresholds of ∞ ms, 60 ms, and 20 ms for level-0, -1, and -2 clusters, respectively. The immediate consequence of this is that clients can first query the servers in higher-level, fast clusters before those in lower-level, slower clusters. This both reduces the latency of `Get` and `Put` operations and increases the chances of returning values stored by nearby nodes.

An example of Coral hierarchical routing is shown in Fig. 3.1. For a `Get` on key k, the querying node first applies Path-Constrained routing on its highest cluster to benefit from network proximity. The routing fails on that level if the node closest to key k on the identifier space, node t_2, does not store the key. If this hapens, Coral continues routing on a lower-level cluster, having *already* traversed the identifier space up to node t_2.

3.1.1 Goals

From the preceding discussion, it is clear that a hierarchical design has the potential to improve query throughput. However, there are some issues that must be considered before constructing a hierarchical version of DHT, such as how to inherit the load and functionality of flat designs.

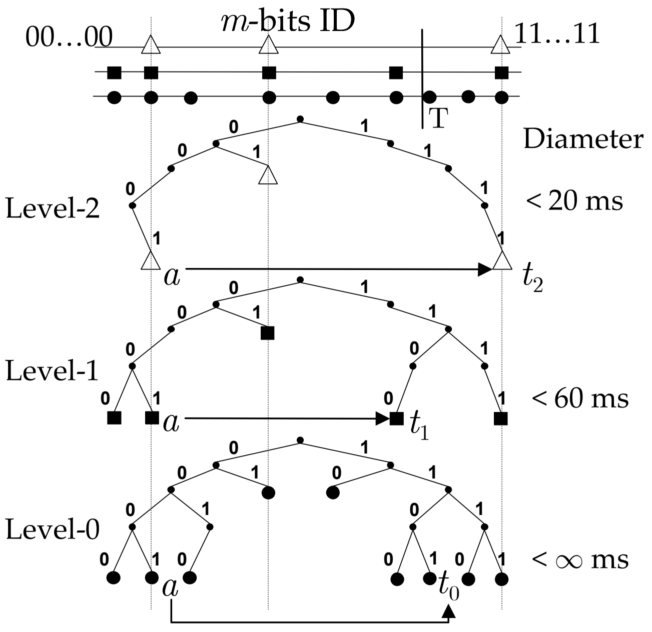


Figure 3.1: *Coral hierarchical routing. Nodes use the same ids in each of their clusters; higher-level, smaller-diameter clusters are naturally sparser. For a lookup on key k, the querying node a first applies Path-Constrained routing on its highest cluster. The routing fails on that level if the node closest to key k on the identifier space, node t_2, does not store the key. If this occurs, Coral continues its lookup on a lower level cluster, having already traversed the identifier space up to t_2.*

For example, the expected in-degree of a Chord node (i.e., the number of routing table entries referring to a given node) is $\mathcal{O}(\log N)$. This property is desirable, as it tends to balance communication load across all computers in the network. However, it requires that peers are uniformly distributed along the identifier space, which limits the possibilities of mapping peers into a hierarchy.

To address this issue, we introduce Cyclone in this Chapter, a framework that combines the homogeneity of load and functionality offered by plain DHTs with the advantages of hierarchical design, which are diverse, but generally aim to improve query efficiency. Put another way, the challenge we face is how to design hierarchical DHTs that obtain the "best" of both worlds, without inheriting the disadvantages of either. To better understand the contributions of this research, we discuss the key aspects of our framework for coming up with "good" hierarchical designs. To ease exposition, we assume a generic DHT (similar to Chord) that has logarithmic degree and diameter. Then, we consider the following key aspects:

- **Efficiency**. Our framework should guarantee short routing paths with small state per node. At most, a diameter of $\mathcal{O}(\log N)$ routing hops. Also, maintenance should be kept minimal, i.e., affecting $\mathcal{O}(\log N)$ number of peers and mostly no burden on anyone.

- **Scalability**. The concept of scalability comprises many aspects. We focus on numerical scalability. Broadly speaking, we want to show that N can grow without performance degradation in our hierarchical designs.

- **Fault-tolerance**. Under normal conditions, the integrity of cluster structures is guaranteed by the *maintenance* protocol. However, due to unpredictable peer and link failures, a node might suddenly become unreachable. To provide reliable operation, our framework should guarantee that the hierarchical version of a DHT is resilient against such failures. Moreover, it should report a maintenance complexity as tight as possible to the one claimed in the flat DHT.

- **Load Balancing**. A basic principle of P2P paradigm is that load should be well-balanced across all nodes in the network. If random nodes call `Put` and `Get`, then the communication load should be equitably distributed among all participant nodes. This includes preserving the randomness guarantee of consistent hashing and the distributional properties of in-degree, which are very relevant for message forwarding.

To meet the above requirements, our framework takes advantage of the following fact:

To maximize load-balancing, DHTs make use of a hash function h to map the set of distributed objects $\mathcal{O}$ and the set of participants $\mathcal{P}$ to the identifier space $\mathcal{I}$. Formally, hash function h defines the relationships $h : \mathcal{O} \longrightarrow \mathcal{I}$ and $h : \mathcal{P} \longrightarrow \mathcal{I}$, respectively. This overlap obscures the independence of two primary concerns:

- On the one hand, the distribution of peers on the identifier space $\mathcal{I}$ (i.e., mapping $F_{\mathcal{P}} : \mathcal{P} \longrightarrow \mathcal{I}$); and
- On the other hand, the distribution of objects (i.e., mapping $F_{\mathcal{O}} : \mathcal{O} \longrightarrow \mathcal{I}$).

In our framework, we consider these two concerns separately. While each cluster has available the whole identifier space for mapping data objects, the space for hosting nodes is partitioned into disjoint clusters. The partitioning is, of course, done according to the recursive decomposition of each graph. This leads to a hierarchical structure in which nodes assume equal roles and take the same duties and responsibilities for all operations. If stability was a mandatory requirement, our hierarchical framework does not impede participation to be allowed only to superpeers such as in TOPLUS [23]. More precisely, what we provide in this thesis is a generic framework for building hierarchies of telescoping clusters (i.e. clusters of ... of clusters of nodes) such as in TOPLUS [23], Coral [24] and Canon [13]. More formally, our hierarchical constructions are as follows:

Definition 8. *Let $\mathcal{H}$ be a collection of sets $C_i \subseteq \mathcal{P}$. We term each C_i cluster. Then, collection $\mathcal{H}$ is said to be a telescoping hierarchy of clusters if it has the following properties:*

i. $\mathcal{P} \in \mathcal{H}$. The cluster $C_i = \mathcal{P}$ is called the global cluster of $\mathcal{H}$.

ii. For any two clusters $C_i, C_j \in \mathcal{H}$, either $C_i \cap C_j = \emptyset$, or one is a proper subset of the other.

iii. For each cluster $C_i \in \mathcal{H}$, C_i is the union of a finite number of sets in $\mathcal{H}$.

Let $\mathcal{H} = \{C_1, C_2, \ldots, C_K\}$ be a telescoping hierarchy. Then, it is easy to see that the relation $\subset$ defines a partial ordering over the sets of $\mathcal{H}$. This generates a tree with levels, where each cluster is successively partitioned into disjoint clusters.

Definition 9. *Let $\mathcal{H}$ be a telescoping hierarchy of clusters. A cluster C in $\mathcal{H}$ is said to be a leaf cluster if and only if it does not contain another cluster, i.e., for all $\widehat{C} \in \mathcal{H}$, $\widehat{C} \neq C$: $C \not\supseteq \widehat{C}$.*

We note that clusters may differ in size, and at any moment in time, leaf clusters may belong to different tiers. Further, our hierarchical model reflects the paths that messages follow:

Definition 10. *Let $\mathcal{H}$ be a telescoping hierarchy of clusters. Then, two nodes communicate over the lowest-tier cluster they share. More formally, assume that two nodes u and v belong to distinct leaf clusters in $\mathcal{H}$. Then, u and v communicate over the cluster C such that $u \in C$, $v \in C$ and for all $\widehat{C} \in \mathcal{H}$, $C \supset \widehat{C} : u \notin C$ or $v \notin C$.*

This is exactly one of the main characteristics of our constructions: a message from a node u to another node is injected to u's leaf cluster, and then it "works its way up" to the lowest cluster that is shared by them.

Summary of results

In this Chapter, we make the following contributions:

i. We provide the first formal framework that generically yet smartly augments plain DHTs to accommodate hierarchies almost for free, i.e., without renouncing to the scalability, fault-tolerance and load balancing properties of the original designs. Its distinguishing feature is that is based on the Cayley graph group-theoretic model [16], an algebraic machinery that has been extensively used to study the structural and algorithmic properties of interconnection networks (ICNs).

   We have used the Cayley graph model for three reasons:

   (a) By using the Cayley model, we ensure that the developed theory is not tied to a given DHT topology; many important DHT geometries are Cayley graphs like the *hypercube* (HyperCuP [26]), the *torus* (CAN [9]), the *cube-connected cycles* (Cycloid [27]), and the *pancake graph* (IHOP [28]);

   (b) Every Cayley graph is *vertex-symmetric*, i.e., a graph in which no node occupies a more prominent position than the others. This is fundamental for load balancing, since one can expect that all nodes operate in a uniform manner and whatever occurs at a node occurs equivalently at all the other nodes. On the search side, this means that communication is well balanced across all peers in the network.

   (c) This model allows us to abstract our topological problem to a sorting problem, which is tractable programmatically using tools like GAP [29].

ii. We show how to apply our framework to Chord. We also describe how to adapt other DHTs, including non-deterministic Chord [30], Symphony [11], Kademlia [6] and Pastry [5].

iii. We argue that our hierarchical constructions have numerous benefits, including significant reductions in the average number of routing hops and in network latency for `Get` and `Put` operations, particularly when locality in communication is high.

The rest of this Chapter is structured as follows:

- In §3.2, we provide a survey of related work.
- §3.3 gives necessary background information.

- We describe our framework in §3.4.
- In §3.5, we show how to obtain hierarchical versions of other DHTs.
- In §3.6, we validate our framework and quantify its advantages.
- We conclude this Chapter in §3.7.

3.2 Related Work

Hierarchical Designs

Many research works have proposed hierarchical designs for accommodating growth and isolating faults. As discussed in Chapter 2, these designs can be classified into two groups:

- Homogeneous designs, in which a unique global identifier space is shared by a hierarchy of self-contained clusters. Prominent examples of this class are Coral [24] and Canon [13].
- Superpeer designs, in which each cluster constitutes an independent DHT that a small set of gateway nodes (often referred to as superpeers) maintains interconnected. A representative example of this design is the work of Garcés-Erice et. al. [22].

The difference between the two architectures is that while in the first class, a node maintains the same identifier on all clusters, in the second class, each node has a distinct identifier for each cluster in which participates.

With respect to all these works, the distinguishing property of our framework is that we provide a formal technique to build a hierarchical DHT from its flat version, rather than a hierarchical model tight to a particular topology. For brevity, we only describe the most akin work to ours in each class.

Superpeer design: Garcés-Erice et. al. in [22] studied the benefits that emerge when hierarchical design is combined with peer-to-peer foundations. They examined the advantages obtained from superpeers, network proximity, and caching in this type of networks. Architecturally, this design is essentially the opposite to our hierarchical constructions from the viewpoint of load balancing, as inter-cluster queries are routed through a small number of peers, i.e., the superpeers. However, it bears some similarities with our work in the sense that they defined a new hierarchical model for DHTs.

Homogeneous design: the most similar work in spirit to our framework is Canon. The major aim of Canon [13] is to build a hierarchy of telescoping clusters (i.e., clusters of ... of clusters of nodes) from a flat design in such a way that it inherits the homogeneity of load and functionality offered by its flat counterpart, just our goal. However, it presents a major shortcoming. Canon does not provide any formal mechanism to automatically embed a hierarchy into a DHT. In contrast, we provide a generic framework based on Cayley graphs which is extensible to almost all existing topologies, and allows to exploit the vast literature on Cayley topologies to improve flat DHTs.

Cayley graphs

While the Cayley graph model is very common in ICNs literature, to date only a few proposals have attempted to study the structural properties of DHTs according to this model. Ratajczak and

Hellerstein [28] claimed that there exist two complementary aspects to bear in mind when designing a DHT: an *ideal topology* and an *emulation scheme*. It is worth noting that unlike ICNs, DHTs allow its overlay graph to evolve as nodes join/leave. Therefore, once an ideal topology has been chosen, it is necessary to emulate it when the name space is sparsely populated. For this purpose, they evaluated several emulation schemes and proposed IHOP (Internet Hashing over Pancake graphs), a DHT design based on pancake topology. Analagously, Qu and colleagues [31] pointed out that several insights can be gained by concentrating on the ideal DHT topologies. Based on the observation that the ideal topologies influence important features such as *load balancing*, *scalability*, *congestion* and *fault-tolerance*, they showed that many DHTs employ Cayley graphs as their ideal topologies, thus taking advantage of several important properties of Cayley graphs such as vertex/edge symmetry, fault-tolerance and hamiltonicity. However, the important distinguishing contribution of our work is that we show how the Cayley graph model can improve standard DHTs in the sense that communication becomes more efficient. Further, this is done without sacrificing load-balancing and other basic properties of DHTs. In fact, in all the above proposals only the Cayley graph representation of the current DHTs is given, thus exposing the theoretic benefits of the Cayley model but without showing how to work with it.

Another interesting work has been recently proposed by Mihai Lupu et. al. [32]. Their main contribution was to identify several graph-theoretic measures that ease the election of a DHT given a set of prerequisites such as query completeness, fault-tolerance, load balancing and scalability. To identify the necessary conditions for a DHT to be optimal, they looked at their static topologies using the corresponding Cayley graph representations. Although they worked with Cayley graphs in their analysis, their work is complementary to ours. Their aim was to come up with some interesting measures to assess the possibilities of a DHT rather than improving it.

There are some recent works that use the Cayley graph model for the definition of their topologies. For instance, W. Xiao and B. Parhami proposed a model of deterministic small-world graph in [33]. They used an algebraic method to construct Cayley graphs which display small-world features. More interesting, however, is ComNET [34], a new Cayley DHT topology that supports logical grouping, and defines a set of P2P protocols which are suitable for providing file browsing services. These examples show, in conjunction with this Chapter, that with the use of the Cayley model, one can solve distributed problems easily; in our particular case, the identification of the better manner to embed a hierarchy in a representative DHT.

3.3 Background

In this section, some basic definitions are given, and some notation is introduced.

3.3.1 Graph-theoretic and Group-theoretic Notions

We provide a set of basic definitions from graph and group theory to facilitate discussions in the sections that follow.

Definition 11. *A graph $G = (V, E)$ is defined by a set V of vertices and a set E of edges, where E is a subset of elements (u, v) of $V \times V$.*

In our formal representation of a DHT, each node corresponds to a *vertex* and each link to an *edge*. The terms "graph" and "network" will be considered synonymous here. Nodes connected by an edge in E are said to be *adjacent* to each other. Nodes adjacent to $v \in V$ will be also referred as the *neighbors* of v.

Now, some basic algebra definitions:

Definition 12. *A group* $\Gamma = (V, \circ)$ *is a set of elements* V *together with a binary operation* $\circ : V \times V \longrightarrow V$ *with the following properties:*

i. closure: $\forall u, v \in V$*:* $u \circ v \in V$.

ii. associativity: $\forall u, v, w \in V$*:* $u \circ (v \circ w) = (u \circ v) \circ w$.

iii. identity: $\exists 1 \in V$ *such that (s.t.)* $\forall u \in V$*:* $u \circ 1 = 1 \circ u = u$.

iv. inverse: $\forall u \in V, \exists v \in V$*:* $u \circ v = v \circ u = 1$.

By an abuse of notation, we will often write Γ as the set of elements of the group. We will refer to the binary operation of a generic group as multiplication.

Definition 13. *A subset* $S \subset \Gamma$ *is called a generating set of group* Γ *if every element* $u \in \Gamma$ *can be written as the multiplication of a finite set of elements from* S.

Cayley graphs [16] are based on groups and constitute a large class of *vertex-symmetric* networks, i.e., networks which look the same from any vertex (node). More formally,

Definition 14. *Given a set* $S = \{s_1, s_2, ..., s_d\}$ *of generators for a group* Γ*, a Cayley graph* $\mathrm{Cay}(\Gamma, S)$ *has the vertices corresponding to the elements of* Γ *and the edges corresponding to the action of the generators: if* $u, v \in \Gamma$*, the edge* (u, v) *exists in* $\mathrm{Cay}(\Gamma, S)$ *if and only if there is a generator* $s \in S$ *such that* $v = s \circ u$.

A common assumption is that the identity element of Γ does not belong to S (in order to avoid *self-loops*) and that S is closed under inverses (i.e., $s \in S \Rightarrow s^{-1} \in S$).

We will also make use of the following two definitions.

Definition 15. *Given a set of elements* ξ *within some group* Γ*, the group generated by* ξ *is defined as the smallest subgroup of* Γ *that contains all the elements in that set.*

Throughout this Chapter, we will denote the group generated by ξ as $\langle\xi\rangle$. Moreover, we will denote by $o(\langle\xi\rangle)$ the *order* of subgroup $\langle\xi\rangle$.

Finally, we briefly define what a *graph isomorphism* is.

Definition 16. *A graph isomorphism is a bijection between the vertex sets of the two graphs that preserves the adjacencies. We say that two graphs,* G_1 *and* G_2*, are isomorphic (written* $G_1 \cong G_2$*) if there exits an isomorphism between them.*

3.3.2 Chord

Although Chord [4] has been extensively described in Chapter 2, we include here a brief summary to refresh it.

To support searches in a highly dynamic environment, Chord maps keys to nodes by means of *consistent hashing* [35], which has some desirable properties. Consistent hashing assigns an n-bit identifier to both nodes and objects using a collision-resistant hash function such as SHA-1. Then, objects are mapped to nodes as follows. Identifiers are ordered on an identifier circle modulo 2^n, labeled from 0 to $2^n - 1$. An object with identifier k (also know as *key*) is assigned to the first node, called the successor of this key and denoted by $succ(k)$, whose identifier follows (or is equal to) k in the identifier space (i.e., the first node going clockwise from k). If a node u is given as input, $succ(u)$ returns the immediate successor of this node in the Chord circle.

To accelerate searches, Chord maintains logarithmic routing information. Each node u maintains a routing table with up to n entries called *finger table*. The i^{th} entry in the table contains the id of the first node v that succeeds u by at least 2^i on the identifier ring, where $0 \leq i < n$. In addition, node u maintains a list, called *successor list*, of pointers to the $\mathcal{O}(\log N)$ immediate successors of u on the Chord circle. Its main purpose is to guarantee a reliable operation even if half of the nodes fail. Such a structural definition can be represented in form of Cayley graph as follows:

Definition 17. *Let Γ be the group $(\mathbb{Z}_{2^n}, +)$ of 2^n elements with generators $\{\pm 2^i\}_{0 \leq i < n}$. Then, the pair* $\text{Cay}\left(\Gamma, \{\pm 2^i\}_{0 \leq i < n}\right)$ *is the Cayley graph representation of Chord with diameter $n/2$ and degree $2n$.*

The *standard* Chord search protocol works as follows: Upon receiving a message for a key k, a node u forwards the query to the furthest finger whose id precedes most immediately (or is equal to) key k. By $\mathcal{O}(\log N)$ forwardings, the message reaches the *destination node*. Hence, routing is clockwise and *greedy*, never overshooting the destination.

From here on, we will refer to the clockwise distance $d_{clockwise}(u, v)$ between two nodes u and v as $d(u, v)$ for simplicity.

3.4 Cyclone Framework

In this section, we first present an overview of our framework, followed by a detailed description of its functionalities.

3.4.1 Overview

In Section 3.1, we have shown that by logically grouping computers by similarity, one can increase query efficiency. However, it is unclear how to do this so that the hierarchical version of a DHT does not lose the abilities that characterize it. To answer this question, we base our framework on the *algebraic-combinatorial* properties of Cayley graphs. In particular, the basic idea is to exploit the recursive structure of DHTs (if any) to incorporate clusters almost for *free*. In order to do this, we need to address two technical challenges.

First, we need to determine the way in which their structures recursively decompose into smaller graphs of the same family. To be specific, what we want is not to view a DHT as *dynamic* graph, but rather as a family of "static" graphs $\{G_0, G_1, ...\}$ with the important property that $\forall i > 0$, the i^{th} graph

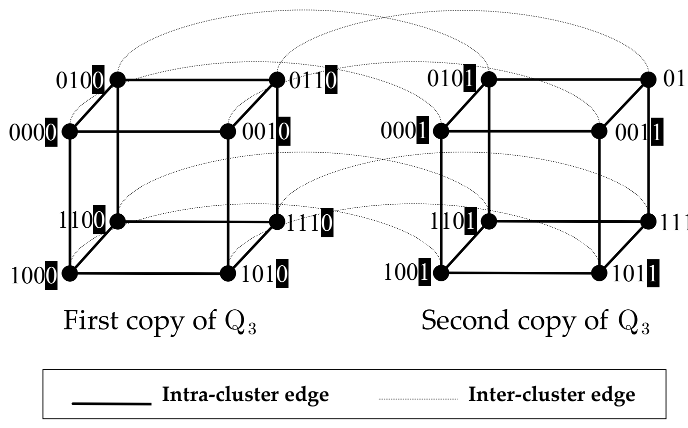


Figure 3.2: *Recursive decomposition of Q_4. Edges belonging to the 3-dimensional hypercubes are shown in bold style. Interconnection edges are drawn in thin style.*

can be constructed in a recursive manner from identical copies of G_{i-1}. For some structures such as the *hypercube*, this is already known: the hypercube of dimension n, Q_n, can be recursively constructed from 2 hypercubes of dimension $n-1$, Q_{n-1} (see Fig. 3.2). For other DHTs, however, this may not be so *obvious*. We reflect this in the recursive construction of Chord.

In other words, the first problem we deal with in this Chapter is the *decomposition problem* for DHTs. Assuming that a DHT can be recursively defined as a family of static graphs $\{G_0, G_1, G_2, ...\}$, our first challenge is to elucidate the relationship between successive graphs, G_i and G_{i-1}, of the same family. To whet the reader's appetite for the conclusions, Cyclone reduces this problem to a simple combinatorial problem that consists in ordering generators such that certain conditions are satisfied.

Second, once this task is accomplished, we need to accommodate a hierarchy without losing efficiency, scalability, fault-tolerance and load-balancing. In order to do this, the key idea behind our framework is to have each telescoping cluster "approximate" the structure of some graph of the family. In this way, we ensure that each cluster implicitly inherits the properties and complexity of the original design, but now with the benefits in query throughput brought by a hierarchical design. Given a cluster, the election of the most appropriate graph of the family mainly depends on the application built atop. For this reason, we restrict our attention on demonstrating the validity of this strategy.

This issue is tightly related to the dynamics of peers. Since each peer can freely decide to join or leave the system at any time, our second task requires to demonstrate that from the recursive definition of a DHT, a hierarchy of dynamic clusters can be incorporated without negatively affecting efficiency, scalability, fault-tolerance and load-balancing. Specifically, we show that it suffices to apply the construction and maintenance rules of the original design to meet this challenge.

In summary, Cyclone takes two steps to tackle the problem of constructing hierarchical DHTs. We describe them as follows:

- **Step 1**: *Recognizing Recursive Structure.* Given a Cayley DHT (say, $\mathrm{Cay}(\Gamma, S)$), in this step, the task of Cyclone is to elucidate (if possible) how $\mathrm{Cay}(\Gamma, S)$ can be recursively decomposed into smaller graphs of the same variety. As a result, we get a budget of subgraphs that can potentially behave as the full overlay.

- **Step 2**: *Mapping the Hierarchy.* Once the above is done, the second job is to determine how to embed the hierarchy into the DHT. To this aim, we associate a subgraph of $\mathrm{Cay}(\Gamma, S)$ to each cluster and we apply the link construction and maintenance rules of the original DHT. As a result, we obtain a new DHT, with each cluster approximating a small subgraph of the same family.

3.4.2 Determining Recursive Structures: Cayley DHTs

The first problem we consider in this paper is how to elucidate the *recursive structure* of a DHT. In this section, we propose a solution to address this challenge in terms of the Cayley graph model. The idea is to consider a DHT topology as a family of Cayley graphs $F = \{G_0, G_1, G_2, ...\}$ and use this characterization to identify the recursive relationship between every pair of successive graphs of the family. Specifically, determining this relationship requires answering the following two questions:

- First, *how many copies of G_{i-1} are needed to construct G_i?* A natural way of doing so is to provide a *non-null* function ϕ that specifies for all $i > 0$, the number of copies of G_{i-1} needed to construct G_i. For example, the corresponding ϕ for the family of binary hypercubes is the constant function 2; and

- Second, *how these copies must be interconnected to construct G_i?* For that matter, it is necessary to describe two aspects. Assuming that $G_{i-1}^j = \left(V_{i-1}^j, E_{i-1}^j\right)$, $1 \leq j \leq \phi(i)$, denotes the j^{th} copy of G_{i-1}, we need to specify the following: One the one hand, how to relabel the vertices of each copy such that $\bigcup_{1 \leq j \leq \phi(i)} V_{i-1}^j = V_i$. On the other hand, how to interconnect the relabeled copies such that $\left[\bigcup_{1 \leq j \leq \phi(i)} E_{i-1}^j\right] \bigcup X = E_i$, where X is the set of additional edges used to construct G_i.

For answering the above questions, we use the concept of *quasiminimality* introduced in [36]. The importance of quasiminimality is that allows determining the recursive structure of Cayley graph through the simple rearrangement of the elements in its generating set. This can be done *programmatically*, and for many cases, it suffices to examine a small graph of the family to infer its decomposition procedure, thus requiring little effort in terms of CPU cycles. In general, for families of graphs where ϕ is *constant* or *linear*, the same set of rules can be applied to decompose any graph of the family. In the recursive decomposition of Chord, we will provide enough evidence of this. In what follows, we make use of the following definition of quasiminimality:

Definition 18. *A generating set S is a quasiminimal symmetric generating set if there exists a numbering of the elements of S: s_1, s_2, ..., s_n such that*

i. if the order of s_i is greater than 2, then s_i^{-1} is either s_{i-1} or s_{i+1}.

ii. if the order of s_i is 2, then $\langle\{s_1, s_2, ..., s_{i-1}\}\rangle$ is a proper subgroup of $\langle\{s_1, s_2, ..., s_i\}\rangle$.

iii. if the order of s_i is greater than 2 and $s_i^{-1} = s_{i-1}$, then $\langle\{s_1, s_2, ..., s_{i-2}\}\rangle$ is a proper subgroup of $\langle\{s_1, s_2, ..., s_i\}\rangle$.

To complete the picture, we now explain why *quasiminimality* can solve the decomposition problem for structured overlays. For simplicity in discussion, let us simplify Definition 18 as follows. Let $\Omega(s)$ be the subset $\{s, s^{-1}\}$ of S. Then, under this notation, we say that a generating set S is *quasiminimal* if there exists an ordering of the Ω-sets, $\Omega(s_1)$, $\Omega(s_2)$, ..., $\Omega(s_j)$, such that $\Omega(s_i)$ is outside the subgroup generated by the first $(i-1)$ Ω-sets, $2 \leq i \leq j$.

Now consider a Cayley graph with its generating set S ordered in *quasiminimal order*: $\Omega(s_1)$, $\Omega(s_2)$, ..., $\Omega(s_j)$. Let ξ_j denote the set $\bigcup_{1\leq i\leq j} \Omega(s_i)$. Denote by G_j the Cayley graph $\mathrm{Cay}(\langle\xi_j\rangle, \xi_j)$. Then, it can be easily seen that G_{j+1} can be built from $\phi(j+1) = o(\langle\xi_{j+1}\rangle)/o(\langle\xi_j\rangle)$ vertex-disjoint copies of G_j along with the edges interconnecting them: *there is one copy of G_j for each left coset of $\langle\xi_j\rangle$*. In fact, G_{j+1} can be viewed as collection of copies of G_j with the edges between them corresponding to the action of generator s_{j+1} and its inverse on the copies of G_j. Certainly, this is what makes quasiminimality extremely useful. To be specific, if one examines the quasiminimal sequence in reverse order, i.e., $\Omega(s_j)$, $\Omega(s_{j-1})$, ..., $\Omega(s_1)$, decomposing a Cayley graph mainly consists in increasingly removing the Ω-sets from S in this order. Note that each removal implies eliminating all the edges with a specific label from the graph. Remember that an edge (u, v) is labeled s if and only if there exists a generator $s \in S$ such that $v = s \circ u$.

An Example: Chord

To illustrate the above theory, next we show how quasiminimality allowed us to elucidate the recursive structure of Chord. Throughout this thesis, we will denote the Chord graph of order 2^n by CH_n. Before understanding the recursive structure of CH_n (See Theorem 21), we introduce some auxiliary formalities.

Theorem 19. *Let CH_n and $CH_n[V_k]$ be the Chord graph of order 2^n and the subgraph of CH_n induced by the set $V_k = \{v | v \equiv k \ (mod\ 2)\}$, $0 \leq k < 2$, respectively. Then, the graph CH_{n-1} is isomorphic to $CH_n[V_k]$, $n > 0$.*

Proof. See appendix □

By considering the following remark, we will prove that Chord has a recursive structure in Theorem 21.

Remark 20. *All subgroups of a cyclic group are cyclic. If $\Gamma = \langle g\rangle$ is a cyclic group of order n, then for each divisor m of n there exists exactly one subgroup of Γ of order m; it can be generated by $g^{n/m}$ (Lagrange's theorem).*

Now we are in position to show that CH_n can be recursively defined. We use the following theorem.

Theorem 21. *Let V_i be the subset of vertices of CH_n such that $V_i = \{v | v \equiv i \ (mod\ 2)\}$, $n \geq 1$ and i, $0 \leq i < 2$. Let $CH_n[V_i]$ denote the graph induced by V_i. Then, CH_n has a recursive structure that contains two vertex-disjoint copies of the graph CH_{n-1}, i.e., the graphs $CH_n[V_0]$ and $CH_n[V_1]$. The unused edges form a Hamiltonian cycle.*

Proof. We verify that CH_n has a recursive structure. For reaching this aim, we have to show that for $n > 0$, the set $\xi = \{\pm 2^{n-1}, \pm 2^{n-2}, ..., \pm 1\}$ is a quasiminimal symmetric generating set and then, formally state the recursive definition of CH_n by taking ξ in reverse order. Throughout the proof, we will denote the additive cyclic group of 2^n elements by $\mathbb{Z}_{2^n}$. For the first part, we will make use of the following lemma (recall that the vertex set of graph CH_n is $V = \{0, 1, ..., 2^n - 1\}$):

Lemma 22. *For $k > 0$, let $\xi = \left\{\pm 2^{k-1}, \pm 2^{k-2}, ..., \pm 1\right\}$ denote the generating set for CH_k. Then, ξ with the given order is a quasiminimal symmetric generating set of $\mathbb{Z}_{2^k}$.*

Proof. The strategy of the proof is as follows. Since $\left\langle +2^l \right\rangle = \left\langle -2^l \right\rangle$ is true for all $0 \leq l < k$, we must show that definition 18.3 holds for each pair $\left\{+2^l, -2^l\right\} \in \xi$, $1 \leq l < k$. Notice that $\left\{+2^{k-1}, -2^{k-1}\right\}$ is a single element, since $\left\langle +2^{k-1} \right\rangle$ has order 2. Then we must prove:

1. *For each $\left\{+2^l, -2^l\right\} \in \xi$, $1 \leq l < k-1$, both $o(+2^l)$ and $o(-2^l)$ is greater than 2; and*

2. *For all $2 \leq l \leq k$, $\left\langle \xi_l = \left\{\pm 2^{k-1}, \pm 2^{k-2}, ..., \pm 2^{k-l}\right\}\right\rangle$ is a proper supergroup of $\left\langle \xi_{l-1} = \left\{\pm 2^{k-1}, \pm 2^{k-2}, ..., \pm 2^{k-l+1}\right\}\right\rangle$.*

The above two claims are shown as follows:

1). From Remark 20, it follows that for each m, $0 \leq m \leq k$, there exists a unique subgroup of group $\mathbb{Z}_{2^k}$ of order 2^m. Let K be the subgroup of order 2^m. With the exclusion of the trivial subgroup, we have by Remark 20 that $K = \left\langle +2^l \right\rangle$, where $l = k - m$. Since $\left\langle +2^l \right\rangle = \left\langle -2^l \right\rangle$, we can claim $o(+2^l) = o(-2^l) = 2^{k-l}$ for each l, $1 \leq l < k-1$, thereby concluding **1)**.

2). *By contradiction.* Without loss of generality, we assume that $2^{k-l} \in \left\{\pm 2^{k-1}, \pm 2^{k-2}, ..., \pm 2^{k-l+1}\right\}$. Then, we come to a contradiction. We know from elementary group theory that

$$\langle \xi_{l-1} \rangle = \left\{v | v \equiv x_1 2^{k-1} + x_2 2^{k-2} + ... + x_{l-1} 2^{k-l+1} (mod\ 2^k), x_1, x_2, ..., x_{l-1} \in \mathbb{Z}\right\}$$

Thus, we have

$$\begin{aligned} x_1 2^{k-1} + x_2 2^{k-2} + ... + x_{l-1} 2^{k-l+1} - 2^{k-l} &\equiv 0\ (mod\ 2^k), \\ 2^{k-l}\left(x_1 2^{l-1} + x_2 2^{l-2} + ... + x_{l-1} 2 - 1\right) &\equiv 0\ (mod\ 2^k), \end{aligned}$$

Since 2^{k-l} is not congruent to $0\ (mod\ 2^k)$, then $x_1 2^{l-1} + x_2 2^{l-2} + ... + x_{l-1} 2 \equiv 1\ (mod\ 2^k)$. Because each $x_i 2^{l-i} \equiv 0\ (mod\ 2)$, we come to a contradiction and the lemma follows. □

Now let us denote by $CH_n[V_i]$ the subgraph of CH_n induced by the set $V_i = \{v | v \equiv i\ (mod\ 2)\}$, $0 \leq i < 2$. In view of the above, it is natural to ask: *what is the relationship between ξ, ordered as in Lemma 22, and the subgraph $CH_n[V_i]$?* The answer is the following: $CH_n[V_i]$ has as generating set the subset $\xi \setminus \{+1, -1\}$.

Lemma 23. *Let $CH_n[V_i]$ be the subgraph of CH_n induced by $V_i = \{v | v \equiv i\ (mod\ 2)\}$, $0 \leq i < 2$. Then, the generating set for $CH_n[V_i]$ is $\left\{\pm 2^k\right\}_{1 \leq k < n}$.*

Proof. Suppose that $(v, v + 2^k)$ is an edge of $CH_n[V_i]$. Since both v and $v + 2^k$ are in V_i, then we have

$$\begin{aligned} v &\equiv i\ (mod\ 2), \\ v + 2^k &\equiv i\ (mod\ 2), \end{aligned}$$

this implies that $2^k \equiv 0\ (mod\ 2)$. This occurs when $k > 0$, so the lemma is proved. □

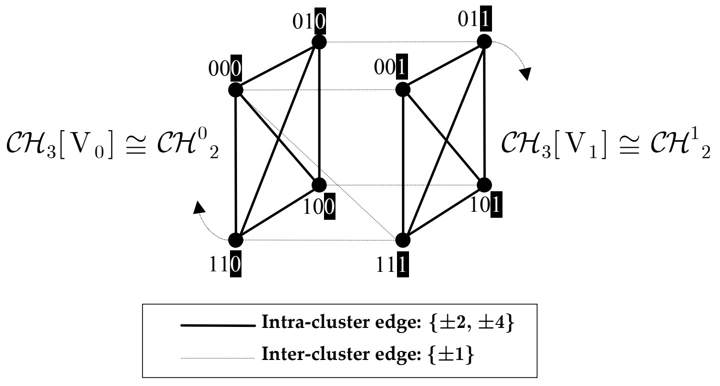

010
011
000
001
$\mathcal{CH}_3[V_0] \cong \mathcal{CH}^0{}_2$
$\mathcal{CH}_3[V_1] \cong \mathcal{CH}^1{}_2$
100
101
110
111
Intra-cluster edge: {±2, ±4}
Inter-cluster edge: {±1}

Figure 3.3: *Recursive definition of* $CH_3 = Cay(\mathbb{Z}_8, \{\pm 4, \pm 2, \pm 1\})$. *All vertices are represented in base 2. Intra-cluster edges of the two copies of* CH_2 *are shown in* bold *style while inter-cluster edges are drawn in* thin *style. All nodes belonging to the same copy share the last bit of their* $nodeIds$.

Our development to this point allow us to infer, by the preceding lemma, that CH_n contains two *vertex-disjoint* and *edge-disjoint* subgraphs, called $CH_n[V_0]$ and $CH_n[V_1]$, obtained by removing the edges labeled 1 from CH_n. Now, it only remains to show that the subgraphs $CH_n[V_i]$, $0 \leq i < 2$, are copies of CH_{n-1} to verify that the recursive decomposition of CH_n is into $\phi(n) = 2$ copies of CH_{n-1}.

Consider now the mapping from the vertex set of CH_{n-1} to the vertex set of $CH_n[V_i]$ defined by $u \rightarrow i + 2u$. By Theorem 19, this mapping defines a graph isomorphism between CH_{n-1} and $CH_n[V_i]$, and hence, we can find 2 copies of graph CH_{n-1} into graph CH_n. Note that the automorphism of group $\mathbb{Z}_{2^n}$ defined by $s \rightarrow 2s$ maps the generating set of CH_{n-1} to the generating set of $CH_n[V_i]$. Hence, the edges labeled 2 of $CH_n[V_i]$ are indeed the edges labeled 1 of CH_{n-1} and ergo, the next ones to be removed.

As said before, the edges that are between a vertex in $CH_n[V_0]$ and a vertex in $CH_n[V_1]$ are obtained by adding 1 $(mod\ 2^n)$ to their vertices. As a result, they describe together a Hamiltonian cycle. □

Summary of results

We have shown how CH_n, $n \geq 1$, can be *recursively* decomposed into $\phi(n) = 2$ copies of CH_{n-1}. However, we have not described explicitly the two aspects that define the recursive structure of Chord yet. These are:

- On the one hand, we have the relabeling of the vertices in each copy; and
- On the other hand, the set of edges, namely X, which interconnect the copies.

For the former aspect, assuming that $CH^j_{n-1} = (V^j_{n-1}, E^j_{n-1})$, $1 \leq j \leq \phi(n)$, denotes the j^{th} copy of CH_{n-1}, we must relabel each vertex $u \in V^j_{n-1}$ by $u \rightarrow j+2u$. For the latter aspect, it is easy to see that $X = \{(u, v) | v \equiv u + 1\ (mod\ 2^n)\}$. To clarify these aspects, the recursive construction of CH_3 is illustrated in Fig. 3.3.

It is important to note that finding the quasiminimal generating set for Chord, and presumably other topologies, is not time-consuming. For the example shown in Fig. 3.3, it suffices to discover that $\{0, 4\} \subset \{0, 2, 4, 6\}$, and then, $\{0, 2, 4, 6\} \subset \{0, 1, 2, 3, 4, 5, 6, 7\}$, which can be done by permuting the 3 generators of CH_3. This is the major strength of quasiminimality: the permutation of 3 generators has elucidated the internal structure of Chord.

Traffic Concentration: Node and Link Congestion for Chord

One crucial requirement of P2P systems is that communication load across all nodes (resp., links) is well balanced. In this section, we examine *node-* and *link-congestion* in Chord. Our primary aim behind this is to give a quantitative idea about how much traffic concentrates in a node (resp., a link) in Chord, to show later that our hierarchical version of Chord balances load as effectively as the original design.

So far, we have treated Chord as an undirected graph for ease of explanation (for the figures, in particular), but we have to keep in mind that Chord is in reality a Cayley digraph with binary operation $+(mod\ 2^n)$ and generating set $\{2^i\}_{0\leq i<n}$. For this reason, we will measure the load on arcs instead of the load on edges. We will make use of the following standard notation:

Definition 24. *A routing R in a connected (di)graph G of order N is the set of $N(N-1)$ paths specified for every (ordered) pair of vertices of G.*

To measure the traffic concentration deterministically, Chung et al. in [37] and Heydemann et al. in [38] introduced the concept of the *vertex-forwarding index* and Manoussakis and Tuza in [39] the notion of *arc-forwarding index* of a routing R. We made use of the following definitions:

Definition 25. *For a routing R, the load, $\xi(G, R, v)$, of a vertex v is defined as the number of paths in R that pass through v (where v is not an end vertex).*

Likewise, they defined the load of an arc as follows:

Definition 26. *For a routing R, the load, $\pi(G, R, a)$, of an arc a is defined as the number of paths of R going through a.*

Then, the parameters

$$\begin{aligned}\xi(G,R) &= \max_{v\in V(G)} \xi(G,R,v)\\ \pi(G,R) &= \max_{a\in E(G)} \pi(G,R,a)\end{aligned}$$

were defined as the *vertex-forwarding index* and the *arc-forwarding index* specified by routing R over the graph G, respectively. Then,

Definition 27. *The vertex-forwarding index $\xi(G)$ of the graph G is the minimum of $\xi(G, R)$ over all routings R in G.*

Analogously,

Definition 28. *The arc-forwarding index $\pi(G)$ of the graph G is defined as the minimum of $\pi(G, R)$ over all routings R in G : $\pi(G) = \min_R \pi(G, R)$.*

Denote by GR_n the standard Chord GREEDY routing protocol. Now, we give the forwarding indices of CH_n with respect to GR_n. We note that we implicitly assume here that every node in the identifier space exists and is alive. This assumption is acceptable, as there exists no way to achieve a lower bound on the *maximum* number of paths going through any node (resp., any link) than $\xi(G, R)$ (resp., $\pi(G, R)$) when the identifier space is not *fully populated*. This happens because there are nodes in Chord with an *in-degree* of $\Theta(\log^2 N)$ *with high probability* [*]. So they can only increase the number of times a node (respectively, a link) is visited by GR_n. This occurs whenever a node u has not other nodes in the range $[u - 2^n(\log N)/N, u]$, because then u attracts an *average* number of $\log N$ other nodes for each distance 2^i, resulting in $\Theta(\log^2 N)$ incoming links *with high probability*.

Theorem 29. *The vertex-forwarding index of CH_n with respect GR_n is $\xi(CH_n, GR_n) = 2^{n-1}(n-2)+1$.*

Proof. Observe first that $\delta(u, v) - 1$ is the minimum number of vertices through which $GR_n(u, v)$ can possibly pass, where $\delta(u, v)$ is the length (in number of hops) of the shortest, *clockwise* path between u and v. Summing over all $v \neq u$ and then over u, we can obtain a lower bound on the total number of instances in which paths pass through vertices. Since the maximum number of paths passing through any vertex in CH_n cannot be less than the average number, then we have that $\xi(CH_n, GR_n) \geq \frac{1}{V} \sum_u \sum_{u \neq v} (\delta(u, v) - 1)$. Moreover, GR_n is easily seen to be *symmetric* in the sense that it imposes the same forwarding index on each vertex. Thus, $\xi(CH_n, GR_n) = \frac{1}{V} \sum_u \sum_{u \neq v} (\delta(u, v) - 1)$, which by *vertex-symmetry* reduces to $\frac{1}{V} |V| \sum_{v \neq 0} (\delta(0, v) - 1)$.

Let $s(0, CH_n) = \sum_{v \in V(CH_n)} \delta(0, v)$ be the sum of distances from vertex 0 to all other vertices in CH_n. Based on the analysis of Loguinov et. al. [40], it can be easily seen that any shortest path of length l is formed by drawing l unique generators from set $\{+2^0, +2^1, ..., +2^{n-1}\}$. The *uniqueness* is easy to see since any two jumps of size 2^i can be replaced with a single (more optimal) jump of size 2^{i+1}. Consequently, each node can reach $\binom{n}{l}$ other nodes at length l. Hence, $s(0, CH_n) = \sum_{1 \leq l \leq n} l \binom{n}{l} = n2^{n-1}$. Then, $\frac{1}{V} |V| \sum_{v \neq 0} (\delta(0, v) - 1) = s(0, CH_n) - (2^n - 1) = 2^{n-1}(n-2) + 1$ and the theorem follows. □

In addition to vertex transitivity, Cayley graphs may have another important property called *arc transitivity*. A graph G is said to be arc-transitive if for any given pair of arcs (u, v), (a, b), there exists an automorphism σ of G such that $(\sigma(u), \sigma(v)) = (a, b)$. The benefit of arc transitivity is that in an arc-transitive graph, communication load is uniformly distributed on all links. Although Chord is not arc-transitive, its routing protocol subtly compensates the lack of arc transitivity. We notice that if one considers (u, v) and (a, b) without direction, then G is said to be edge transitive.

Lemma 30. *The Chord graph CH_n of order 2^n is not arc-transitive for $n \geq 2$.*

Proof. See appendix. □

By similar reasoning to that in Lemma 30, one can easily demonstrate that Chord is not edge-transitive.

[*]An event ζ occurs with high probability (w.h.p.) if $Pr\{\zeta\} \geq 1 - \frac{1}{N^k}$

Lemma 31. *The Chord graph CH_n of order 2^n is not edge-transitive for $n \geq 3$.*

Proof. Say first that CH_n is edge-transitive for any $n \leq 2$, as in that case CH_n is isomorphic to the complete graph of 2^n vertices. This concludes the first part. For $n \geq 3$, it suffices to apply the same reasoning of Lemma 30 to show that an edge labeled 1 does not participate in the same number of cycles of an edge labeled $n-2$. Hence, we have at least two types of edges- So, CH_n cannot be edge-transitive. □

The following theorem derives the *arc-forwarding index* of Chord and our work appears to be the first to claim it. The important insight here is that $\pi(CH_n) = \pi(CH_n, GR_n)$, which indicates that GR_n is *arc-uniform*, i.e., the load on all arcs is the same. The proof strongly depends on the following lemma [39].

Lemma 32. *If $G = (V, E)$ is a connected graph of order n, then*

1. *$\frac{1}{|E|}\sum_{(u,v)\in V\times V} d(u,v) \leq \pi(G) \leq \pi_m(G)$, where $\pi_m(G)$ denotes the minimum taken over all the routings of shortest paths in G.*

2. *The equalities hold if and only if there exists a shortest path routing in G for which the load of all arcs is the same.*

Theorem 33. *The arc-forwarding index of CH_n is $\pi(CH_n) = 2^{n-1}$.*

Proof. The proof is by induction on n. Since $\pi(CH_1) = 1 = 2^{1-1}$, the theorem is true for $n = 1$. Assume that the theorem is true for every m, $1 \leq m < n$.

By Theorem 21, it follows that CH_n can be recursively constructed from two copies of CH_{n-1}, which are the subgraphs, $CH_n[V_i]$, of CH_n induced by $V_i = \{v|v \equiv i\ (mod\ 2)\}$, $0 \leq i < 2$. Let P_n be the set of paths between the set V_0 and V_1 in an arbitrary routing R_n of CH_n. Then, $|P_n| = 2 \times 2^{n-1} \times 2^{n-1}$. Since there are 2^n arcs (with label 1) between $CH_n[V_0]$ and $CH_n[V_1]$ and every path in P_n uses at least one arc, $\pi(CH_n) \geq |P_n| / 2^n = 2^{n-1}$.

On the other hand, since $\pi(CH_n) \leq \pi(CH_n, GR_n)$, we only need to show that $\pi(CH_n, GR_n) \leq 2^{n-1}$. Now let us find the load of an arc a which is labeled 1. Let $a = (x, x+1\ (mod\ 2^n))$ with $x \in V_i$. Then, it is easy to see that the paths going through a are the paths ending at vertex $x+1\ (mod\ 2^n)$, i.e., the paths $GR_n(u, x+1\ (mod\ 2^n))$ for any $u \in V_i$. Because each path crosses a only once, we have $\pi(CH_n, GR_n, a) = 2^{n-1}$.

Let us calculate the load of an arc a which is not labeled 1. Assume that the ends of a are in V_i. Clearly, all the paths going through a begin in V_i: if the path $GR_n(u,v)$ passes through a, then $u \in V_i$. Now, assume that $v \in V_j$. When $i = j$, the path $GR_n(u,v)$ is simply the path $GR_{n-1}(u,v)$ in copy i. When $i \neq j$, the path $GR_n(u,v)$ can be split into two pieces: the path $GR_{n-1}(u,u')$ in copy i of graph CH_{n-1} followed by the arc (u', v) of label 1. These are all the paths going through arc a. Then, by induction hypothesis, we have

$$\pi(CH_n, GR_n, a) \leq 2\pi(CH_{n-1}, GR_{n-1}) = 2 \times 2^{n-2} = 2^{n-1}.$$

Thus, we have that $\pi(CH_n, GR_n) \leq 2^{n-1}$. Based on the above discussion, the theorem follows. □

3.4.3 Mapping a Hierarchy

So far, we have shown how to elucidate the recursive structure (if any) of a DHT in terms of its Cayley graph. Therefore, what is lacking is to show how to construct a hierarchical version of a DHT based on the *formal definition* of its recursive structure. In order to preserve the uniformity in functionality and load, our intuition is to have each cluster *approximate* a static graph from the family of the targeted DHT. We say "approximate" because the nodes in a cluster do not have to necessarily occupy all positions (identifiers) in the assigned (sub)graph. We remark here that due to the dynamics of peer-to-peer networks, the structure of each subgraph is *time-dependent*. Each node can *freely* decide to join or leave the system at any time, but what is evident is that its decision irremediably alters the structure of the underlying graph. We are aware of this fact and show why sparsity on clusters does not attenuate the validity of our proposal.

Construction Strategy

As we discussed earlier, DHTs use *consistent hashing* [35] to spread keys, and also nodes, evenly over the identifier space. Such a uniform distribution is vital in uniform DHT topologies.

As defined by Xu *et. al.* [41], a graph over N nodes arranged in a circle is said to be uniform if the set of off-sets for out-going links is identical for all nodes. This has important consequence: for any uniform graph with $\mathcal{O}(\log N)$ links per node and diameter $\Omega(\log N)$, GREEDY *routing is congestion-free*. Notice that uniformity is only possible if nodes are uniformly distributed in the identifier space, a task that consistent hashing carries out well, albeit not perfectly.

For Cayley DHTs, note that uniformity follows directly from the definition of Cayley graph. Let $Cay(\Gamma, S)$ be the Cayley graph defined by the generating set S on group Γ. Then, we know that two vertices u, $v \in \Gamma$ are adjacent in $Cay(\Gamma, S)$ if and only if $u^{-1} \circ v \in S$. Therefore, offsets for out-going links are identical in all nodes and hence, Cayley DHTs are uniform.

Uniformity implies that the greedy paths starting from distinct nodes but traveling the same distance pass through disjoint sets of nodes. This obviously balances routing load over all nodes in the network. Nearly all DHT geometries are uniform (e.g. [4–6, 10]). So, one should expect that our hierarchical constructions inherit uniformity, too. To preserve this property, our construction strategy can be summarized as follows:

i. When joining the system, the key idea is that the joining node acquires a free position in the subgraph emulating its leaf cluster. In our hierarchical model, note that the election of this position automatically determines the set of clusters a node needs to join (as clusters follow the recursive decomposition of the original topology).

ii. The second step is that the joining node applies the linking rules of the original design, but first applied to the nodes lying in its leaf cluster; second to the nodes belonging to in its next higher-tier cluster and so on, until reaching the global cluster.

More formally, let τ be the denote the number of tiers in a telescoping hierarchy $\mathcal{H}$. Each peer u is contained in an ascendant sequence of telescoping clusters in $\mathcal{H}$; hence, u maintains the same $nodeId$ in all clusters. If we denote these clusters by $C_1(u) \subset C_2(u) \ldots \subset C_\tau(u)$, where $C_1(u)$ is the leaf cluster of u and $C_\tau(u)$ is the global cluster, then

i. At tier 1, node u joins its leaf cluster $C_1(u)$, where it is assigned an identifier from $\mathcal{I}$. $C_1(u)$ emulates a graph G_1 of the same family of the original DHT; hence, node u establishes the links dictated by this graph but to the live nodes in $C_1(u)$. In order to do this, it applies the linking rules of the original design to the nodes in $C_1(u)$.

ii. From the leaf cluster to the global cluster, u successively joins each cluster $C_i(u)$, $1 < i \leq \tau$. Within each cluster, it creates the additional links specified by graph G_i, but retaining all the links from cluster $C_{i-1}(u)$. More specifically, node u establishes the links corresponding to the generators of G_i that are not present in the generating set of G_{i-1}, the graph associated with cluster $C_{i-1}(u)$.

The result is a hierarchy of clusters, where the sibling clusters in the same subtree operate as a unique DHT in the next higher tier. From the perspective of the system, constructing a hierarchical DHT can be viewed as the process of aggregating sibling clusters at each tier, to form increasingly larger clusters of the same type.

This strategy is advantageous for two reasons:

1. Nodes can route queries using keys as usual, as the mapping $h : \mathcal{O} \longrightarrow \mathcal{I}$ is complete within every cluster. Since each object is always under the responsibility of at least one peer in each cluster, nodes can store and search objects at any tier with operations `Put(key, value, level)` and `Get(key, level)`. This property is very desirable, since it avoids bothering the non-member nodes of a particular cluster by the queries originating in it.

2. The mapping $h : \mathcal{P} \longrightarrow \mathcal{I}$ can be extended to clusters. Thus each cluster can inherit the uniform distribution of peers over the identifier space, which implicitly favors communication load-balancing.

We now introduce the Cyclone construction of Chord: Whirl. It is worth mentioning that the election of the appropriate subgraph for each cluster is responsibility of the application built atop. Therefore, this issue is not longer considered in this Chapter. In general terms, what we describe below is the construction of Whirl from its recursive definition, with particular emphasis on the relationship between the identity of a node and its neighborhood. Recall that the label of a node determines the sequence of clusters where it is contained.

Whirl: The hierarchical version of Chord

So far, we have shown that Chord is a hierarchical Cayley graph. In this section, we provide the hierarchical version of Chord by exploiting its recursive structure. Recall that the static Chord version can be constructed by the successive application of the generators $\{\pm 2^{n-1}, \pm 2^{n-2}, ..., \pm 1\}$ starting from any element of group $\mathbb{Z}_{2^n}$.

Now, we need to show how to embed a hierarchy $\mathcal{H}$ in Chord. As discussed above, we need to provide the mapping between the sequences of clusters in hierarchy $\mathcal{H}$ to the sequence of graphs $\{G_0, G_1, G_2, \ldots\}$ of the Chord family, so that the graph at the i^{th} tier can be built from the graphs at the $(i-1)^{th}$ tier.

For a peer u, let $\sigma(u, s)$ be the s rightmost bits of its $nodeId$. The mapping for Chord consists in logically partitioning $\sigma(u, s)$ into τ substrings increasingly shorter, where τ denotes the height of the hierarchy. To better understand this, consider the following:

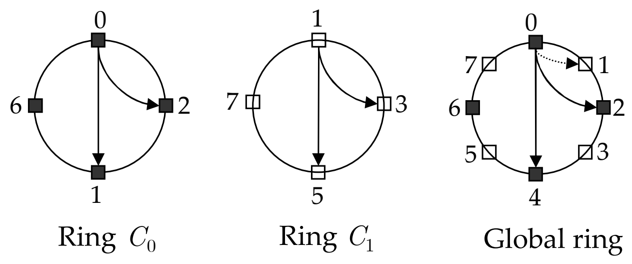


Figure 3.4: *An example of a Whirl construction.*

Represent each vertex of CH_n by its n-bit binary label $(u_1 u_2 ... u_n)$, where $u_i \in \{0, 1\}$. Then, it is easy to see that the set of vertices with u_n fixed, i.e., $CH_n[V_{u_n}]$ (see Fig. 3.3), is one of the $\phi(n) = 2$ copies of graph CH_{n-1} into which CH_n decomposes. More generally, CH_n can constructed from $\phi(m) = 2^{n-m}$ copies of subgraph CH_m, for $0 \leq m \leq n$. An immediate consequence of this is that all nodes assigned to a copy of subgraph CH_m share the $n - m$ rightmost bits of their $nodeIds$.

At a given tier, this means that the $nodeId$ of node u is logically split into two parts: a `PREFIX` of m bits, and a `SUFFIX` of $n-m$ bits, $\sigma(u, n-m)$, where m refers to the index of the Chord graph, CH_m, the cluster of this tier is emulating.

The role of the two parts is complementary. While the `PREFIX` provides a node with a unique intra-cluster identity, the `SUFFIX` identifies its cluster. In other words, the `SUFFIX` acts as a cluster identifier and therefore, it is necessarily common to all nodes in a cluster. The `PREFIX`, however, is as usual, the outcome of a hash function, which ensures that communication load is uniformly balanced over all nodes in the network.

At the next higher tier, the `PREFIX` is enlarged and the `SUFFIX` is reduced by a certain number of bits s to represent the *aggregation* of the children clusters into the new cluster. In algebraic terms, this aggregation corresponds to the action of the next s generators sorted in *quasiminimal* order of CH_n. Without loss of generality (w.l.o.g.), assume that the associated graph of node u at a given tier is CH_m. If the next higher-tier cluster of u emulates $CH_{(m+s)}$, then the $clusterId$ of this cluster will be $\sigma(u, n - (m + s))$. This has the following three effects:

i. This cluster is the result of the aggregation of $\phi(n-(m+s))\phi(n-(m+s-1))\ldots\phi(n-(m+1)) = 2^s$ clusters emulating CH_m;

ii. The identifier space for the participating nodes has enlarged from 2^m to $2^{(m+s)}$; and

iii. The links to add to u in order to maintain logarithmic guarantees on routing correspond to generators $\{\pm 2^{n-(m+1)}, \pm 2^{n-(m+2)}, ..., \pm 2^{n-(m+s)}\}$.

The main benefit of this mapping is that links can be created as dictated by the linking rules of the original DHT, preserving logarithmic guarantees on the out-degree. More specifically,

Definition 34 (Whirl linking rule)**.** *Let u be an arbitrary node in cluster C_u. At next tier, node u sets up a link to a node v in another cluster $C_v \neq C_u$ if and only if v is the closest node that is at least distance 2^i away for some $0 \leq i \leq h$, where h is the index of the largest power of two between u and its successor in C_u.*

To better understand this, we provide the following example:

Example 1. An example of Whirl is shown on Fig. 3.4. Since $n = 3$, each node u is labeled by a binary string of the form $(u_1u_2u_3)$. The figure illustrates how to combine two Chord clusters to produce a DHT graph isomorphic to a Chord network of $N = 2^{(n=3)}$ nodes. Cluster C_0 hosts the nodes with labels of the form $(u_1u_2 0)$ whereas cluster C_1 the nodes labeled $(u_1u_2 1)$. Both clusters approximate CH_2.

For simplicity in discussion, we only consider the routing connections set up by node 0. Since bit y_3 is fixed, the links in cluster C_0 are initialized by considering only the `PREFIX`. Hence, node 0 creates its i^{th} link by finding the closest node whose `PREFIX` is at least distance 2^i away in the diminished identifier space induced by fixing the last bit. Clearly, this does not require to modify the Chord policy for link selection, as these links are virtually set up in a Chord network with 2^2 possible positions instead of $2^{(n=3)}$. The result is that the intra-cluster fingers of this node point to nodes 2 (010) and 4 (100) as in standard Chord. At the next tier, node 0 applies the Chord linking rule to the whole identifier and hence, it establishes a link to node 1 (001).

We remark that some nodes may not create additional links at all. For instance, this eventuality could have occurred to node 0 if node 1 had gone off-line. Astute readers may thus wonder if our constructions lead to a skewed degree distribution, especially if the size of the clusters follows a skewed distribution (*Zipf-like* distribution). Nonetheless, our simulation results demonstrate that such is not the case. The main reason is that graph CH_n can be viewed as two interleaved copies of graph CH_{n-1} and generalizing, as 2^m interleaved copies of CH_{n-m}. This has a positive effect on the distribution of inter-cluster neighbors along the identifier space.

Without loss of generality, consider that a node u joins a leaf cluster approximating CH_{n-m}. Within this cluster, the recursive decomposition of Chord causes the immediate successor of u to be at least at clockwise distance 2^m from it on the Chord circle. This assures at least one exclusive position for each other cluster between u and its successor, which increases the odds of u to end up with an additional link, irrespective of the distribution of nodes in its own cluster. Combined with mapping $h : \mathcal{P} \longrightarrow \mathcal{I}$, this property is of great help to ensure that the union of all clusters does not yield a skewed distribution of nodes. By preserving this property, we take an important step toward inheriting the distributional properties of Chord out-degree.

Theorem 35. *In a Whirl network of N nodes, with nodes distributed uniformly at random on the identifier space, the expected out-degree of a node is bounded by $\log(N-1) + \log N + 2$, irrespective of the structure of the hierarchy.*

Proof. We start by showing that the expected out-degree of a node in a two-tier Whirl network of N nodes is $\log(N-1) + 3$. Then, we generalize this result to more than two levels.

Let $C_1, C_2, ..., C_K$ be the K different clusters in the system. W.l.o.g., consider some node u in C_1. Let n_1 be the number of nodes in C_1, and let CH_m (for some $m \leq n$) be its *emulation* graph. Clearly, its out-degree is given by $X_{in} + X_{out}$, where X_{in} is the number of links to other nodes in C_1 and X_{out} is the number of inter-cluster links. We define the following two indicator random variables:

1. I_k as the indicator that there is at least one other node within distance $d \in \left[2^k, 2^{k+1}\right)$ of u within C_1; and

2. J_k as the indicator for the event that there exists at least one node outside C_1 within distance $d \in \left[2^k, 2^{k+1}\right)$ of u.

Observe that $E[I_k] = Pr\{I_k = 1\}$. To obtain $Pr\{I_k = 1\}$, note first that the probability that some specific node lies in a range of length 2^k is $\frac{2^k}{2^m}$. Therefore, by the Union bound, we have $Pr\{I_k = 1\} \leq \frac{(n_1-1)2^k}{2^m}, \forall 0 \leq k < m$. Moreover, $E[I_k] \leq 1$, for all $0 \leq k < m$. Let $\alpha = m - \lceil\log(n_1 - 1)\rceil$ (assume $n_1 > 2$). By linearity of expectation, the expected value of X_{in} is given by

$$\begin{aligned} E[X_{in}] &= \sum_{k=0}^{m-1} E[I_k] = \sum_{k=0}^{\alpha} E[I_k] + \sum_{\alpha+1}^{m-1} E[I_k] \\ &\leq \sum_{k=0}^{\alpha} \frac{(n_1-1)2^k}{2^m} + \sum_{k=\alpha+1}^{m-1} 1 \\ &= \frac{(n_1-1)}{2^m}\left(2^{\alpha+1} - 1\right) + m - \alpha - 1 \quad \text{(for } n_1 > 2) \\ &= \frac{2(n_1-1)}{2^{\lceil\log(n_1-1)\rceil}} - \frac{(n_1-1)}{2^m} + \lceil\log(n_1-1)\rceil - 1 \\ &< \log(n_1-1) + 1 \end{aligned} \tag{3.1}$$

Combined with the fact that $E[X_{in}] = 1$ for $n_1 = 2$, we have $E[X_{in}] \leq \log(n_1 - 1) + 1$. Thus, once we bound the expected value of X_{out}, we shall be done. As before, we first bound $E[J_k]$. Observe that the expected number of nodes outside C_1 that lie within the range $[u, succ(u))$ is $\frac{(N-n_1)}{n_1}$. Therefore we have $E[J_k] \leq \frac{(N-n_1)}{n_1}\frac{2^k}{d}$, where d is the clockwise distance between u and $succ(u)$ in the global ring. Now notice that the automorphism of group $\mathbb{Z}_{2^n}$ defined by $s \longrightarrow 2^{(n-m)}s$ maps the generating set of CH_m to the generating set of CH_n, i.e., the emulation graph of the global ring. As a result, we get that α corresponds to $l = \alpha + (n - m) = n - \lceil\log(n_1 - 1)\rceil$. Since d can be at most 2^l, it follows that $E[J_k] \leq \frac{(N-n_1)}{n_1}\frac{2^k}{2^l}$. By a similar reasoning to the above, it is easy to see that

$$E[X_{out}] \leq \sum_{k=0}^{l-1} E[J_k] = \sum_{k=0}^{\beta} E[J_k] + \sum_{\beta+1}^{l-1} E[J_k] \tag{3.2}$$

where $\beta = l - \left\lceil\log\frac{(N-n_1)}{n_1}\right\rceil$ (assume $\frac{(N-n_1)}{n_1} \geq 2$). There are two cases to be considered: (1) $\frac{(N-n_1)}{n_1} \leq 2$ and (2) $\frac{(N-n_1)}{n_1} > 2$. We show that both cases lead to a total out-degree of at most $\log(N - 1) + 3$.

Case (1): $\frac{(N-n_1)}{n_1} \leq 2$. In this case, $E[X_{out}]$ is at most $\frac{(N-n_1)}{n_1 2^l}\left(2^l - 1\right) < \frac{(N-n_1)}{n_1} < 2$. Thus, the total out-degree is at most $\log(n_1 - 1) + 3$, which is maximized when $n_1 = N$, so our claim follows.

Case (2): $\frac{(N-n_1)}{n_1} > 2$. Using the same arguments as in the derivation of Eq.(3.1) above, the expectation of X_{out} is $E[X_{out}] \leq \log\left(\frac{(N-n_1)}{n_1}\right) + 1$. Combining this fact with Eq.(3.1) gives that the expected total out-degree is at most $\log\left[\left(\frac{(N-n_1)}{n_1}\right)(n_1 - 1)\right] + 2$, which is maximized for $n_1 = 2$. The total out-degree is at most $\log(N - 2) + 1 < \log(N - 1) + 3$.

Now we generalize this result for more than two tiers. At each tier, our construction aggregates all the clusters from the lower tier to produce the telescoping cluster for this tier. Now say there is a cluster with n_t nodes which has been produced after aggregating t times. By reasoning similar to the one above, the expected out-degree for a node in this cluster is given by $\log(n_t - 1) + t + 2$. Since there are at most $\log N$ aggregations of clusters, where nodes in one cluster are aggregated with at least as many nodes outside that cluster, we have that the expected out-degree is at most $\log(N - 1) + \log N + 2$. This concludes the proof.

□

Lemma 36. *With high probability, when N peers are distributed uniformly at random on the Chord circle, the minimum distance between two consecutive nodes is $\Omega\left(2^n \frac{\ln N}{N^2}\right)$.*

Proof. Consider that we are tossing the joining nodes onto the Chord circle one at a time. Let ξ_i be the event that some interval is of length α or less when the i^{th} node joins the Chord network. Clearly, $Pr\left\{\xi_N\right\} = 1 - Pr\left\{\overline{\xi_N}\right\}$, which is

$$1 - \prod_{i=2}^{N} Pr\left\{\overline{\xi_i}|\overline{\xi_{i-1}}\right\} \leq 1 - \prod_{i=1}^{N} \frac{(2^n - i\alpha)}{2^n}.$$

Note that $\alpha = 2^n\beta$, for some β, $0 \leq \beta \leq 1$). Then, we have

$$Pr\left\{\xi_N\right\} \leq 1 - e^{-\sum_{i=1}^{N} i\beta} \leq 1 - e^{-\beta\frac{(N+1)N}{2}}.$$

For $\beta = \Omega\left(\frac{\ln N}{N^2}\right)$, this probability is $1 - \frac{1}{N}$. This completes the proof. □

Theorem 37. *The out-degree of any node in Chord is $\mathcal{O}(\log N)$ with high probability (w.h.p.).*

Proof. From Lemma 36, the distance between two consecutive nodes is at least $2^n \frac{C \ln N}{N^2}$ *w.h.p.*, for some small positive constant C. Consequently, for all i, $i < n - 2\log N + \log C \ln N$, we have that the i^{th} finger of a node points to its immediate successor *w.h.p.* Hence, the out-degree of a node will be at most $n - (n - 2\log N + \log C \ln N) = 2\log N - \log C \ln N$ *w.h.p.*, which completes the proof. □

Theorem 38. *The out-degree of any node in Whirl is $\mathcal{O}(\log N)$ with high probability.*

Proof. The proof of this theorem follows from the proofs of Theorems 35 and Theorem 37. □

Routing In Whirl

We now shift gears to routing. First, we describe Whirl's routing protocol. Then, we analyze its theoretical performance compared to Chord. Informally, routing in Whirl can be viewed as generalization of routing in standard Chord. Messages are forwarded along those links that diminish the distance by some power of two. So routing is clockwise and greedy, never overshooting the destination.

Routing protocol

Let u be a node starting a query for key k. The routing protocol can be described as follows:

Initially, u attempts to take the largest possible steps towards the destination by routing within its leaf cluster. Meanwhile, if routing encounters some node caching the key, the query halts and the corresponding value is returned. If not, the query arrives at the closest predecessor of k in this cluster. Let u' be this node. Then, u' switches to the next higher-tier cluster and continues routing in that cluster. This procedure is repeated until either the manager for key k is found or the query contacts some intermediate node storing a copy of k.

Even if the query eventually switches to the global cluster, the total number of hops incurred by Whirl is almost the same as Chord, as a query always restarts from the point where it was left off in the lower-tier cluster.

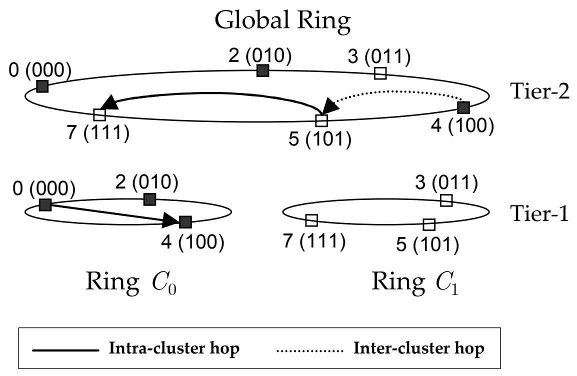


Figure 3.5: *Routing example in Whirl. Node* 0 *routes a query for key* $k = 7$*. Within each cluster, clockwise* GREEDY *routing is performed until the node responsible for the key is found.*

Example 2. Fig. 3.5 shows the routing path a request for key 7 and originating at peer 0 follows through the infrastructure. Initially, node 0 routes the query along its leaf cluster, until the closest predecessor of key 7 is found in this ring. As shown in the figure, such a predecessor is node 4. Next, node 4 switches to the next higher-level ring, taking an *inter-cluster hop* that leads the query to node 5. Finally, node 5 uses GREEDY, clockwise routing to forward the query to node 7, i.e., the node responsible for the key.

We now characterize the number of hops required for routing in Whirl.

Lemma 39. *Assume that* N *peers are distributed regularly on the Chord ring. Then, the expected number of routing hops between two Chord nodes is* $\frac{1}{2}\log N$, $N > 1$.

Proof. Consider two randomly chosen nodes u and w with path $u = v_0, v_1, ..., v_t = w$. The search strategy of Chord halves the clockwise distance to the destination at each step. Hence, for every j, $d(v_{j+1}, w) \leq d(v_j, w)/2$. Consequently, for $j = \log N$, we have that $d(v_j, w) \leq \frac{2^n}{2^{\log N}} = \frac{2^n}{N}$. W.h.p., it can be easily seen that v_j must have a successor pointer to w provided the number of nodes in the successor list is sufficiently large. Thus, we have that Chord can resolve all requests within $\log N$ hops and more importantly, the sequence of jumps is unique: two jumps of size 2^i are replaced with a single (more optimal) jump of size 2^{i+1}. So we obtain that any path of length l is formed by following *exactly* l edges, all with distinct labels. This means that each Chord node can reach $\binom{\log N}{l}$ other nodes at length l. Since the probability that each jump is selected is $\frac{1}{2}$, the *probability density function* of routing path length l is given by a *binomial distribution* with parameter $p = \frac{1}{2}$. Hence, the average path length is

$$\sum_{l=1}^{\log N} l \binom{\log N}{l} \left(\frac{1}{2}\right)^{l} \left(\frac{1}{2}\right)^{\log N - l} = \frac{1}{2}\log N$$

□

Theorem 40. *In an Whirl overlay of* N *nodes, with nodes regularly placed around the Chord circle, the expected number of routing hops between two nodes is* $\frac{1}{2}\log N$, $N > 1$.

Sketch of proof. The proof follows from a generalization of Lemma 39. We make use of the following fact. With the assumption that nodes are uniformly distributed in $[0, 2^n)$, it can be easily seen that the expected number of routing hops between any two nodes that are within a distance d is bounded by $\frac{1}{2}\log\left(N\frac{d}{2^n}\right)$.

First, observe that routing in Whirl can be visualized as an alternating sequence of intra- and inter-cluster hops. Routing always starts in the cluster where the source belongs to, generating a sequence of intra-cluster hops until the message reaches the closest predecessor, say u', of the key. Then, node u' uses an inter-cluster link to forward the message to the neighbor that is closest to the destination, say u''. Node u'' then performs the same operation as the source: it first attempts to route within its cluster and then it follows an inter-cluster link. This sequence is repeated until the message reaches the destination.

With every hop, Whirl ensures that the remaining distance to the destination decreases by at least a factor of $\frac{1}{2}$. Hence, the number of intra-cluster hops will gradually decrease in each new sequence. Combining this fact with the following two observations, we shall be able to prove the theorem:

1. Once intra-cluster routing is accomplished, the expected distance remaining to the destination is at most $\frac{2^n}{2n_c}$, where n_c denotes the number of nodes in the current cluster; and

2. Each inter-cluster hop is expected to decrease the distance by at least a factor of $\frac{1}{2}$.

For a random distance, now consider a path traversing two clusters, C_1 and C_2, which starts from a node in C_1 and then follows an *inter-cluster link* to a node u in C_2. Further, assume that C_1 and C_2 contain n_1 and n_2 nodes, respectively. Then, it is easy to see that the expected number of nodes between u and the destination is bounded by $\frac{n_2}{2^n}$ times the distance from u to the destination. Clearly, this is $\frac{n_2}{2^n}\left(\frac{1}{2}\frac{2^n}{2n_1}\right) = \frac{n_2}{4n_1}$. By Lemma 39, then it follows that the expect number of *intra-cluster hops* is at most

$$\frac{1}{2}\log n_1 + \frac{1}{2}\log\left[(n_1+n_2)\frac{1}{4n_1}\right] = \frac{1}{2}\log(n_1+n_2) - \frac{1}{2}\log 4 = \frac{1}{2}\log(n_1+n_2) - 1 \tag{3.3}$$

Eq. (3.3) can be generalized to any number of clusters. In general, when a query traverses K clusters over a random distance, the expected number of intra-cluster hops is given by

$$\frac{1}{2}(n_1+n_2+...+n_{K-1}) - \frac{1}{2}\log 4^{K-1} = \frac{1}{2}\log(n_1+n_2+...+n_{K-1}) - K + 1 \tag{3.4}$$

which is bounded by $\frac{1}{2}\log N - K + 1$. Since the number of inter-cluster hops is $K-1$, the expected number of routing hops is thus $\frac{1}{2}\log N$. □

Theorem 41. *In a Whirl network of N nodes, the number of routing hops to route between any two nodes is $\mathcal{O}(\log N)$ with high probability.*

Proof. The proof of this theorem follows from the proof of Theorem 40. □

In what follows, we quantify the potential improvement in query efficiency achieved by Whirl with respect to Chord. The idea is to validate theoretically the improvement on query efficiency brought by path locality.

Routing Benefits. In general, existing DHT works assume that communication is *uniform*, i.e., all nodes have an equal probability of processing a query. While this assumption is acceptable for file-sharing applications, locality in communication is frequent in many distributed applications, such as content delivery networks and data management systems. In this sense, our constructions offer an ideal substrate. They organize peers into clusters, which allow communication locality to be exploited by the applications built atop.

In this section, we drop this assumption and assume that nodes within a cluster communicate with a higher (or lower) probability than do two nodes from different clusters. For simplicity, we also assume a two-tier hierarchy, with multiple leaf clusters but with only one telescoping cluster: *the global cluster*. Recall that the global cluster spans all nodes in the system.

Now, let γ denote the probability that both the source and the destination peers belong to the same leaf cluster. Hence, $(1-\gamma)$ is the probability of inter-cluster communication. Thus, the larger the value of γ, the more likely the locality in communication. Further, we assume that:

- *Intra-cluster communication* is distributed uniformly at random over the nodes in each cluster. Put differently, a node routes an intra-cluster message to each other node in its cluster with equal probability; and
- *Inter-cluster communication* is distributed uniformly at random. That is, a node forwards an inter-cluster message to each other node outside its leaf cluster with equal probability.

In what follows, we assess the improvement in query efficiency achieved by Whirl as a function of γ. The reason is that it is difficult to foresee what values of γ a particular application may expect in practice. Intuitively, if intra-cluster requests are frequent, we should expect a noticeable improvement in query efficiency from Whirl. We provide analytical evidence for this conjecture.

For simplicity in discussion, we assume that the number of nodes is a power of two, $N = 2^\alpha$, $\alpha \leq n$. Further, we assume that the number of nodes in each leaf cluster is the same and equal to 2^m, $m < \alpha$. Hence, the number of leaf clusters is $2^\alpha - 2^m$. Now we derive the average path length, $\mathcal{R}_{Whirl}$, of Whirl. Clearly, $\mathcal{R}_{Whirl}$ is given by

$$\mathcal{R}_{Whirl} = \gamma\mathcal{R}_{\text{in}} + (1-\gamma)\mathcal{R}_{\text{out}}, \tag{3.5}$$

where $\mathcal{R}_{\text{in}}$ denotes the average number of routing hops between two nodes in a leaf cluster, and $\mathcal{R}_{\text{out}}$ is the average path length between two nodes belonging to different clusters. By Lemma 39, $\mathcal{R}_{\text{in}} = \frac{1}{2}m$. Thus, once we compute $\mathcal{R}_{\text{out}}$, we shall be done. First, note that the number of nodes at $(i+j)$ hops from the source equals to $\binom{m}{i}\binom{\alpha-m}{j}$, where $i(i>0)$ is contributed by the jumps from set $\{+2^{n-(m+1)}, +2^{n-(m+2)}, ..., +2^{n-1}\}$ required for intra-cluster routing, and $j(j>0)$ is contributed by the remaining $(\alpha - m)$ jumps. Then, it easy to see that

$$\mathcal{R}_{\text{out}} = \frac{\sum_{i=0}^{m}\sum_{j=1}^{\alpha}\binom{m}{i}\binom{\alpha-m}{j}(i+j)}{2^\alpha - 2^m} = \frac{\alpha 2^{\alpha-1} - m2^{m-1}}{2^\alpha - 2^m} \tag{3.6}$$

Combining Eq. (3.5) and (3.6) gives

$$\mathcal{R}_{Whirl} = \gamma\left(\frac{m}{2}\right) + (1-\gamma)\left(\frac{\alpha 2^{\alpha-1} - m2^{m-1}}{2^\alpha - 2^m}\right). \tag{3.7}$$

Note that when $\gamma = 2^{m-\alpha}$ (i.e., the probability of intra-cluster communication is proportional to m), $\mathcal{R}_{Whirl} = \frac{1}{2}\alpha$ as in Chord. The average path length for both Chord and Whirl is illustrated

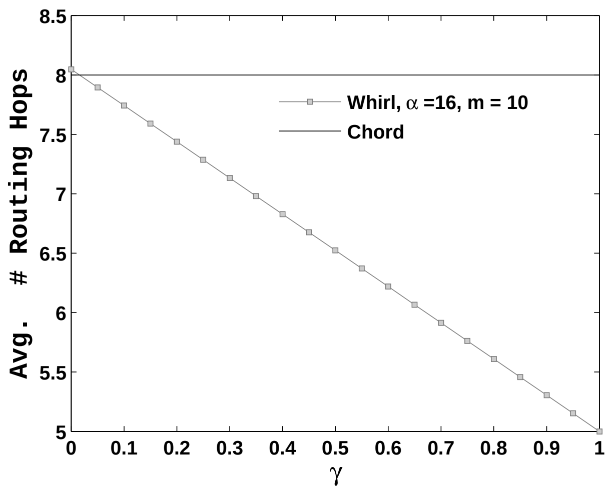


Figure 3.6: *Comparison between the average path length of Chord and Whirl for* $\alpha = 16$ *and* $m = 10$. *The value of* γ *is varied from* 0.0 *to* 1.0

in Fig. 3.6. Both systems have a network size $N = 2^{16}$. For Whirl, the cluster size is fixed at $2^{m=10}$. The value of γ is increased from 0 to 1. The main observation is that as γ approaches to 1, Whirl can significantly reduce the average path length while Chord misses any opportunity to do it due to the lack of a clustered structure. This makes apparent the necessity of new techniques, able to capture locality such as Cyclone.

Maintenance in Whirl

Dynamic maintenance in Whirl is a natural generalization of dynamic maintenance in Chord. It must deal with nodes joining and leaving the system but also with node failures. In what follows, we restrict our attention to the $join$ protocol. The $leave$ protocol is similar and has been omitted for clarity. Since fault-tolerance mechanisms are the same as Chord, we only give a short description of how to apply them to Whirl.

Join protocol

Let u denote the joining peer. The $join$ protocol consists of three phases:

1. *Find an arbitrary peer in the destination leaf cluster.* To enter the system, node u must know at least one existing peer in the destination cluster. This information can be provided by many different mechanisms. For example, u could look up a central server for this knowledge; this server could maintain a cache of live peers from each cluster. Alternatively, this information could be stored in the DHT itself, and u might obtain it by simply querying the DHT. In any case, we assume that u is able to obtain the IP address of some live node in the destination cluster.

2. *Using the peer found in Phase 1, construct the finger table.* Let u' be the peer found in Phase 1. Using this node, u "inserts" itself in all clusters using the standard Chord technique for insertion, applied at each tier of the hierarchy. Specifically, node u' routes a query using u's $nodeId$ as key. This query reaches the predecessor of u at each tier of the hierarchy. At this

point, the $join$ protocol proceeds as follows. After inserting itself after each predecessor p, u notifies $succ(p)$ to announce itself as its new predecessor. Once this is done, u constructs its finger table using the linking rules defined in Section 3.4.3. Since node u requires a `Get` operation for each finger establishment, the cost of initializing $\mathcal{O}(\log N)$ fingers is $\mathcal{O}(\log^2 N)$, the same complexity as Chord.

3. *Copy all keys for which node u has become their successor to node u.*

By itself, the $join$ protocol does not make the rest of the network to be aware of u. However, u may become the finger of other nodes in the system. To correct the topological changes caused by the addition of node u, we adopt function $fixfingers$ of Chord [4]. As in Chord, each Whirl node runs periodically $fixfingers$ to make sure its finger table entries are correct. Specifically, each call to this function updates one finger entry. The main benefit of this strategy is that maintenance complexity is upper bounded by $\mathcal{O}(\log^2 N)$. By Theorem 41, we have that each call to $fixfingers$ costs $\mathcal{O}(\log N)$ messages. Since $\mathcal{O}(\log N)$ calls to $fixfingers$ are required to initialize the entire finger table, the complexity is thus $\mathcal{O}(\log^2 N)$.

Key Observation: Phase 2 can be accelerated by using the finger tables of u's predecessors. That is, the i^{th} finger of u can be efficiently obtained by asking the i^{th} finger of the corresponding predecessor. This operation costs $\mathcal{O}(\log N \log\log N)$ *w.h.p.*, since the number of nodes that must be contacted to initialize the new finger table is $\mathcal{O}(\log N)$, and each one takes $\mathcal{O}(\log\log N)$ hops *w.h.p.*, and only $\mathcal{O}(1)$ hop on average [42].

Fault-tolerance. For fault tolerance reasons, each Chord node maintains a *successor list* of $\Omega(\log N)$ *successor pointers*, containing the node's first $\Omega(\log N)$ successors in the ring. To inherit the same robustness, each Whirl node maintains a separate successor list for each cluster it belongs to. We note that successor lists are cheap to maintain since can be updated by passing a single message along the ring. Further, they do not incur keep-alive maintenance overhead, as they do not correspond to physical connections.

3.5 Adaptability to Other DHTs

In this section we show the usefulness of our strategy, by applying Cyclone to six families of DHTs that are benchmarks among P2P systems:

R-Chord [30] and Symphony [11] Yet another variant of Chord is R-Chord, which differs from standard Chord on the great amount of *freedom* it provides in choosing neighbors. In R-Chord, each node u picks n neighbors, where the i^{th} neighbor can be *any* node within clockwise distance $[2^{i-1}, 2^i)$ of u, instead of the closest node that is at least 2^{i-1} away. For this reason, we can regard the construction of R-Chord in the same way as Chord, but with the non-deterministic rule for link selection instead of the deterministic rule.

Symphony, a randomized version of Chord, was initially conceived as a constant degree DHT. To approximate it as a Cayley graph, it is necessary to transform it into a logarithmic degree network. This requires that each node u creates $\mathcal{O}(\log N)$ links to other nodes, each chosen independently at random, such that the probability of selecting a node v is inversely proportional to the distance from u to v. Such a generalization permits viewing Symphony as a non-deterministic version of Chord, which leads to the following construction:

Each node u initially creates $\lfloor \log n_1 \rfloor$ connections, where n_1 is the number of nodes in its leaf cluster. Leaf neighbors are selected as in Symphony, i.e., with a probability that is inversely proportional to the distance between u and each neighbor. At the next higher tier, u creates $\lfloor \log n_2 \rfloor$ links by the same random process, where n_2 is the number of nodes in the next higher-tier cluster, but retains only those connections that are closer than its successor in the lower tier. This process is repeated successively at all tiers of the hierarchy. Notice that both Symphony and its Cyclone version require to know each n_i exactly. This can be done using the techniques developed in [43].

Kademlia [6] and P-Grid [12] Kademlia can be viewed as a *hypercube* version of R-Chord. Like in R-Chord, each node creates a link to an arbitrary node within distance $\left[2^{i-1}, 2^i\right)$, for all $0 < i \leq n$. However, it differs from R-Chord in the definition of the *distance function* required for routing greedily. More specifically, Kademlia uses the XOR metric rather than clockwise distance between pairs of nodes. In other words, the distance $d(u, v)$ between two nodes u and v is the XOR of their binary $nodeIds$, i.e., $d(u, v) = \sum_{i=1}^{n} |v_i - u_i| 2^{i-1}$.

Let K_n denote the Kademlia graph of order 2^n. Using hypercube as the Cayley representation of Kademlia, it is easy to see that K_n can be partitioned into $\phi(n) = 2$ copies of K_{n-1}, with the first copy containing all the nodes labeled $(0u_1...u_n)$ and the second copy the nodes labeled $(1u_2...u_n)$. Let us consider that leaf clusters emulate graph K_m, for some $m \leq n$. Altogether, this means that a hierarchy should be mapped to Kademlia by logically partitioning the first $\alpha = (n - m)$ bits of node labels. W.l.o.g., assume that prefixes $u_1u_2...u_\alpha$ are divided into two parts: $u_1u_2...u_\beta$ and $u_{\beta+1}...u_\alpha$, in order to create a hierarchy of three levels. Our construction for Kademlia is then as follows. Each node creates its links as dictated by Kademlia. For its leaf cluster, each node sets up *only* the links that cover distance ranges $\left[2^{i-1}, 2^i\right)$, for $i = 1 \ldots \alpha$. At the second tier, nodes create the connections that cover ranges $\left[2^{i-1}, 2^i\right)$, for $i = (\alpha + 1) \ldots (\beta - 1)$. Finally, the rest of links are established within the global cluster.

Another approximation of hypercube is P-Grid. P-Grid implements a binary trie abstraction, i.e., a tree for storing binary strings in which there is one node for every common prefix. Specifically, each node u is associated with a leaf of the tree and hence to a binary string $\pi(u)$ of the identifier space. Notice that $\pi(u)$ is also the path from u to the root of the tree. For routing, node u maintains for each prefix $\pi(u, i)$ of $\pi(u)$ of length i a link to a node v such that $\overline{\pi(v, i)} = \pi(u, i)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. Geometrically, this means that at each level of the prefix tree, node u has a pointer to a node not contained in its subtree. This enables the efficient implementation of prefix routing. Consequently, the appropriate way to map a hierarchy of telescoping clusters to P-Grid is to group nodes into clusters according to a common prefix, in a similar way to Kademlia construction.

Pastry [5] and Tapestry [10] Finally, we consider the variant of PRR trees [44] employed in Pastry and Tapestry. Nodes are labeled by a string $(u_1u_2...u_n)$ of n digits in base b. Each node is connected to $n(b-1)$ distinct neighbors with labels $(u_1u_2...u_{i-1}x\,y_{i+i}...y_n)$, for $0 < i \leq n$ and $x \neq u_i$, $u_i \in \{0, ..., b-1\}$. Besides, Pastry uses a ring-like geometry in a similar fashion to Chord to make sure that requests can always make progress towards the destination if no entry is available for routing. This makes its adaptation more tedious. To simplify the process, we make use of the following observation.

Among the different possibilities for the remaining digits, i.e., $y_{i+1}, ..., y_n$, both DHTs select a node that is nearby with respect to some proximity measure such as round-trip-time. However, if we pick $y_{i+1} = u_{i+1}, ..., y_n = u_n$, it is possible to represent Pastry and Tapestry as an hypercube. Since the generating set of the hypercube is quasiminimal for any permutation of the generators, the construction rule for Pastry and Tapestry should be based on the permutation that preserves the uniformity in the spatial distribution of nodes and keys. Such an order consists in picking the $(b-1)$ generators that determine the edges for the i^{th} digit of node labels in right-to-left order. This implies that the hierarchical construction for Pastry and Tapestry is carried out by logically partitioning the suffix part of Pastry labels into levels. In this sense, the construction for Pastry and Taspestry is similar to Whirl. A joining node starts setting up its links by applying the Pastry linking rule *only* to the nodes belonging to its leaf cluster. At the next higher tier, the joining peer uses again Pastry linking rule to create its links in that tier. This process is repeated at *successively* higher tiers of the hierarchy, until the links at the global cluster are established (if needed).

3.6 Performance Evaluation

To conclude the description of our framework, we decided to use a simulator and evaluate Whirl. Since none of the tools available fitted our needs, we built our own simulator. We evaluated the following characteristics of Whirl:

- *Out-degree distribution*;
- *Average path length*;
- *Average query time*, using both synthetic and real Internet latency models; and
- *Load Balancing*

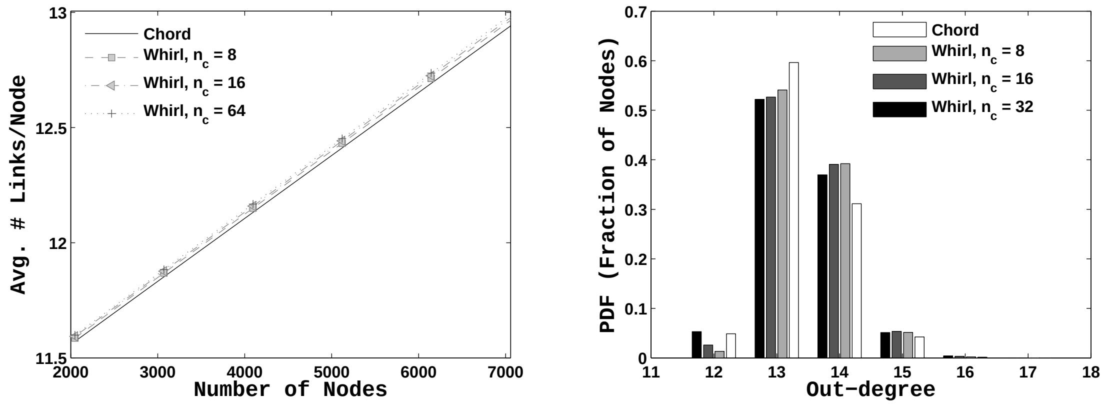
Setup For simulations conducted in this evaluation, we used a Whirl hierarchy of 2 tiers. The simulation model consisted of N participant peers, gathered in a varying number of leaf clusters n_c. Specifically, we set n_c to 8, 16 and 32 in all tests, which corresponded to $nodeId$ `SUFFIX`es of 3, 4 and 5 bits, respectively. We restricted Chord's identifier space to $[0, 2^{n=16})$. Further, we used two different distributions to assign nodes to their clusters:

- A *uniform* distribution; and
- A *Zipfian* distribution where the number of nodes in the i^{th} largest cluster is proportional to $\frac{1}{i^{0.95}}$.

Since the results reported by our simulations were practically identical for both distributions, we will only show the results corresponding to the Zipfian distribution. As a comparison baseline, we implemented Chord as specified in [4].

3.6.1 Basic Properties: Degree and Average Path Length

First we evaluated the degree distribution of Whirl. Fig. 3.7a plots the average number of links per node, as a function of the size of the network. The comparison shows that the average out-degree



a) Average number of links per node.

b) Out-degree distribution for $N = 10,000$.

Figure 3.7: *Out-degree comparison between Chord and Whirl.*

is extremely close to $\log N$ irrespective of the number of clusters. This implies that Whirl retains Chord complexity as the network grows, which facilitates scalability as well as link maintenance.

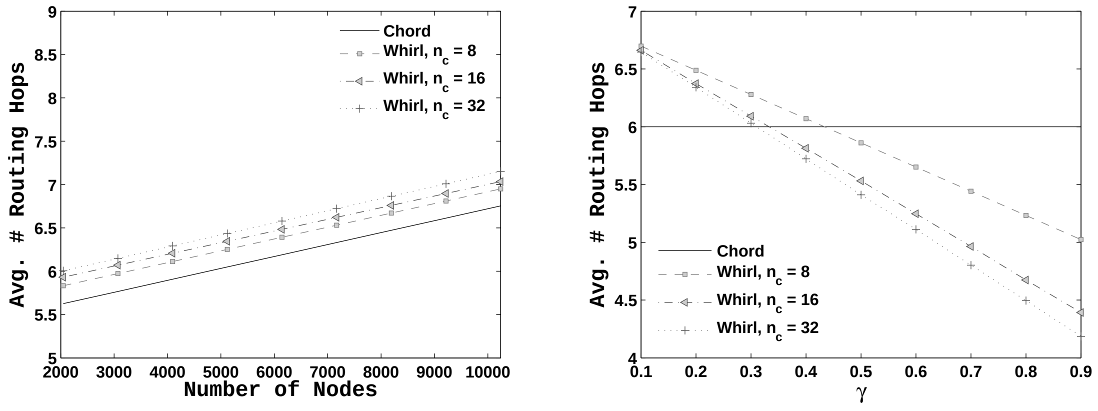
Fig. 3.7b plots the distribution of the number of links for a network of $N = 10,000$ nodes. The results show that the distribution "flattens out" to the right of the average (of 13.29 links per node) as the number of clusters increases. This can be explained by the following observation: as n_c increases, the distribution of the arc lengths at the global cluster is increasingly skewed. We also note that the maximum number of links increases slightly, which indicates that the equivalence between Whirl and Chord is near perfect in this regard. Whirl, moreover, enables the exploitation of path locality thanks to its clustered architecture.

In Fig. 3.8a, we depict the average number of hops required to route between two nodes, as a function of the network size. The results show that the average path length of Whirl is $\frac{1}{2}\log N + c$, where c is a small constant depending on the number of clusters. This phenomenon can be again explained by our earlier observation. Moreover, it is noteworthy to remark that this increase is at most 0.5, irrespective of the number of clusters (and even beyond to what we have represented on this figure).

Finally, we measured the hop-count savings of Whirl with respect to Chord, as a function of γ. The network size was set to $4,096$ for this test. As shown in Fig. 3.8b, for communication patterns with more than 40% of locality, Whirl increasingly outperformed Chord. For $\gamma = 0.9$ and $n_c = 32$, Whirl exhibited a decrease of 33% in the average path length. This can be easily explained by the reduction in the search space offered by each cluster. As γ approaches to 1, inter-cluster links are taken less frequently; so intra-cluster routing takes place more commonly. Since each cluster approximates a Chord graph of smaller order than the one in the global cluster, the average path length reduces proportionally to the logarithm of the average cluster size, as shown in this figure.

3.6.2 Query Latency

In an effort to assess scalability, the typical measure that has been used is the average number of routing hops. In most cases, either logarithmic complexity (e.g., Chord [4]) or polylogarithmic



a) Average path length.

b) Routing improvements achieved by Whirl with respect to Chord.

Figure 3.8: *Routing comparison between Chord and Whirl.*

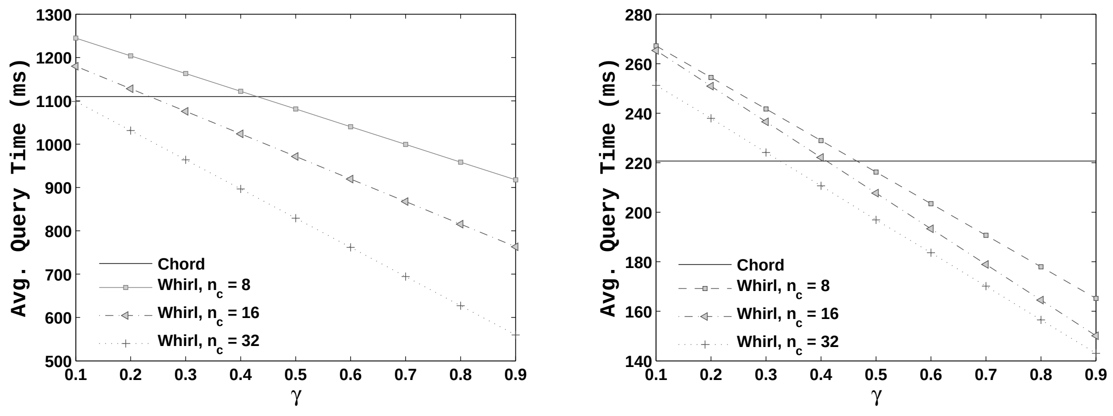
complexity (e.g., Symphony [11]) has been sought. However, if the only requirement is a constant number of hops, say three, a query could go, for instance, from Berlin to Paris through Tokyo and New York. Although the number of hops is small, network latency was indeterminate and high in this example. While bounding the number of hops by a logarithm is important, more important is that the total cost of communication between peers is low.

In this section, we evaluate routing in terms of query time, rather in terms of number of routing hops. We used two different datasets to model real communication latencies:

- **GT-ITM [45]:** The first dataset was produced using GT-ITM topology generator. The resulting topology provided a backbone of $1,000$ core routers arranged in a hierarchical manner. Specifically, it contained 3 transit domains at the top level, with 4 routers in each. Each transit router was broken into an average of 3 stubs, and each stub contained an average of 24 routers. Network latencies (edge weights) were assigned according to GT-ITM default policy. Finally, we attached an average number of $\frac{N}{1,000}$ peers to each stub router. Such an assignment was carried out to make possible the creation of a *physical* topology including all the N participating peers.

- **DIMES [46]:** The second dataset was downloaded from DIMES, the largest-scale deployed Internet exploration tool concerning the number of monitors, more than $8,700$ monitors scattered over five continents. In particular, we used the inferred AS graph corresponding to September of 2007. To assign the weights to edges, we selected the minimum latency (in milliseconds) between the first hop in the source AS to the last hop in the destination AS. Further, we removed parallel edges, always maintaining the edge with the largest delay. For the simulations, we chose a random subset of $1,000$ ASes and attached an average number of $\frac{N}{1,000}$ peers to each one, so that the final topology mimicked a real network of exactly N peers interconnected through $1,000$ ASes.

To obtain exactly n_c clusters with an accuracy similar to the one expected in reality, we made use of Vivaldi [47]. Vivaldi is a distributed algorithm that assigns synthetic coordinates to nodes,



a) GT-ITM dataset. b) DIMES dataset.

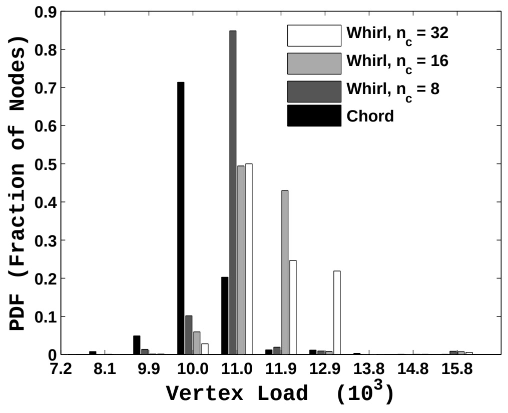
Figure 3.9: *Improvements in query time achieved by Whirl as a function of communication locality, γ.*

so that the Euclidean distance between the coordinates of two nodes approximates the network latency between them. Each node computes its coordinates by simulating its position in a network of physical springs. Vivaldi is *decentralized* and *efficient*, in the sense that no fixed infrastructure needs to be deployed and a new node can compute its coordinates after collecting latency information from only few other nodes. These features align well with the requirements of peer-to-peer systems; hence we chose Vivaldi.

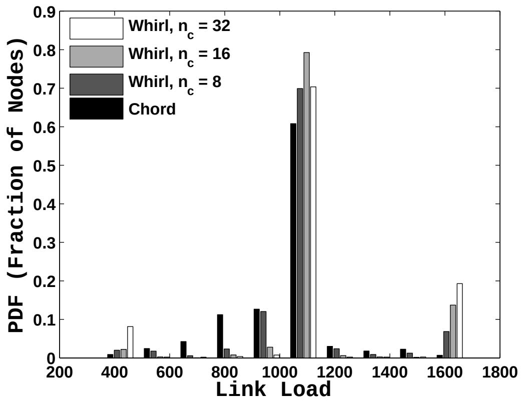
The use of coordinates allowed us to deal with the creation of clusters as a traditional clustering problem which is solvable by popular clustering algorithms such as K-Means [48]. K-Means is a general-purpose heuristic to cluster multi-dimensional data. It is simple and fast, and works well with a wide variety of data distributions. Moreover, it has only two parameters: the number of clusters and the degree of accuracy. K-Means obtains a set of n_c clusters in such a way that the sum of the squared Euclidean distance from each node to the center of the cluster it was assigned to is minimized.

For Vivaldi, we used the implementation provided by Bamboo [49]. Then, we ran Vivaldi on the two datasets to obtain two independent sets of coordinates. To stabilize the coordinates, we fed the same set of measurements as an input to Vivaldi $1,000$ times. Afterward, we ran K-Means 5 times on the set of Vivaldi coordinates and returned the best solution. For this experiment, the network size was set to $N = 4,096$.

Fig. 3.9 depicts the results corresponding to GT-ITM (left) and DIMES (right), as a function of communication locality (γ). As can be seen in the figure, for communication patterns with more than 40% of locality, Whirl becomes the "winner", obtaining savings of up to 50% to the reference query time established by Chord in GT-ITM. In DIMES, the results are somewhat worse, with a cut-down on query time to 35%. Hence, we see that, irrespective of the underlying latency model, Whirl has the potential to achieve significant reductions in query time by means of its hierarchical structure, which makes our technique very appealing in practice.



a) Distribution of vertex load.



b) Distribution of link load.

Figure 3.10: *Load balancing comparison between Chord and Whirl.*

3.6.3 Load Balancing

We also evaluated the load balancing characteristic of Whirl. The question we set out to answer was whether Whirl was able to maintain a satisfactory load-balancing among peers, an assurance provided by random hash assignments in Chord. To shed light on this question, we empirically derived the load distribution on nodes and links imposed by GREEDY routing in both designs. In this case, we limited the number of participants to $2,048$ nodes. The main reason for doing so was to reduce the execution time, as accurate computation of load requires to consider the $N(N-1)$ *greedy* paths specified for every pair of nodes.

Fig. 3.10a shows the fraction of peers that appeared as intermediate routers in a particular number of routing paths. From the inspection of this figure, it can be inferred that all distributions present a similar shape. However, Whirl distributions are shifted rightward (downward) by a variable amount relative to Chord distribution. The quantity of shift increases gradually with the number of clusters, albeit the fraction of nodes whose load is more than 1.5 times the vertex-forwarding index of Chord is insignificant. Recall that the vertex-forwarding index of Chord is given by expression $2^{n-1}(n-2)+1$ (see Theorem 29), which is equal to $9,217$ in a network of $2^{(n=12)}$ participating nodes. The load distribution over nodes is thus almost uniform and therefore, Whirl has a good load balance, which is comparable, to a remarkable extent, to the load balance characteristic of Chord.

The distribution of link load is depicted in Fig. 3.10b. As can be seen in the figure, more than 65% of links have a load close to $1,024$, which coincides with the arc-forwarding index of Chord (see Theorem 33). In general, what this figure shows is that when the identifier space is not fully occupied, Whirl balances load as well as Chord.

3.7 Conclusions

We have described Cyclone, a generic framework for constructing hierarchical DHTs almost at no cost. We have shown how this framework can be applied to flat DHTs that, while yet maintaining homogeneity, can also achieve significant improvements in communication. As a representative example, we have applied our technique to construct a hierarchical version of Chord. Since our framework is based on the Cayley graph model, it can be made extensible to a wide variety of DHTs, including Chord, Symphony, Pastry, HyperCup, IHOP etc., and in theory, to all DHTs where the number of generators is a function of the size of the Cayley graph. Further, we have quantified the advantages of our constructions in terms of routing and adaptation to the physical network, theoretically when possible, but always by means of extensive simulations.

Possible directions for future work:

- An interesting direction would be to improve our framework by considering other aspects. For example, with $\mathcal{O}(\log N)$ connections per node, Whirl can route in $\mathcal{O}(\log N)$ steps as Chord. If nodes could establish a number of links proportionally to their capacities, *Would Whirl be able to reduce the average path length for leaf clusters without worsening routing at higher tiers?*
- It would be interesting to identify and to address practical issues that arise in the implementation of our hierarchical designs, on PlanetLab, for example.
- Since our hierarchical constructions differ from superpeer systems, it would be very useful to provide a comparison between our designs and superpeer systems to identify the advantages as well as the pitfalls of our constructions relative to superpeer systems. We provide a partial answer to this issue in the next Chapter, where we propose a framework to compare the communication costs of hierarchical designs.

Appendix

Proof of Theorem 19

Proof. Let $V(CH_{n-1})$ and V_k be the vertex set of CH_{n-1} and $CH_n[V_k]$, respectively. Let $f_k(x)$ be a mapping from $V(CH_{n-1})$ to V_k such that $f_k(u) = k + 2u$, $\forall u \in \{0, 1, ..., 2^n - 1\}$ and $0 \leq k < 2$. Notice that f_k is *well-defined*: An element in the domain $V(CH_{n-1})$ has exactly one image in V_k. In the following, we show that f_k is an *edge-preserving bijective function*.

(a). (*one-to-one*)

Let u and v be vertices in $V(CH_{n-1})$ such that $f_k(u) = f_k(v)$. Then,

$$\begin{aligned} f_k(u) &= f_k(v) \\ k + 2u &= k + 2v \\ u &= v \end{aligned}$$

(b). (*onto*)

Clearly, f_k is onto since for any vertex v in V_k, there exists a vertex u in $V(CH_{n-1})$ such that $f_k(u) = v$.

(c). (*edge-preserving*)

c.1. Let u and v be two adjacent vertices in CH_{n-1}. We show that their respective images under f_k are adjacent vertices in $CH_n[V_k]$. Then,

$$\begin{aligned} f_k(v) &\equiv f_k(2^i + u)\ (mod\ 2^n) \\ &\quad \text{(since } u \text{ and } v \text{ are adjacent, } v \equiv 2^i + u\ (mod\ 2^{n-1}) \text{ for some } i \in \{0, 1, ..., n-2\}) \\ &\equiv k + 2(2^i + u)\ (mod\ 2^n) \\ &\equiv 2^{i+1} + (k + 2u)\ (mod\ 2^n) \\ &\equiv 2^{i+1} + f_k(u)\ (mod\ 2^n) \end{aligned}$$

Therefore, vertices $f_k(u)$ and $f_k(v)$ are *adjacent* with respect to the generators 2^{i+1}, $\forall i \in \{0, 1, ...n-2\}$. By a similar argument, it is easy to prove that $f_k(u)$ and $f_k(v)$ are adjacent with respect to -2^{i+1}, $\forall i \in \{0, 1, ..., n-2\}$.

c.2. Let u and v be adjacent vertices in $CH_n[V_k]$. Now, we show that $f_k^{-1}(u)$ and $f_k^{-1}(v)$ are adjacent vertices in CH_{n-1}. Then,

$$\begin{aligned} v &\equiv 2^i + u\ (mod\ 2^n), \qquad \text{for some } i \in \{0, 1, ..., n-1\} \\ &\equiv f_k(2^{i-1} + 2^{-1}(-k + u))\ (mod\ 2^n). \end{aligned}$$

Thus,

$$f_k^{-1}(v) \equiv 2^{i-1} + 2^{-1}(-k + u)\ (mod\ 2^{n-1})$$

Since $f_k^{-1}(u) \equiv 2^{-1}(-k+u)\ (mod\ 2^{n-1})$, it follows that the vertices $f_k^{-1}(u)$ and $f_k^{-1}(v)$ are adjacent in CH_{n-1} with respect to the generators 2^{i-1}, $\forall i \in \{1, ..., n-1\}$. By similar reasoning to that in **c.1.**, it is easy to show that the same follows for the inverse generators -2^{i-1}, $\forall i \in \{1, ..., n-1\}$. □

Proof of Lemma 30

Proof. For $n \leq 1$, note first that CH_n is isomorphic to the complete digraph of 2^n vertices. Hence, CH_n is *arc-transitive*, which completes the first part of the proof.

Suppose that $n \geq 2$. The strategy of the proof is as follows. Let us consider an arc (u, v) of CH_n and compute N^3_{uv}, *the number of distinct cycles of length* 3 *that contain* (u, v). If for two arcs (u, v) and (u', v'), their numbers differ, then CH_n cannot be *arc-transitive*. Now, let us consider that (u, v) is an arc with label 1, $v = u + 1 \ (mod\ 2^n)$, and (u', v') is an arc with label $n - 2$, $v' = u' + 2^{n-2} \ (mod\ 2^n)$. Let us show that $N^3_{uv} \neq N^3_{u'v'}$.

Let us first consider (u, v). Without loss of generality, let us assume that $u = 0$ and $v = 1$. Further, suppose that (u, v) lies in a cycle of length 3. Hence, this cycle is of the form $1 \longrightarrow p \longrightarrow 0 \longrightarrow 1$, with $p \neq 1$ and $p \neq 0$. By definition, this means

$$\begin{aligned} p &\equiv 1 + 2^i \ (mod\ 2^n), &&\text{for some } i \in \{0, 1, ..., n-1\} \\ 0 &\equiv p + 2^j \ (mod\ 2^n), &&\text{for some } j \in \{0, 1, ..., n-1\} \end{aligned}$$

Altogether, this gives $1 + 2^i + 2^j \equiv 0 \ (mod\ 2^n)$. Since 0 is even, this implies that either $1 + 2^i$ or $1 + 2^j$ must be an even integer. Without loss of generality, let us consider that $1 + 2^i$ is even. Then, $i = 0$. Thus, we have that $1 + 2^{j-1} \equiv 0 \ (mod\ 2^n)$, which is impossible. Consequently, $N^3_{uv} = 0$.

Now let us consider (u', v'). Without loss of generality, let us assume that $u' = 0$ and $v' = 2^{n-2}$. We are looking for a cycle of the form $2^{n-2} \longrightarrow q \longrightarrow 0 \longrightarrow 2^{n-2}$ (with $q \neq 2^{n-2}$ and $q \neq 0$). Then,

$$\begin{aligned} q &\equiv 2^{n-2} + 2^i \ (mod\ 2^n), &&\text{for some } i \in \{0, 1, ..., n-1\} \\ 0 &\equiv q + 2^j \ (mod\ 2^n), &&\text{for some } j \in \{0, 1, ..., n-1\} \end{aligned}$$

In this case, we can see that it is possible to find at least two distinct solutions for the doublest (i, j). They are the following: $\{(n-1, n-2), (n-2, n-1)\}$. As a result, $N^3_{u'v'} \leq 2$. Hence, CH_n is not *arc-transitive* for $n \geq 2$. In other words, arcs labeled 1 do not participate in any cycle of length 3, whereas arcs labeled $n - 2$ participate in some cycles; thus, no *automorphism* of CH_n can transfer the first type to the other. $\square$

4 A Comparative Study of Hierarchical DHT Systems

In Chapter 3, we have proposed a framework for building hierarchical DHT designs. However, a significant number of hierarchical DHT designs have been proposed in the literature, which have particularities that distinguish them from our hierarchical constructions. Since no design can be considered "universally" better, what is lacking is an analytic framework to provide the means for identifying the optimal hierarchical design for a given workload.

In this Chapter, we provide such a comparative framework, and we use it to compare the two principal hierarchical DHT designs: the homogeneous design, in which all nodes take equal roles, against the superpeer design, in which a small subset of peers (the most powerful and stable), behave as proxies, interconnecting clusters with highly dynamic membership. Our analysis reveals that, on the contrary to what was initially expected, the costs incurred by the superpeer design are not necessarily minimized.

4.1 Introduction

"Your faith was strong but you needed proof"

Cohen, Leonard

In the previous Chapter, we presented our framework for constructing hierarchical DHTs from their flat designs. Although our framework is novel in many aspects, there exist other hierarchical designs in the literature that present particularities that complement our work, and make hard the identification of under which circumstances our design is better. The same occurs between other existing designs. For instance, a software engineer may choose our constructions for their graph-theoretic properties, while an Autonomous System (AS) administrator may select the design offering the greatest amount of administrative autonomy.

In such a relative wide sea of designs: Coral [24], Canon [13], HONET [50], HIERAS [25], our constructions, what is lacking is an analytic framework that takes the first step towards a general consensus for the comparison of the existing, as well as the future, hierarchical designs against each other.

In this context, our work provides an analytical basis, by which an engineer can choose the underlying hierarchical network.*What makes superpeer systems better than homogeneous designs like ours, or more generally, one design better than another? And what does better mean?* These are the main questions that we seek to answer in this Chapter.

We answer these questions from the viewpoint of communication cost. More specifically, we provide an analytic framework to measure the communication cost of hierarchical DHTs. Also, we take the extreme designs on the spectrum to show that not necessarily a design is better than the other. In particular, we compare the superpeer architecture proposed by Garcés-Erice et. al. [22] against our framework for designing homogeneous hierarchical DHTs. We note that because our aim is to explore in which situations a particular hierarchical design is better, we do not lose generality by restricting attention to these two particular instantiations.

Summary of Results

In this Chapter, we make the following contributions:

i. We develop a cost-based model to assess the resources that a hierarchical DHT system has to contribute to efficiently support communication operations. Our formal model is novel in several aspects, among which, emphasizing locality in communication is the most important. It is obvious that the cost will be greatly influenced by the degree of locality, known as intra-cluster communication, that exits in the communication patterns of any specific peer-to-peer application.

ii. Using the proposed cost model, we compare the two main hierarchical DHT designs:

   - The homogeneous design, in which all nodes take the same duties and responsibilities for all operations; against
   - The superpeer design, in which a small set of peers, usually referred to as superpeers, process and relay queries on behalf of regular peers.

We believe that our framework is useful in determining which type of topology is appropriate for a specific P2P application, and in identifying pitfalls in existing hierarchical designs that are mainly driven by the desire to exploit the heterogeneity of peers to their advantage.

The rest of the paper is structured as follows:

- In §4.2, we discuss related work.
- In §4.3, we present the designs we analyze in this Chapter.
- In §4.4, we examine the impact that locality has upon routing performance.
- In §4.5, we evaluate the hierarchical designs we have introduced in §4.3.
- Finally, we draw some conclusions in §4.6.

4.2 Related Work

Despite the growing interest in hierarchical DHTs, as far as we know, our work is the first attempt at providing a formal analysis of the two main hierarchical DHT designs: the *homogeneous design* [13, 24] against the *superpeer design* [22, 25, 50]. However, recent work has extensively examined superpeer architectures, shedding light on their performance trade-offs, their potential drawbacks and reliability. They are described below.

S. Zoels et. al. [51] proposed a cost model to analyze when a superpeer architecture is better than a flat architecture. They found the existence of a natural trade-off between minimizing total cost and minimizing the cost for the highest loaded peer in the network. However, their study was tailored to a particular hierarchical architecture. They did not provide an analytical framework to compare existing hierarchical DHT designs.

A more practical analysis was performed by Yang and Garcia-Molina [52]. They considered, in addition to performance trade-offs, redundancy and topology variations in superpeer design. They were able to extract a few rules of thumb, even though they did not contemplate the homogeneous alternative.

For flat DHTs, Christin and Chuang [53] proposed a cost model to evaluate the resources that each node has to contribute for participating in the network. They investigated some of the most representative topologies, and concluded that from the point of view of reliability and scalability, all the geometries may create large imbalances in the load imposed on different nodes. We argue that such a model was a good starting point for our analytic framework. It helped us to formalize the forwarding traffic and characterizing the efficiency of a system as whole. We shared their aim, but for hierarchical designs.

In the following, we review some relevant works that showed the advantages of hierarchical designs.

Ganesan et. al. [13] proposed Canon, a technique to construct hierarchical DHTs from their flat counterparts, so that the homogeneity of load and functionality offered by the flat designs can be incorporated to the hierarchical versions. They mostly adduced better fault isolation, more effective bandwidth utilization, and better adaptation to the physical Internet as the main arguments to use hierarchical DHT systems.

Complementing this work, Garcés-Erice et. al. [22] studied the potential benefits of structuring peers into two layers: a superlayer, where the superpeers (the nodes with relatively long lifetime and large capacities) behave like proxies for the peers in the "regular" layer. The authors showed that superpeer systems scale better because queries are mainly processed by superpeers, which actually constitute the "backbone" of the network.

4.3 Hierarchical Architectures

In this section, we describe the two architectures we compare in this work. Since we instantiate both designs with Chord, we start with a brief description of Chord followed by our assumptions.

Chord

Although Chord [4] has been extensively described in Chapter 2, we provide a brief description to refresh it here.

Chord maps keys to nodes by means of *consistent hashing* [35], which has some desirable properties. Consistent hashing assigns an D-bit identifier to both nodes and objects using a collision-resistant hash function such as SHA-1. Then, objects are mapped to nodes as follows. Identifiers are ordered on an identifier circle modulo 2^D, labeled from 0 to $2^D - 1$. An object with identifier k (also know as *key*) is assigned to the first node, called the successor of this key, whose identifier follows (or is equal to) k in the identifier space (i.e., the first node going clockwise from k).

To accelerate searches, Chord maintains logarithmic routing information. Each node u maintains a routing table with up to n entries called *finger table*. The i^{th} entry in the table contains the id of the first node v that succeeds u by at least 2^i on the identifier ring, where $0 \leq i < D$.

The *standard* Chord search protocol works as follows: upon receiving a message for a key k, a node u forwards the query to the furthest finger whose id precedes most immediately (or is equal to) key k. By $\mathcal{O}(\log N)$ forwardings, the message reaches the *destination node*. Hence, routing is clockwise and *greedy*, never overshooting the destination.

Assumptions

For the remainder of this work, we make use of the following assumption.

Definition 42 (All-exist-all-live assumption). *Given that the identifier space under consideration is $\mathcal{I} = \{0, 1, \ldots, 2^D - 1\}$, all-exist-all-live assumption determines that each position in $\mathcal{I}$ is occupied by a live node.*

As we look for comparative results under the same conditions, we observe that this assumption is clearly acceptable. Moreover, it prevents our results to be biased against the nodes with the longest arcs. In fact, this bias is only significant when providing a worst-case analysis. However, for an average-case analysis, this simplification is acceptable and very helpful to ease exposition.

For the sake of simplicity, both architectures are restricted to two layers. It has been discussed in [25] that two is a pragmatic choice for the number of layers in a hierarchy. Of course, to make a fair comparison, we suppose that both architectures have the same number of nodes N, the same number of peers n in a cluster, and the same number of clusters K, where $N = Kn$. Under

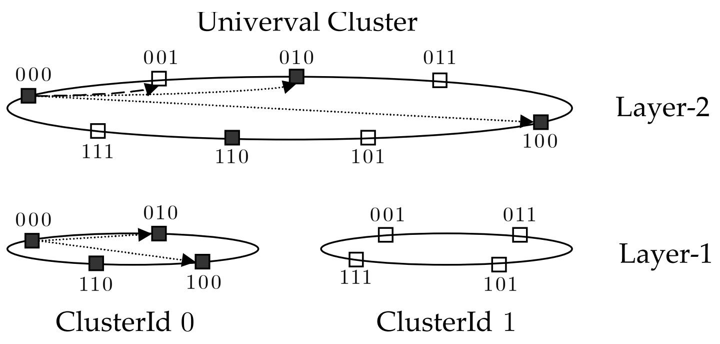


Figure 4.1: *Two-tier Whirl architecture.*

all-exist-all-live assumption, the above requirements force N to be 2^D. Hence, n must be equal to 2^d for some positive integer $d < D$, and $K = 2^{D-d}$. Also, we assume that and each cluster has a unique identifier.

4.3.1 Homogeneous Design

Among the existing homogeneous designs [13, 24], we selected our design for one basic reason: to identify in which characteristics our design is superior to superpeer systems. Since an extensive discussion of our construction was provided in Chapter 3, we only revisit the components needed to easily follow exposition in this Chapter.

Under all-exist-all-live assumption, our construction of Chord, called Whirl, can be defined as follows.

Definition 43 (Whirl). *Consider a directed graph on 2^D nodes distributed on a circle. Nodes are labeled with D-bit identifiers from 0 through $2^D - 1$ going clockwise. At tier-1, all nodes are organized into 2^{D-d}, $0 \leq d < D$, disjoint clusters, which together form a unique cluster at tier-2. All nodes participate to the two tiers simultaneously, and use the $(D-d)$ rightmost bits of their $nodeIds$ to identify their cluster. That is, all nodes within the same cluster share the $(D-d)$ rightmost bits of their $nodeIds$.*

Each node establishes the following links:

i. At tier-1, a link (u, v) exists between two nodes u and v in the same cluster iff $d_{clockwise}(u, v) = 2^i$ on the circle for all $i \in \{(D-d), (D-d)+1, \ldots, D-1\}$.

ii. At tier-2, a link (u, v) exists iff u and v are 2^i positions apart on the circle for all $i < (D-d)$. In this case, nodes u and v come from distinct tier-1 clusters.

According to the above definition, Whirl can be seen as a ever-rising sequence of regular interleaved polygons with corners uniformly distributed along the identifier circle $\{0, 1, \ldots, 2^D - 1\}$. Technically speaking, given a tier-1 cluster and a node u within that cluster, the *immediate* successor of u in its tier-1 cluster will be at least clockwise distance 2^d away from u on the circle. Fig. 4.1 shows a two-tier Whirl architecture for two clusters ($D - d$ = 1).

From the perspective of shortest paths, the definition of Whirl is deceptively simple; it hides a rich combinatorial structure, some of which we will unearth throughout this chapter.

Routing

Routing in Whirl is identical to routing in Chord, i.e., clockwise GREEDY routing. Throughout this Chapter, a source node will first attempt to route on its tier-1 cluster, and only if the node that is responsible for the key is not reached, the query will be routed through the global cluster.

Advantages and Disadvantages

As signaled in [13], the primary benefit of the homogeneous design is the uniform distribution of load among all nodes in the network, which also ensures that there is no a single point of failure. We illustrate this aspect through a brief discussion.

If one views a cluster as a single node, it is evident that a cluster can only be disconnected if the clusters to which it is connected are faulty. Otherwise, any message will have the chance to leave it, albeit this is done through a (suboptimal) routing path. To provide a better understanding of this, we provide the following simple Lemma.

Lemma 44. *Let each node fail with probability p in a time period of length $\lambda(p)$, with probability at least $\varepsilon = 1 - 1/N^k$ the following statement is true. If all clusters have size at least $n > \ln\left(\frac{1}{N}\ln\left(\frac{1}{\varepsilon}\right)\right)\frac{1}{\ln p}$, then no cluster is faulty.*

Proof. Recall that for a cluster to fail all peers in that cluster must be faulty. The probability that this event happens is p^n. Since the total number of clusters is $K = \frac{N}{n}$, we have that the probability that no cluster is faulty is given by $(1-p^n)^{N/n} > (1-p^n)^N \approx e^{(-p^n)N}$ (N is supposed to be large). Picking $n > \ln\left(\frac{1}{N}\ln\left(\frac{1}{\varepsilon}\right)\right)\frac{1}{\ln p}$, this probability is more than ε and the lemma follows. □

However, this design has a major drawback. Transient and low-capacity peers can seriously compromise the scalability of the whole system. For example, if a low-capacity peer stores a popular file, it will be rapidly overwhelmed, becoming a bottleneck. To remedy this, there are some solutions such as *proactive caching* [54] that can compensate the disadvantages of a homogeneous treatment of all peers.

4.3.2 Superpeer Design

In this section, we describe the superpeer design selected for our comparison. This design is the superpeer architecture proposed by Garcés-Erice et. al. [22]. We extensively describe it in Chapter 2. Hence, in this section, we only revisit the main parts.

Architecture

In this design, peers are divided into two layers: the superlayer and the regular-layer. Each peer in the superlayer, or the layer-2 network, is called *supeerpeer*, and is responsible for propagating the queries on behalf of the *regular-peers* in its cluster. For this purpose, each superpeer is connected to other superpeers in the superlayer, according to the standard Chord rule for creating links.

More specifically, all clusters are organized into a Chord overlay network defined by a directed graph $(\mathcal{C}, E)$, where $C = \{C_1, C_2, \ldots, C_K\}$ is the set of all clusters and E denotes the set of edges between the clusters in $\mathcal{C}$. If s_i is a superpeer in cluster C_i, and (C_i, C_j) is an edge in the superlayer

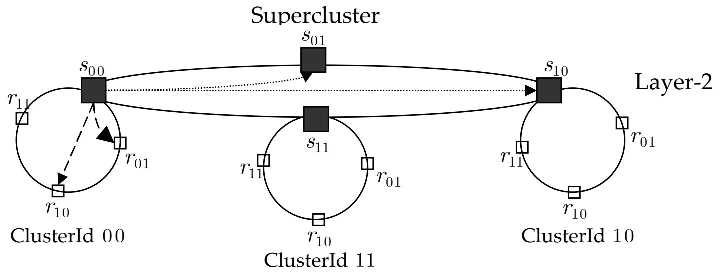


Figure 4.2: *Two-tier superpeer Chord-like architecture.*

overlay, $(\mathcal{C}, E)$, then s_i knows the IP address of superpeer s_j in cluster C_j. With this knowledge, s_i can directly forward queries to s_j.

It is important to note here that we assume that each cluster has exactly one superpeer, though in [22] there is no such a restriction. However, this assumption is reasonable because we consider that all superpeers are always up and never leave the network. In other words, we are optimistic with respect to the performance of this design.

A peer in the regular-layer, or equivalently in the layer-1, is called *regular-peer*. A regular-peer is characterized because it only keeps connections to other regular-peers in its cluster. Within each cluster, there is also a Chord overlay network that is used for routing queries in that cluster. Fig. 4.2 depicts this architecture with a single superpeer per cluster.

Routing

Essentially, the routing protocol exploits the multi-layer structure: first, the routing protocol looks for the cluster that stores a given key; next, it looks for the manager of this key within that cluster.

More precisely, consider a peer p looking for a key k not stored in its cluster. Then, routing can be described in the following way:

1. First, p forwards the query to the superpeer in its cluster, using standard Chord GREEDY routing.

2. Upon the reception of the query, the superpeer routes the query across the superlayer overlay $(\mathcal{C}, E)$ until the destination cluster is reached. Since $(\mathcal{C}, E)$ is a Chord overlay, the destination cluster is the cluster whose superpeer s has the closest id to the key. Let $F_{sup}(C)$ be the superpeer of cluster C. Formally, the destination cluster, C_{dest}, is

$$C_{dest} = \arg\min_{C_i \in \mathcal{C}} d_{clockwise}(F_{sup}(C_i), k)$$

   Observe that during this phase, the query only passes through superpeers. This is the major strength of this architecture.

3. Finally, using the Chord overlay network in cluster C_{dest}, superpeer $F_{sup}(C_{dest})$ routes the query to the peer responsible for k.

Advantages and Disadvantages

The obvious benefit of superpeer systems is the exploitation of heterogeneous peers, by assigning responsibility in proportion to the capacities of peers. By designating as superpeers the peers that are ”up” the most, the superlayer overlay will be more stable, thereby letting the system approach its optimal performance.

Disadvantages of superpeer systems include the potentially high traffic rates on inter-cluster links, and the diminished fault-tolerance due to the special role played by superpeers. However, there is another factor that most notably limits the feasibility of superpeer systems. This factor is the use of an effective management protocol that promotes the long-lifetime and large-capacity peers as superpeers when needed [55]. Since no complete knowledge is available in a decentralized system, it is difficult to determine what values are ”long” or ”large” enough in relation to the peers currently present in the system. We provide a large discussion on this topic in [56].

4.4 Locality Analysis

Previous works [13, 22, 51] that evaluate hierarchical DHT systems suppose that communication is *uniform*, that is, all nodes have an equal probability of serving a request. While such an assumption is realistic for flat DHTs, hierarchical DHTs exploit the locality that exists in communication patterns to their benefit, for example, by allowing nodes to specify in which clusters the content is made accessible or stored [13]. In other words, we drop the assumption that a node communicates with any other node with equal probability, assuming that nodes in a cluster communicate together with a higher (or lower) probability than do two nodes from two different clusters.

Let γ be the probability that both the source and the destination nodes of a request are in the same cluster. Hence, $(1-\gamma)$ denotes the probability of inter-cluster communication. That is, the smaller the value of γ, the more likely the anti-locality in communication. Further, we assume that:

- *Intra-cluster communication* is distributed uniformly at random over the set of nodes within each cluster. This means that the source peer will route an intra-cluster query to each intra-cluster peer with equal probability; and

- *Inter-cluster communication* is uniformly random, that is, a source forwards an inter-cluster query to each other cluster and to each node in the target cluster with equal probability.

In what follows, we characterize the performance measures to evaluate both architectures as a function of γ. Observe that it is difficult to predict what values of γ an architect may expect in practice. We remark that although the results of the following sections suggest that no design is ”universally” better, it is clear that if inter-cluster messages are frequent, it is more beneficial to use a hierarchical design that requires higher maintenance but saves significantly on requests.

First, we begin by clarifying the impact that locality has upon the average path length, denoted by μ (in terms of number of hops). The expected value of this quantity can be viewed as simple metric capturing the overall routing cost suffered by each node.

4.4.1 Homogeneous Design

We derive the average path length, μ_{HD}, for the homogeneous design. Then,

$$\mu_{HD} = \gamma(\mu_{l1}) + (1-\gamma)(\mu_{l2}) \tag{4.1}$$

where μ_{l1} is the average number of hops between two nodes within a cluster at tier-1, and μ_{l2} is the average path length of the global cluster (when both nodes belong to distinct tier-1 clusters).

To compute μ_{l1}, we exploit the fact that under *all-exist-all-alive* assumption, a Chord network embeds a hypercube. Thus, if a message is for a node that is clockwise distance η away, routing is equivalent to performing left-to-right bit fixing to convert the 1s in the binary representation of η to 0s. That is, if η is 9 (1001 in binary), Chord routing uses the jumps of size 8 and 1 in that order, fixing the leftmost 1 in the remaining distance at each step. Thus, μ_{l1} can be calculated as follows,

$$\mu_{l1} = \frac{\sum_{i=0}^{d}\binom{d}{i}i}{2^d} = \frac{d}{2} \tag{4.2}$$

Once we compute μ_{l2}, we shall be done. To calculate μ_{l2}, the first constraint to be imposed is that the source and the target clusters are distinct. This implies the existence of at least 1 inverted bit in the rightmost $(D-d)$ bits of the source and target $nodeIds$. Now, let η be the clockwise distance between the source and target nodes. Then, it is easy to see that $\eta = i + j$, where distance i $(i > 0)$, is contributed by the $(D-d)$ rightmost bits of η and the distance j $(j > 0)$ is contributed by the d rightmost bits of η. Consequently, the number of nodes at clockwise distance η is given by $\binom{D-d}{i}\binom{d}{j}$. Then, μ_{l2} is computed from

$$\mu_{l2} = \frac{\sum_{i=1}^{D-d}\sum_{j=0}^{d}\binom{D-d}{i}\binom{d}{j}(i+j)}{2^D - 2^d} = \frac{D2^{D-1} - d2^{d-1}}{2^D - 2^d} \tag{4.3}$$

Equations (4.1), (4.2), and (4.3) then give

$$\mu_{HD} = \gamma\left[\frac{d}{2}\right] + (1-\gamma)\left[\frac{D2^{D-1} - d2^{d-1}}{2^D - 2^d}\right]$$

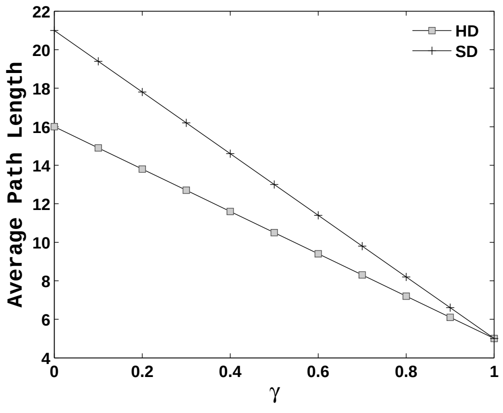
It is worth noting that when $\gamma = 2^{d-D}$ (i.e., the probability of intra-cluster communication is proportional to n), $\mu_{HD} = \frac{D}{2}$ as expected.

4.4.2 Superpeer Design

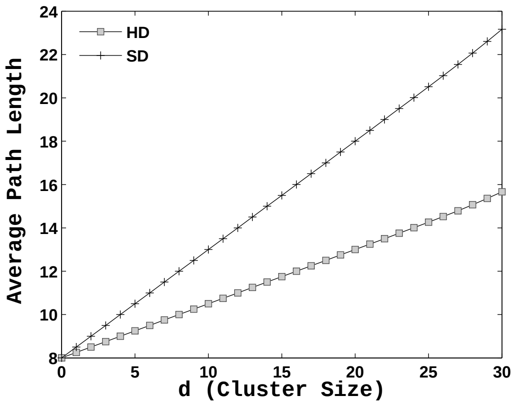
We now compute the average path length, μ_{SD}, of the superpeer design as a function of γ. Clearly, μ_{SD} is given by:

$$\mu_{SD} = \gamma(\mu_{l1}) + (1-\gamma)(2\mu_{l1} + \mu_{sl})$$

where μ_{l1} is the average path length of the layer-1 overlay, and μ_{sl} is the average path length of the superlayer overlay (notice that both the source and the destination node belong to distinct clusters). We have already shown that $\mu_{l1} = d/2$. Using reasoning analogous to that in Eq.(4.3), it is easy to see that



a)



b)

Figure 4.3: *Comparison between homogeneous (HD) and superpeer (SD) designs' average path length for* $D = 32$ *and* $d = 10$*. The value of* γ *is varied from* 0 *to* 1 *(a).* γ *is fixed at* 0.5 *and* d *is varied from* 1 *to* 32 *(b).*

$$\mu_{sl} = \frac{\sum_{i=1}^{D-d} \binom{D-d}{i} i}{2^{D-d} - 1} = \frac{(D-d)2^{D-d-1}}{2^{D-d} - 1} \tag{4.4}$$

Therefore, combining equations (4.4) and (4.2), μ_{SD} is given by

$$\mu_{SD} = \gamma \left[\frac{d}{2}\right] + (1-\gamma)\left[\frac{(D+d)2^{D-1} - d2^d}{2^D - 2^d}\right]$$

Again, when $\gamma = 2^{d-D}$, μ_{HD} equals to $\frac{D}{2}$ or that of Chord.

4.4.3 Discussion of Results

A comparison of the average path length for the two designs is plotted in Figures 4.3a-4.3b. Both instances have network sizes $n = 2^{10}$ and $N = 2^{32}$. We make the following observations.

- In Fig. 4.3a, the value of γ is varied from 0 to 1. As γ approaches 1, both designs have similar performance, as inter-cluster links are rarely used. However, as γ tends to 0, the superpeer design increasingly outperforms the homogeneous design. It must be noted that this is done at the price of increasing the required capacities of superpeers. This claim will be clarified in the next section.

- As can be seen in Fig. 4.3b, the average path length in both designs rises when cluster size approaches N (γ is fixed at 0.5). Our results indicate that is not recommendable to create large clusters. With a few thousands of nodes in each cluster, the average path length of the superpeer network is 40% higher than the path length of the homogeneous network. Thus, if query traffic dominates maintenance traffic, then hierarchical overlays with large clusters will perform poorly. Most of the traffic will cross the already congested inter-cluster links, thereby incurring intolerable network delays.

4.5 Cost-Based Analysis

In this section, we describe our cost framework for hierarchical DHTs. To this purpose, we start with a formal description of the metrics and the workload. We define workload as follows.

Definition 45. *Denote by l the average lifetime of a node. Denote by f the average number of queries each node processes per second. Then, we define a workload w by the quadruplet $w :=< N, n, l, f >$, where N is the number of nodes in the system, and n the cluster size.*

Our typical workload assumes that node lifetime L follows an exponential distribution:

$$\Pr[L = t] = \lambda_l e^{-\lambda_l t},$$

where $\lambda_l = \frac{1}{l}$. We take a node lifetime similar to the one encountered in real P2P systems. For instance, the average node lifetime in Gnutella is 2.9 hours [57]. We also assume that nodes generate traffic independently of each other, following a Poisson process with a medium to high mean rate $f = 0.1 \sim 10$ queries/second (Note that a node itself can represent a gateway in a large enterprise network). In addition, we treat node arrival as a Poisson process with rate λ_a. To maintain a stable population of N nodes at all times, we set $\lambda_a = N\lambda_l = \frac{N}{l}$.

Next, we define the individual cost as follows.

Definition 46. *Let $\mathcal{D}$ be a hierarchical design. Let $\mathcal{I}_\mathcal{D}$ be a concrete realization of $\mathcal{D}$ with vertex set $V(I_\mathcal{D})$. Given a workload quadruplet $w :=< N, n, l, f >$, the individual cost imposed on a peer p in $V(I_\mathcal{D})$ is given by $C_{I_\mathcal{D}}(p, w) = C_{I_\mathcal{D},m}(p, w) + C_{I_\mathcal{D},r}(p, w)$, where $C_{I_\mathcal{D},m}(p, w)$ and $C_{I_\mathcal{D},r}(p, w)$ denote the maintenance and routing costs imposed by $I_\mathcal{D}$ on p under workload w, respectively.*

We now define total cost as follows.

Definition 47. *Let $\mathcal{D}$ be a hierarchical design. Let $\mathcal{I}_\mathcal{D}$ be a concrete realization of $\mathcal{D}$ with vertex set $V(I_\mathcal{D})$. Given a workload quadruplet $w :=< N, n, l, f >$, we define the total cost $C_{I_\mathcal{D}}(w)$ of an instance $\mathcal{I}_\mathcal{D}$ as $C_{I_\mathcal{D}}(w) = C_{I_\mathcal{D},m}(w) + C_{I_{\mathcal{D},r}(w)}$, where $C_{I_\mathcal{D},m}(w)$ and $C_{I_\mathcal{D},r}(w)$ are the maintenance and routing costs of $I_\mathcal{D}$ under workload w, respectively. Clearly,*

$$C_{I_\mathcal{D}}(w) = \sum_{p \in V(I_\mathcal{D})} C_{I_\mathcal{D},m}(p, w) + \sum_{p \in V(I_\mathcal{D})} C_{I_\mathcal{D},r}(p, w)$$

To simplify exposition throughout this Chapter, we drop parameter w from all cost equations (as it is clearly implicit). Further, we assume that both query and maintenance messages have a size of b bytes, all messages are explicitly acknowledged and the acknowledgments have length $0.5b$. Below, we use the total cost to compare the two hierarchical designs against each other.

4.5.1 Homogeneous Design

We start with the maintenance costs of the homogeneous design.

Maintenance cost

In this section, we calculate the *minimum traffic* required for updating the routing tables in the face of node arrivals and departures. In an N-node homogeneous DHT with node lifetime l, on average $\frac{N}{l}$ nodes join and $\frac{N}{l}$ nodes leave each second. As our analysis uses Chord as a representative substrate, maintenance costs for the homogeneous design are strongly characterized by the costs of keeping up the Chord overlay structure in each cluster.

The main characteristic of the homogeneous design is that all nodes act equal roles and adopt the same responsibilities for all the operations. Under our workload model, this means that the traffic suffered by each node will be the same, irrespective of capacity constraints at each node. In our model, this property is advantageous for computing the cost, as the cost of the homogeneous design, C_{HD}, is equivalent to computing the total cost of a Chord network (note that requests are uniformly distributed over the set of nodes and all nodes have an equal number of fingers).

In what follows, we compute the maintenance cost for Chord. In Chord, maintenance traffic is generated by: 1) *heartbeat* messages, 2) *stabilization* messages and 3) *fixfinger* messages.

Heartbeat Cost. Heartbeat messages are sent *periodically* by each node to check if a finger is still alive. Assuming that 1) heartbeat messages are sent every T_{beat} seconds; 2) they have length $0.5b$, and 3) each node handles D fingers, the total traffic for the heartbeats is

$$C_{beat} = \frac{0.5bND}{T_{beat}}$$

Stabilization Cost. Chord specifications define a *stabilization* algorithm that each node executes periodically to verify if a joining node has inserted itself between a node and its successor. A detailed description of this procedure can be found in [4]. For convenience, we assume here that *stabilization* requires three messages of length b. If we consider that stabilization is run every T_{stab} seconds, the traffic generated is

$$C_{stab} = \frac{3bN}{T_{stab}}$$

Finger Updating Cost. Adding or removing a node is accomplished at a cost of $\mathcal{O}(\log^2 N)$ messages. By way of explanation, each node periodically invokes *fixfinger* to make sure that its fingers are correct; this is how existing nodes incorporate new nodes into their finger tables. In particular, fixing a finger requires looking up the finger's id, which costs $\mathcal{O}(\log N)$ messages on average. Because $\log N$ rounds of *fixfinger* are required to initialize a finger table, *fixfinger* cost is $\mathcal{O}(\log^2 N)$. For the time being, we make a further simplification by assuming that *fixfinger* acquaints each node that is affected by a topological change. In other words, *fixfinger* is run solely when a node joins or leaves the overlay. We take thus a conservative approach, contrariwise to Chord that needs periodic updates on all nodes, a simpler strategy that incurs a higher overhead. Consistent with this observation, the cost for *fixfinger* is presented below.

When a node joins, the number of existing nodes that need to update its finger table is $\mathcal{O}(logN)$ on average. In our case, this value is exactly D, since we supposed that the identifier space is fully populated. Since fixing each finger requires performing one lookup, a mean of $D(D/2)$ messages is required to update all the affected nodes in the face of a node arrival. Assuming these messages have unit size b and each is acknowledged by a packet of length $0.5b$, the total traffic needed for updating all the affected finger tables is

$$\frac{(1+0.5)bND^2}{2l}$$

The same cost is incurred when a node leaves. Specifically, the leaving node sends a message to each of its fingers to notify them about its *imminent* departure. Upon reception of this message, each notified finger immediately updates its routing table to continue offering logarithmic guarantees on routing. Altogether, this requires $\frac{ND^2}{2l}$ messages, which increases the maintenance cost by another amount of

$$\frac{(1+0.5)bND^2}{2l}$$

bytes. Moreover, a mean of $D^2/2$ messages are required to initialize the finger table of a new node. Therefore, the *fixfinger* traffic is

$$C_{fix,HD} = (1+0.5)b\left[\frac{N}{l}D + \frac{N}{l}D + \frac{N}{l}D\right]\frac{D}{2}.$$

Maintenance cost. The maintenance traffic (*fixfinger* + *heartbeat* + *stabilize*) is then given by

$$C_{m,HD} = \left(4.5\frac{D^2}{2l} + \frac{0.5D}{T_{beat}} + \frac{3}{T_{stab}}\right)bN.$$

Routing Cost

Once we compute the routing cost, we shall be done. With this formulae, we will be able to infer if it is economical to use the homogeneous design under a given workload. To compute the routing cost, we know that each node processes f queries per second. Hence, the total number of lookups processed by the system is Nf. Since each lookup takes μ_{HD} hops, then the total traffic dedicated for lookups is

$$C_{r,HD} = (1+0.5)bNf\mu_{HD}.$$

Total cost

Therefore, the total cost (*maintenance* + *routing*) is given by

$$C_{HD} = \left[4.5\frac{D^2}{2l} + 1.5f\mu_{HD} + \frac{0.5D}{T_{beat}} + \frac{3}{T_{stab}}\right]bN. \tag{4.5}$$

We can in turn use expression (4.5) to compute the individual cost that each overlay node p has to afford for being part of the overlay. Let $C_{HD}(p)$ denote the individual cost suffered by peer p. Then, $C_{HD}(p)$ can be computed directly by dividing the total cost, C_{HD}, by N. This is possible because all nodes have equal responsibilities from the system's viewpoint. Therefore,

$$C_{HD}(p) = \left[4.5\frac{D^2}{2l} + 1.5f\mu_{HD} + \frac{0.5D}{T_{beat}} + \frac{3}{T_{stab}}\right]b \qquad \text{for all } p.$$

4.5.2 Superpeer Design

As in the case of the homogeneous design, we start with the derivation of the maintenance cost. Next, we continue with the routing cost to finally provide an accurate closed-form expression for the total cost. The analysis here is more complex, since we need to distinguish between two types of peers: regular-peers and superpeers.

Maintenance cost

There are two classes of connections in superpeer networks: the connections between superpeers and the connections which have at least one regular-peer as an endpoint. The maintenance cost on any peer is directly related to the number and the stability of the peers in its finger table. This means that the maintenance cost is proportional to the number of fingers and inversely proportional to the average lifetime of them. For ease of explanation, we consider that supeerpeers are always up. So *fixfinger* traffic must be considered conservative in this design. Formally, this leads us to establish that total *fixfinger* traffic as (i.e., only links to regular-peers can fail)

$$C_{fix,SD} = (1+0.5)b\left[\frac{N}{l}d + \frac{N}{l}d + \frac{N}{l}d\right]\frac{d}{2}.$$

Consequently, the total maintenance cost $C_{m,SD}$ is

$$C_{m,SD} = \left(4.5\frac{d^2}{2l} + \frac{0.5D}{T_{beat}} + \frac{3}{T_{stab}}\right)bN. \tag{4.6}$$

One important observation about superpeer design is that individual maintenance costs $C_{m,SD}(p)$ suffered by each node p can be directly derived from $C_{m,SD}/N$.

Routing cost

Next, we derive the routing cost imposed on each superpeer. Let $C_{SD,r}(s)$ denote the individual routing cost that superpeer s processes each second. Then, it is easy to see that $C_{SD,r}(s)$ can be calculated as follows:

$$C_{SD,r}(s) = C_{SD,CL}(s) + C_{SD,NCL}(s)$$

where $C_{SD,CL}(s)$ is the routing cost at s due to intra-cluster traffic, and $C_{SD,NCL}(s)$ is the routing cost at s due to inter-cluster traffic.

In the derivation of $C_{SD,r}(s)$, note that each inter-cluster message traverses $\mu_{sl} = \left[\frac{(D-d)2^{D-d-1}}{2^{D-d}-1}\right]$ superpeers on average before reaching the destination cluster. Since each cluster injects these messages at rate $(1-\gamma)fn$ into the system, then

$$C_{SD,NCL}(s) = (1-\gamma)fn\,\mu_{sl}(1+0.5)b \qquad \text{for all } s,$$

while

$$C_{SD,CL}(s) = \gamma f\,\mu_{l1}(1+0.5)b \qquad \text{for all } s.$$

This completes the computation of $C_{SD,r}(s)$.

Cost on regular-peers. In inter-cluster routing, a query issued by a regular-peer r is first routed to the closest superpeer to r, and thence routed to the target cluster. This causes the regular-peers close to a superpeer to receive more inter-clusters queries. As a consequence, the individual cost imposed on regular-peers is not *unique* and depends on *how close each peer is to its superpeer*. This complicates the analysis, since now each regular-peer p has an associated routing cost, $C_{SD,r}(p)$, that depends on its relative position to the closest superpeer. More specifically, we can compute $C_{SD,r}(p)$ as

$$C_{SD,r}(p) = C_{SD,CL}(p) + C_{SD,NCL}(p)$$

where:

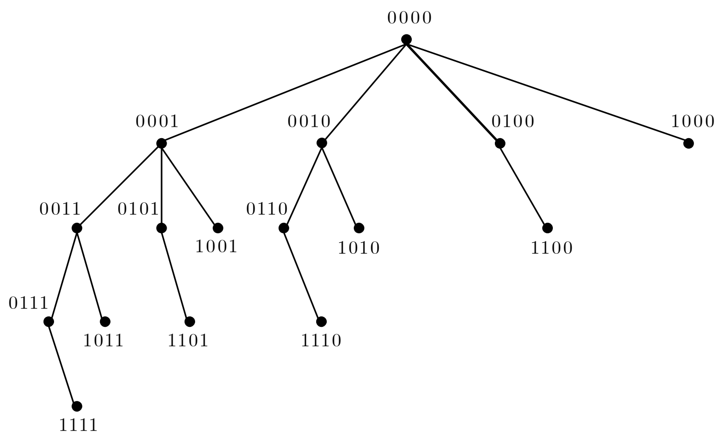


Figure 4.4: *Chord's spanning binomial tree (d = 4).*

- $C_{SD,CL}(p)$ denotes the routing load suffered by p due to intra-cluster traffic; and
- $C_{SD,NCL}(p)$ denotes the routing cost at p due to inter-cluster traffic.

Since the intra-cluster traffic cost is equally distributed over all nodes in the cluster, $C_{SD,CL}(p)$ is given by

$$C_{SD,CL}(p) = \gamma f(\mu_{l1})(1 + 0.5)b \qquad \text{for all } p.$$

However, $C_{SD,NCL}(p)$ is not the same for all regular-peers (as discussed above). To compute $C_{SD,NCL}(p)$, our analysis draws attention to the geometric intuition that Chord GREEDY routing resembles hypercube geometry. Denote by $\eta(i,j)$ the clockwise distance, $d_{clockwise}(i,j)$, between nodes i and j in binary form. Then, recall that GREEDY routing in Chord is effectively achieved by "correcting" the 1s in the binary representation of $\eta(i,j)$ to 0s. In a hypercube, this is equivalent to routing from node $\eta(i,j)$ to node 0. The basic difference between routing in Chord and in the hypercube is that the hypercube allow bits to be corrected in any order while on Chord bits have to be corrected from left-to-right.

The consequence of this way of routing is that the union of the clockwise-GREEDY paths from all nodes to one specific node spawns a *spanning binomial tree* [58] (as in hypercubes). One basic particularity of this tree is that is unique. By unique we mean the following. Let $T_{Chord}(u)$ be the spanning binomial tree rooted at node u. Then, for each node $v \neq u$, $T_{Chord}(v)$ is isomorphic to $T_{Chord}(u)$, which means that all the trees have the same structure and there is only one tree.

There is a particularly simple way of determining if two Chord trees $T_{Chord}(u)$ and $T_{Chord}(v)$ are isomorphic. First, assume that $T_{Chord}(u)$ and $T_{Chord}(v)$ have the same number of nodes and the same height (otherwise they are not isomorphic). Then, nodes can be grouped into *levels*, i.e., sets of nodes that are at the same distance from the root; since distance from the root is preserved by isomorphism, nodes in $T_{Chord}(u)$ must correspond to nodes in $T_{Chord}(u)$ at the same level.

This property allows us to simplify considerably the computation of $C_{SD,NCL}(p)$, as the cost on p depends only on the (clockwise) distance to its closest superpeer and not on the $nodeId$ of the latter. Using this property, we can avoid including the root node as parameter and expose our findings in a somewhat more general context. Notice that if we label each node by its clockwise distance to the superpeer, we obtain a unique tree, T_{Chord}, for all clusters. This tree corresponds to the spanning binomial tree rooted at vertex 0 of a d-dimensional binary hypercube. In Fig. 4.4, we illustrate T_{Chord} for a Chord cluster of 16 nodes.

Formulation of T_{Chord}

Formally, let $V(T_{Chord})$ be the set of nodes in T_{Chord}. T_{Chord} can be then uniquely defined using function $children : V(T_{Chord}) \longrightarrow 2^{V(T_{Chord})}$, which returns the children of node p in T_{Chord}. The children of p are obtained by complementing one of the leading zeros in the binary representation of p.

Analogously, T_{Chord} can be uniquely defined using function $parent : V(T_{Chord}) \longrightarrow V(T_{Chord})$, which returns for a node p, the parent of p in T_{Chord}. Specifically, the parent of p is obtained by complementing the leftmost 1-bit in binary representation of p. We observe that this operation corresponds to performing one Chord hop towards the root of T_{Chord}.

By the definition of these two functions, it is easy to see that T_{Chord} has an optimal height d, an optimal average height equal to $d/2$, and more importantly, it is highly unbalanced. By a simple inspection of Fig. 4.4, it is easy to see that the subtrees of the root have sizes 2^0, 2^1, ... and 2^{d-1}, which corresponds to a poorly balanced tree. This imbalance is precisely what causes each node to present a particular cost depending on its position in the tree.

Definition 48. *Assume a T_{Chord} tree of size 2^d. Denote by $\eta = \eta_{d-1}\eta_{d-2}...\eta_0$ the binary representation of the clockwise distance between the root and an arbitrary node other than the root. Let i be such that $\eta_i = 1$ and $\eta_j = 0, \forall j \in \{i+1, i+2, ..., d-1\} \equiv \mathcal{L}^{\mathcal{SBT}}(\eta)$ and let $i = -1$ if $\eta = 0$. Clearly, $\mathcal{L}^{\mathcal{SBT}}(\eta)$ is the set of leading zeros of η. Then, we have that*

$$children(\eta) = \left\{\eta_{d-1}\eta_{d-2}\ldots\bar{\eta}_j\ldots\eta_0 \mid j \in \mathcal{L}^{\mathcal{SBT}}(\eta)\right\}.$$

The next lemma provides the number of times a node is seen on the clockwise-GREEDY paths from all nodes to one specific node in Chord. To the best of our knowledge, we are the first to claim it.

Lemma 49. *Assume a T_{Chord} tree of size 2^d. Assume that each node is represented by its clockwise distance to the root. Now let $\eta = \eta_{d-1}\eta_{d-2}...\eta_0$ be the binary representation of the clockwise distance between node η and the root, such that $\eta_i = 1$ and $\eta_j = 0, \forall j \in \{i+1, i+2, \ldots, d-1\} \equiv \mathcal{L}^{\mathcal{SBT}}(\eta)$. Then, the number of nodes in the subtree induced by η (including node η itself) is exactly 2^{d-i-1}.*

Proof. In T_{Chord}, the children of a node η are obtained by complementing one of the leading 0s in the binary representation of η. Let i be such that $\eta_i = 1$ and $\eta_j = 0, \forall j \in \{i+1, i+2, \ldots, d-1\} \equiv \mathcal{L}^{\mathcal{SBT}}(\eta)$, which implies that each child of node η is formed by drawing exactly one element of $\mathcal{L}^{\mathcal{SBT}}(\eta)$. The possibilities to draw one element of a set of size $(d-i-1)$ is $C_1^{d-i-1} = \binom{d-i-1}{1}$. This means that there are exactly C_1^{d-i-1} other nodes that route through η. By a similar argument, it is easy to see that $|children^k(\eta)| = C_k^{d-i-1}$, $k > 0$, where $children^k(\eta)$ denotes the children of η that results after applying k times children function to η. Clearly, the number of nodes that lie on the subtree rooted at η is then

$$1 + \sum_{k=1}^{d-i-1} \binom{d-i-1}{k} = 2^{d-i-1},$$

so the lemma follows. □

With this Lemma, we are now in position to derive $C_{SD,NCL}(p)$ and conclude the computation of the individual routing cost.

Let $\eta(p,s) = \eta_{d-1}\eta_{d-2}\ldots\eta_0$ be the clockwise distance from node p to superpeer s expressed in binary form. In addition, let i be the position of the leftmost 1-bit in $\eta(p,s)$, i.e., the index i such that $\eta_i = 1$ and $\eta_j = 0, \forall j \in \{i+1, i+2, \ldots, d-1 \equiv \mathcal{L}^{\mathcal{SBT}}\}(\eta)$. Note that the total number of inter-cluster messages a cluster generates and receives per second equals to $2(1-\gamma)fn$. Then, $C_{SD,NCL}(p)$ can be calculated as follows:

$$C_{SD,NCL}(p) = (1-\gamma)fn(1+0.5)b(R_{p,OUT} + R_{p,IN})$$

where $R_{p,OUT}$ (resp., $R_{p,IN}$) denotes the probability that an inter-cluster message gets routed through node p on its way out of (resp., into) the cluster.

Now, we derive the expression for $R_{p,OUT}$. One way to do so yields the result we seek. Say first that the set of source nodes $\mathcal{S}(p)$ for which p acts as an intermediate hop (including himself) is the set of nodes that are in the subtree rooted at p. By Lemma 49, $|\mathcal{S}(p)| = 2^{d-i-1}$, since i is the position of the leftmost 1-bit in $\eta(p,s)$. Hence, $R_{p,OUT}$ is given by (recall that $n = 2^d$):

$$R_{p,OUT} = \frac{2^{d-i-1}-1}{n} = \frac{2^{d-i-1}-1}{2^d} = \frac{1}{2^{i+1}} - \frac{1}{2^d}.$$

By a similar argument, it can be shown that

$$R_{p,IN} = \frac{2^k-1}{n} = \frac{2^k-1}{2^d} = \frac{1}{2^{d-k}} - \frac{1}{2^d}. \tag{4.7}$$

where k is the position of the rightmost 1-bit of $\eta(s,p) = \eta_{d-1}\eta_{d-2}\ldots\eta_0$, i.e., k is such that $\eta_k = 1$ and $\eta_j = 0, \forall j \in \{k-1, k-2, \ldots, 0\} \equiv \mathcal{T}^{\mathcal{SBT}}(\eta(s,p))$, with $k = d$ if $\eta(s,p) = 0$. $\mathcal{T}^{\mathcal{SBT}}(\eta(s,p))$ denotes the set of trailing zeros of $\eta(s,p)$. This completes the derivation of $C_{SD,NCL}(p)$. Thus, we have that

$$C_{SD,r}(p) = 1.5bf\left[\gamma\mu_{l1} + (1-\gamma)(2^{d-i-1} + 2^k - 2)\right].$$

One important point to be noted here is that in general, $R_{p,OUT} \neq R_{p,IN}$. More precisely, recall that $\eta(i,j) = (j-i+n)\ mod\ n$ is the clockwise distance from a node i to a node j. Then, one can prove without much difficulty that $R_{p,OUT} = R_{p,IN}$ if and only if $d-i-1 = k$, where $(d-i-1)$ is the position of the leftmost 1-bit in $\eta(p,s)$ and k is the position of the rightmost 1-bit in $\eta(s,p)$, respectively.

Using Eq.(4.7), we show the PMF (probability mass function) of $R_{p,IN}$ in Fig. 4.5a for $n = 256$. As can be seen in the figure, a small subset of nodes (we have excluded superpeers) receive a lot of traffic. As in the case of homogeneous designs, this plot provides an important insight, commonly obscured in the literature: the performance of a superpeer system could be also conditioned by the capacities of weak nodes, thus questioning one of the key benefits of this design. For instance, while for a peer p that is at clockwise distance 1 away of its superpeer, $R_{p,IN} = 0$, the fraction of inter-cluster queries that receives a node p'at clockwise distance 128 away is $R_{p',IN} = \frac{1}{2}$.

Total cost

To complete the picture, we next derive the total cost, C_{SD}, for the superpeer design. In this case, observant readers will realize that this can be done using the individual costs experienced by each node (as in Definition 47). More precisely, we compute C_{SD} as follows

$$C_{SD} = C_{m,SD} + K \sum_{p=0}^{n-1} C_{SD,r}(p) + \sum_{s=0}^{K-1} C_{SD,r}(s)\mu_{sl}.$$

Since each superpeer manages an equal number of regular-peers, C_{SD} is given by the following expression (recall that K is the number of cluster and n the cluster size):

$$C_{SD} = C_{m,SD} + K \left(\sum_{p=0}^{n-1} C_{SD,r}(p) + C_{SD,r}(0)\mu_{sl} \right). \tag{4.8}$$

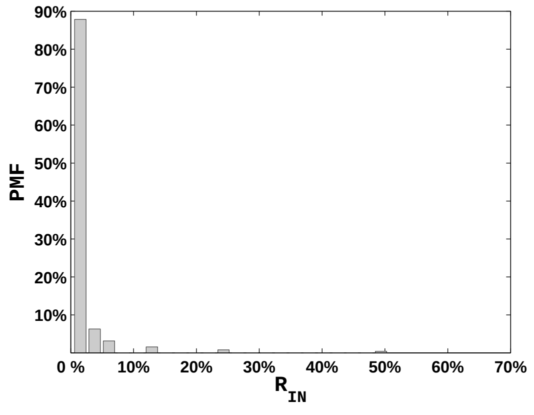
Recall that the cost $C_{SD,r}(s)$ is proportional to the number of regular-peers s manages, which is n. We leave the expansion of Eq.(4.8) to the reader.

4.5.3 Discussion of Results

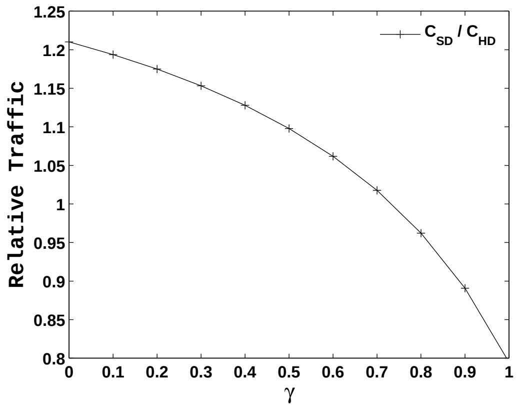
In this section, we compare the traffic costs incurred by each design. For our discussion, we set $D = 32$ and $d = 10$. Unfortunately, there exists some controversy in the literature [51, 52, 57] about the optimal ratio of regular-peers to the number of superpeers. This is of great importance, however, we do not wish to involve ourselves in this issue here. Rather, what we want is to provide a comparative with a workload presumably equivalent to the one found in real P2P systems. Unlike otherwise noted, we assume that $l = 2.9$ hours (Gnutella) and $f = 0.1$ queries/second. As a result, the workload quadruplet for the comparison conducted on this section corresponds to $< 32, 10, 2.9, 360 >$. In addition, T_{beat} and T_{stab} are set to 30 seconds.

In Fig. 4.5b, we compare the total cost of the homogeneous design with the cost of the superpeer design. We use the relative traffic cost to illustrate the traffic savings of one design to that of the reference. Here, we define the total relative traffic as C_{SD}/C_{HD}, where C_{SD} refers to the total traffic of the superpeer design and C_{HD} is the total cost of the homogeneous design. The value of γ is varied from 0 to 1. For communication patterns with up to 70% of locality, the maintenance costs exhibited by the homogeneous design are rightly compensated by the savings obtained from the greater efficiency in routing. In contrast, the superpeer design incurs higher load in this range. However, as γ approaches 1, the homogeneous design introduces more traffic than the superpeer design. The reason for this is easy to explain. As γ approaches 1, inter-cluster links are used less, and the maintenance summand emerges as the "dominant" factor in equations (4.5) and (4.8). As a result, the homogeneous design cannot be recommended for distributed applications that anticipate strong locality on its communications patterns.

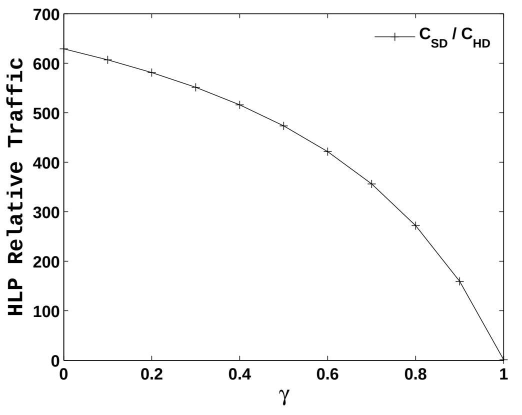
Bearing in mind the special role played by superpeers, Fig. 4.5c and Fig. 4.5d plot the relative traffic experienced by the highest loaded peer (HLP) in each design. In Fig. 4.5c, we vary the amount of intra-cluster communication. In Fig. 4.5d, we vary the average node lifetime ($\gamma = 0.5$) to investigate the impact of churn, that is, the continuous process of node arrivals and departures. Recall that to maintain a stable population, we set the node arrival rate λ_a to N/l. Consequently, if l is of the order of a few seconds, then a sufficient number of nodes will join the network each second to compensate leaving peers. The motivation behind these figures obeys to our conviction that the routing traffic imposed on weak superpeers can limit the performance of a hierarchical system as a whole. In particular, we compare the traffic imposed on *any* superpeer with the traffic found at *any* homogeneous peer. As a rule of thumb, Fig. 4.5c shows that the traffic experienced



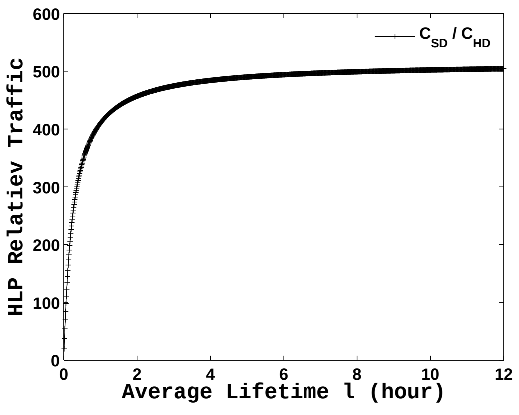
a)



b)



c)



d)

Figure 4.5: *Traffic comparison between homogeneous and superpeer designs for D = 32 and d = 10. (a) illustrates PMF of $R_{p,IN}$. The relative total cost, namely C_{SD}/C_{HD}, is shown in (b) when γ is varied from* 0 *to* 1*. Similarly, we plot the relative cost between the highest loaded peers (HLPs) of both designs. (c) Varies locality γ. (d) Varies node lifetime (γ is set to* 0.5*).*

by a superpeer is 600 times greater than the traffic suffered by a homogeneous peer when $\gamma = 0$ (reflecting a situation of strong "anti-locality"), which provides a practical finding: *depending on the communications patterns, superpeer design, on the contrary to what one can a priory conjecture, does not always enable a system to reach its optimal performance.*

Fig. 4.5d shows that the positive results obtained by the homogeneous design in the preceding evaluation do not longer hold. In fact, this figure confirms the well-known fact that if the average node lifetime l is low (in our case, when $l < 1$ hour), the cost of maintaining a network of transient peers does not compensate a major efficacy in routing. However, we note that this result is rather pessimistic; increased realism would be supposing that superpeers join and leave the network systematically, which requires dropping our assumption that superpeers stay forever.

In summary, while very appealing from the point of view of resilience and scalability, superpeer designs (such as the one in [22]) do not always constitute the best alternative. For instance, they can potentially create large imbalances on the load imposed on regular nodes. Analogously, homogeneous designs cannot also be considered the best solution, since they perform poorly if the nodes participating in the system are too much transient. This leads to the conclusion that does not exist a ″universal″ design. In this regard, we believe that our cost model can enormously help architects in the task of electing the appropriate design under a given workload.

4.6 Conclusions and Future Research

In this Chapter, we have proposed an analytic cost framework aimed at evaluating hierarchical DHTs. Using this framework, we have compared the two main families of hierarchical DHTs: the *homogeneous* design, in which all nodes assume equal roles, against the *superpeer* design, in which a small subset of peers (generally, the most powerful and stable), behave as proxies, interconnecting clusters with highly dynamic membership. More specifically, we have demonstrated that no design is ″universally″ better, and to that effect, we believe that our cost-based model can be very useful in identifying the advantages and disadvantages of a design over the multiple alternatives.

In summary, while traditional superpeer systems have been motivated from the point of view of robustness and scalability, they do not always bring us the best alternative. For example, they can potentially create large imbalances on the traffic imposed on some nodes when inter-cluster communication is high. Although our model does not consider churn in the superlayer, observe that maintenance protocols may impose excessive overhead if superpeers change frequently.

In contrast, homogeneous designs offer a larger amount of load balancing, which derives from their inherent symmetry. Therefore, we believe that when one seeks simplicity, and assuming high anti-locality in communication, node heterogeneous capacities lose preponderance.

This Chapter has sparked a number of avenues for future work.

- We have only analyzed the extreme hierarchical designs on the spectrum. We believe that our framework could be useful to determine which benefits and shortcomings have intermediate designs such as HONET [50]; they may be more appropriate for specific applications.

- Another interesting question consists in obtaining a meaningful set of values for the workload quadruplet $< N, n, l, f >$, for each specific type of application. In this regard, it would be interesting to collect measurement data from real networks, such as content distribution networks and file-sharing systems to provide more realistic results.

5
FALSE CLUSTERING ON PEER-TO-PEER NETWORKS

In this chapter, we study a fundamental question that has been obscured in proximity techniques so far: how often false clustering might happen in reality and how much this affects the overall performance of an overlay. In this regard, we present a novel algorithm called TR-Clustering to cluster nodes in a peer-to-peer overlay network based on their physical positions on the Internet. More precisely, TR-Clustering uses the Internet routers with high vertex betweenness centrality to cluster participating nodes. Briefly, the betweenness centrality of a router is defined as the fraction of shortest paths between all pairs of nodes running through it. Simulation results illustrate that TR-Clustering is superior to existing techniques, with less than a 5% *of falsely clustered peers (of course, relative to the datasets given as input in our simulations).*

5.1 Introduction

"The results you achieve will be in direct proportion to the effort you apply"

Waitley, Denis

As said throughout this thesis, many applications are complex to implement on top of existing peer-to-peer networks. One example is multimedia live streaming. Unlike file sharing, live media is usually delivered synchronously to a large number of clients, with minimum delay in playback compared to the playback at the source. This and many other applications such as VoIP systems, overlay multicast and content distribution networks (CDNs) can significantly benefit from their adaptation to the underlying physical network.

Following our framework to construct hierarchical overlays, a CDN could be implemented as a hierarchical overlay and then, each cluster could be adapted to the underlying network. As we saw in Chapter 3, this practice may achieve significant savings in communication time. However, it poses the following challenge:

How to organize nodes into topologically-aware clusters in a scalable and timely fashion?

Several projects such as IDMaps [59], GNP [60], and Vivaldi [47] have developed infrastructures to estimate network distances (round-trip-times) without the need of direct measurements, offering a scalable substrate, able to generate appropriate input for assistance in clustering.

Although the above initiatives are useful in determining the latencies among nodes that have never communicated, they present a major shortcoming. Congestion or even re-configuration of the network can suddenly change the relative location of many hosts in the network and provoke their reclustering. If this is not done, many hosts may be incorrectly clustered, which occurs when distant hosts are clustered near each other, a problem known as *false clustering*.

Motivated by this observation, in this Chapter, we explore new forms of clustering based on more stable measurements. As the quality of existing clustering algorithms strongly depends on the quality of measurements, it can be significantly inferior to the optimal when there is unstable measurement data such as round-trip-times (RTTs), or artificial data such as network coordinates, based on the embedding of inaccurate information (e.g., RTTs) on a metric space. More precisely, metric embeddings such GNP and Vivaldi assume that RTTs satisfy the triangle inequality.

For two peers u and v, let RTT(u, v) be the distance between u and v in the metric space. Geometric embeddings enforce the triangle inequality, which means that for any three peers u, v, w, RTT$(u, v) \leq$ RTT$(u, w) +$ RTT(w, v). This is not true in the Internet and the accuracy of clustering is clearly affected. For instance, the Vivaldi work [47] observed that 4.5% of the triples violated the triangle inequality in the King data set [61]. Violations of the triangle inequality lead to inaccuracies that rise the fraction of falsely clustered nodes.

Summary of Results

In this Chapter, we make the following contributions:

i. We investigate the impact of false clustering on proximity techniques, a problem that has not received enough attention in previous works, but it is key to solve the topology mismatching problem. We show through an illustrative example,

- how frequently false clustering may happen in reality;
- how much this may affect the performance of proximity techniques; and
- what are the potential gains when the presence of false clustering is negligible.

ii. We present a novel clustering algorithm called TR-Clustering that we deliberately conceived to cut down false clustering. Its most distinguishing characteristic is the use of traceroute to recursively partition the subgroups of topologically closest nodes within a cluster into new, lower-latency clusters.

The use of traceroute as our primary tool not only reduces the fraction of erroneously clustered nodes, but it also makes clusters more resilient to changing network conditions. It turns out to be that most Internet paths are stationary over a time period of one day [62].

Another important characteristic of TR-Clustering is that it is landmark-based. In most landmark-based approaches, clients use the distance measurements to a common, fixed set of "landmark" machines to characterize a host's location in the Internet. In TR-Clustering, however, hosts record traceroute paths to a *dynamic* set of landmarks and use this information to form new, lower-latency clusters. The election of the landmarks is based on vertex-betweenness centrality. Informally, the betweenness centrality of a router can be defined as the fraction of shortest paths going through it. Specifically, TR-Clustering selects as landmarks the nodes with the highest vertex-betwenness. The key intuition is that if traceroute paths from two nearby sources converge to the routers with high betweenness [63], peers can be easily clustered by grouping together those peers that encountered one common landmark on their traceroute paths to other landmarks.

It must be noted that our approach is clearly scalable, as nodes need only to maintain knowledge of (and perform traceroute probes to) the landmark set they used to identify its new cluster.

The rest of the Chapter is organized as follows:

- In §5.2, we survey the related work and establish the distinguishing features of TR-Clustering with respect to prior works.
- In §5.3, we examine false clustering on proximity techniques through Proximity Neighbor Selection. We use several metrics to assess the quality of previous proposals in comparison to optimal clustering. The analysis shows that mitigating false clustering is very relevant to maintain high efficacy in routing.
- §5.4 discusses the principles we heavily rely on for clustering peers based on traceroute data.
- In §5.5, we describe TR-Clustering. The description includes the main algorithms as well as the main implementation issues that might arise in its real deployment.
- In §5.6, we compare our algorithm with previous proposals for the same.
- In §5.7, we summarize and present future research directions.

5.2 Related Work

Much research has been done to allow hosts to discover network topology and accurately predict network distances in a scalable and timely fashion. However, few of these works have examined how to identify clusters of nearby hosts among the potential set of clients, without either requiring the IP addresses of all hosts or falsely grouping distant nodes.

Netvigator (Network Navigator) [64] is a scalable latency estimation tool that aims to discover the closest node offering a certain service to a client. In order to achieve this, Netvigator employs a small number of landmarks and a relatively large set of intermediate routers (termed *milestones*) to increase accuracy. Milestones allow each host to discard those servers that, while having similar landmark vectors to it, are far away from it in the Internet.

Although our strategy bears some similarities to Netvigator, our scope is broader. Specifically, what we want is to provide a generic clustering algorithm, rather than a tool that returns the closest server to a client with a certain confidence. For us, clusters are the basic element to optimize, a task that is obviously more complex to implement than server selection; a cluster is a group of nodes that must be mutually close to each other, while in server selection the closeness is relative to the client. It is for all of this that we see our algorithm as a step further.

Another strategy is based on prefix-length matching of IP addresses. The basic idea is that all the hosts sharing the same first n bits of their IP address are simply grouped into the same cluster. However, the quality achieved by this approach has been questioned [65], as clients with the same prefix may come from distant geographic areas. A representative example is TOPLUS [23], which exploits IP prefixes to organize nearby nodes into topologically-aware clusters. Other approaches propose to employ other sources such as DNS or BGP masks to achieve a better clustering. However, as in TOPLUS, these approaches share an important flaw for their real-world deployment: the use of information such as IP prefixes and DNS domain names that are rarely available to the clients.

In contrast, our approach does not rely on any kind of specialized information to cluster nodes. Rather, each node is responsible for itself; it performs a small number traceroute measurements and thereafter, it decides itself to which cluster go.

There are schemes that use landmarks to cluster hosts. Landmark clustering is based on the intuition that nodes close to each other are likely to have similar network latencies to a few landmark machines spread across the Internet.

A pioneering work on landmark clustering was developed by Ratnasamy et. al. in [66]. They proposed a binning strategy where each node, after measuring its RTTs to the landmarks, sorts them in order of increasing RTT. As a consequence, each node obtains an associated ordering of landmarks, so that hosts having the same landmark ordering are binned together. Although little effort is required for the landmarks to allow nodes identify their bin (they need only to echo ping messages), works such as [64] have identified several shortcomings in this solution. The first one is that it is a coarse-grained approximation, not very effective in differentiating close nodes. The second one is its vulnerability to landmark misplacements: an inadequate selection of landmarks can make landmark ordering completely useless.

In comparison, our solution is more flexible. Since clusters are recursively split, an initial bad selection of landmarks is compensated by adding a specific set of landmarks to each new cluster.

Network coordinate systems (NCs) embed nodes into some metric space so that unmeasured RTTs can be estimated using distance computation in that space. This avoids the cost of explicit measurements between any two nodes the first time they engage in a communication operation. Hence, one may opt here to exploit coordinates to clusters hosts with reasonable accuracy, since nodes close to each other should have similar coordinates. Unfortunately, NCs have several important shortcomings [67]. On the one hand, estimates are quite unpredictable. While many hosts obtain good estimates, a few ones get extremely bad results. On the other hand, NCs assume that the triangle inequality holds, which is certainly untrue. As shown in [68], triangle violations are common, persistent and consequence of Internet routing policies. For instance, the authors of [68] observed that 18% of the triples between 399 Planetlab nodes violated the triangle inequality. Dynamics such as triangle violations force nodes to recompute periodically their coordinates, which lead to oscillations that make hard to determine when it is convenient to update the coordinates. In stark contrast, TR-Clustering prefers stability to predictability and exploits traceroute paths to produce stable clustering, though this supposes more effort to the nodes. Our position is that only clustering makes sense if there are enough means to ensure stability.

In summary, whatever the NC system may be, as the embedding of latencies into any metric space leads to persistent triangle violations, clustering hosts using coordinates can introduce unpredictable oscillations hard to handle, both in the number and the membership of the resultant clusters.

5.3 False Clustering on Proximity Techniques

With the rapid growth of the Internet, overlay networks have been increasingly used to deploy large-scale services. Some examples include application-layer multicast, peer-to-peer file-sharing and content distribution. In order to construct an efficient overlay network, the knowledge of the Internet topology is essential. For many of these services, detecting clusters of nodes that are close to each other may be sufficient to improve their performance and scalability. To better understand this, consider a file-sharing application that offers multiple replica servers from whom download a file. In this scenario, the problem would be where to place the file replicas such that the latency perceived by any requester approximated the average latency of the shortest paths in the Internet. An effective strategy could be:

1. To organize nodes into proximity-aware clusters, such that the nodes within a cluster were relatively closer to one another than the nodes not in that cluster; and then

2. To place a replica in each cluster.

However, clustering algorithms have some drawbacks. Depending on the data set, they may converge to a suboptimal solution with some data points incorrectly clustered, a problem known as false clustering. In our context, false clustering means that the partitioning of hosts has been done into clusters in which their members are not completely close (in terms of RTT) to each other.

The aim in this section is not to investigate which proximity strategy is the most appropriate to achieve optimal network clustering, a problem that is $NP - Complete$. But rather to show that false clustering can be an important barrier to achieve the expected results for a wide range of applications.

5.3.1 Clustering Model

We represent an overlay network as a connected undirected graph $G = (V, E)$, where V represents the set of nodes and E the set of links. Then, we define the clustering of network $G = (V, E)$ as follows:

Definition 50 (Clustering Model). *A clustering $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ of a network $G = (V, E)$ is a division of V into a finite number of sets for which the following two conditions hold:*

I. $\bigcup_{1 \le i \le m} C_i = V$; and

II. $\forall C_i, C_j \in \mathcal{C} : C_i \cap C_j = \emptyset$.

The sets C_i are called clusters. We define the size of a cluster as the total number of nodes in that cluster.

5.3.2 False Clustering Rate

In general, whatever the clustering algorithm may be, there is a fundamental question that needs to be addressed: how "good" or "bad" is a clustering algorithm itself. In this context, this question is equivalent to determining if a clustering algorithm does a reasonable work of grouping nearby nodes into the same cluster.

To answer this question, we define a new metric called *false clustering rate* (fcr) that measures the fraction of falsely clustered nodes in a cluster. Informally, what fcr represents is that if a node were to communicate with a randomly selected node from its cluster, the probability of contacting a falsely clustered node would be equal to the fcr of that cluster. In what follows, we provide a formal description of the fcr.

Definition 51 (Individual fcr). *Let $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ be a clustering of an overlay network $G = (V, E)$. Denote by $\widehat{C_v}$ the set of nodes in the same cluster of a node v with the exclusion of v, i.e., $\widehat{C_v} = C_v - \{v\}$. Denote by $\Gamma_v(r)$ the set of r closest nodes to v in the Internet. For a node v, its false clustering rate is defined as*

$$fcr_v = \frac{|\widehat{C_v} \cap \Gamma_v(|\widehat{C_v}|)|}{|\widehat{C_v}|} \tag{5.1}$$

Informally, fcr_v measures the overlap between the $|\widehat{C_v}|$ closest nodes to v and its current cluster neighbors. We note that to provide $\Gamma_v(|\widehat{C_v}|)$ for all nodes in V, it is necessary to collect the $|V|^2$ pair-wise latency measurements between all nodes. These measurements are only for verification purposes; hence, they should not be interpreted as a part of any clustering algorithm. Specifically, note that $\Gamma_v(|\widehat{C_v}|)$ is not static and depends on the data set given as input. For instance, $\Gamma_v(|\widehat{C_v}|)$ could be implemented using network coordinates, RTT measurements, or traceroute paths.

Finally, we define the fcr as the average over all individual fcrs. More formally,

Definition 52 (fcr). *Let $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ be a clustering of an overlay network $G = (V, E)$. For clustering $\mathcal{C}$ over $G = (V, E)$, we define the false clustering rate as*

$$fcr = \frac{1}{|V|} \sum_{v \in V} fcr_v \tag{5.2}$$

fcr presents the following properties:

I. It assumes a value of 0 at best and a value of 1 at worst;

II. It punishes a clustering only for their false positives; and

III. It penalizes more "sparse" clusters than "dense" clusters.

Property III is very interesting, since fcr rewards clustering algorithms that attempt to create dense clusters. In P2P systems, where clusters are rather dense, it is clearly more convenient to have a metric that biases toward high-density clusters than a density-independent metric. To see that this property holds, we provide the following proof.

Proof. Denote by γ_v the number of falsely clustered nodes with respect to a node v. Clearly, we have that $\gamma_v = |\widehat{C_v}| \cdot fcr_v$. W.l.o.g., consider that two hosts u and v belong to two different clusters C_u, $C_v \in \mathcal{C}$. Now assume that $|\widehat{C_v}| = c \cdot |\widehat{C_u}|$, for some positive constant c. Setting $fcr_u = fcr_v$, we obtain

$$\gamma_v = |\widehat{C_v}| \cdot fcr_v = (c \cdot |\widehat{C_u}|) \cdot fcr_v = c \cdot (|\widehat{C_u}| \cdot fcr_u) = c \cdot \gamma_u$$

Thusly, based on the above equation, we can conclude that cluster C_v can contain c times more falsely clustered peers than cluster C_u. Consequently, a clustering algorithm incurs a higher risk to be penalized if the partitioning is done into sparse clusters than if it is done in dense clusters. This concludes the proof. □

There are some concerns that readers should be aware of:

- First, Why does fcr not account for false negatives? False negatives are the nodes in other clusters that should be included in the present clusters but not have been; and
- Second, Why do we use fcr while there are other quality metrics such as Dunn's Index [69] that are more popular?

Mainly, the reason for not measuring false negatives is that to do so, optimal partitioning must be known in advance, a challenging problem for the Internet itself. So we can only use measures such as fcr that do not depend on the benefit of any a priori classification.

The second question, however, deserves further examination. Cluster validity checking is one of the most important issues in cluster analysis. It aims at the evaluation of clustering results to give an indication of the partitioning that best fits a given data set. However, as shown in [70], the traditional validity criteria (such as variance, density, continuity and separation) is not sufficient when one must deal with *arbitrarily-shaped* clusters. Since we cannot evaluate the results of host clustering based on any prior information about the geometry of the clusters, we have developed a quality measure that do not bias toward a particular shape (e.g., spherical, elliptical). Notice that fcr does not evaluate the compactness of the clusters directly. Rather, fcr measures how well clustering algorithms group hosts that are close to each other, which presumably works better when clusters are mostly compact.

5.3.3 Case Study: Proximity Neighbor Selection

One of the main reasons to cluster nodes in the Internet is to construct topology-aware overlays. For this purpose, it is important to fully understand the relationship between false clustering and proximity techniques. As a case study, we focus on the impact that false clustering exercises upon

Proximity Neighbor Selection (PNS) [71]. According to [71], PNS is amongst the best techniques to adapt overlay networks to the underlying physical network. It works as follows. For a node v, PNS selects, among all the nodes that satisfy the structural constraints of the overlay, the neighbors that are also the closest ones to v.

For a concrete realization of PNS, we use Chord [4]. We observe that although Chord linking rule defines a specific set of neighbors for each node, this rigidity is not inevitably the sine qua non for routing efficiency. More precisely, routing is achieved in at most $\mathcal{O}(\log N)$ hops regardless whether node v picks as its i^{th} neighbor any node in the range $[v + 2^i, v + 2^{i+1})$ $(mod\ 2^n)$ or the closest node to position $v + 2^i$ $(mod\ 2^n)$.

Ideally, PNS would select as the i^{th} neighbor of a node v the closest neighbor lying in the range $[v+2^i, v+2^{i+1})$ $(mod\ 2^n)$. However, identifying the closest node may be unfeasible for large systems, as the size of range $[v+2^i, v+2^{i+1})$ grows exponentially with i. To remedy this situation, we propose a new heuristic that uses clustering to approximate the optimal performance of PNS. With this heuristic, we wish to show the negative impact that false clustering might exercise upon proximity routing, and determine whether it pays off to use clustering to reduce communication delays.

Proximity-Aware Clustering for Neighbor Selection

In what follows, we provide a formal description of our heuristic called C-PNS. Consider a node v and denote by C_v the cluster of this node. To pick the i^{th} neighbor, v proceeds under C-PNS as follows:

1. If there exists one or more nodes that belong to C_v while being at clockwise distance $[2^i, 2^{i+1})$ of v, v selects uniformly at random one server from that range.

2. Otherwise, the i^{th} neighbor of node v is chosen uniformly at random from the set of nodes lying within $[v + 2^i, v + 2^{i+1})$. Note that these nodes do not belong to cluster C_v.

After introducing C-PNS abstractly, it still remains to answer: *Which clustering algorithm should we use to implement C-PNS and explore the effect of false clustering?* Because there exists no *de facto* algorithm for proximity clustering, in the following, we describe two implementations of C-PNS. In our evaluation, we will compare these two implementations against perfect clustering. Throughout this Chapter, we will use the term "Ideal C-PNS" to refer us to this case. The two implementations are:

1. Landmark Ordering (LO) [66]. Landmark ordering causes nodes, at join time, to probe a set of well-known "landmarks", estimating each of their network distances. Each node measures its round-trip-time (RTT) to the landmark machines, and orders the landmarks from the nearest to the most distant in the Internet. Nodes with the same landmark ordering are then clustered into the same bin.

An important thing is noticeable in this scheme: *Landmark ordering is prone to false clustering.* To better understand this, consider two nodes u and v. Node u is located on the East Coast of the United States while node v is physically on the West Coast. Due to the relative positions of the landmarks, u and v may inevitably obtain similar RTT measurements. The result is that they will be binned together despite being far away to each other.

2. Global Network Positioning (GNP) [60]. In GNP, nodes are embedded into a d-dimensional Euclidean space $\mathbb{E}$. Whenever a new node v joins the system, it computes the coordinates of its position in $\mathbb{E}$. In order to do this, v measures the RTTs against a set of λ well-known "landmarks" $\mathcal{L} = \{l_1, l_2, ..., l_\lambda\}$. Observe that for a space of dimensionality d, GNP has to use at least $(d+1)$ landmarks, because otherwise the coordinates for a node might be not unique. As a result of the measurements, v obtains the absolute distances to all landmarks. Then, v uses a Simplex solver to compute its coordinates in such a way that the relative error between the estimated and measured distances is minimized. Having the coordinates of two hosts u and v, every node can then predict the latency between them very easily: it suffices to apply Euclidean distance to the coordinates of u and v.

Further, the use of coordinates is very advantageous. It allows us to treat proximity clustering as a traditional clustering problem, which is solvable by popular algorithms such as K-Means [48]. K-Means is a general heuristic to cluster multi-dimensional data. It has some nice properties, such as speed and simplicity, which has converted K-Means into a popular algorithm for clustering any kind of data. However, the main reason that motivated its election is its adaptability to a wide variety of data distributions.

K-Means [48]. K-Means can be described at a high level as an iterative algorithm that evolves k compact clusters on a d-dimensional data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$. K-Means iterates from some initial solution to obtain a set of k clusters centers $\{c_1, c_2, ..., c_k\}$, such that the average quantization error $q(\mathcal{C})$ defined as

$$q(\mathcal{C}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} d(x_i, c_j) \tag{5.3}$$

is minimized, where $d(x_i, c_j)$ denotes the squared Euclidean distance from point x_i to center c_j. The quantity $q(\mathcal{C})$ measures the average squared distance from each point to the center of the cluster where it was assigned to. The centers are generally initialized with k random points. Then, $\mathcal{X}$ is split into k clusters based on $q(\mathcal{C})$ criterion. Each center is subsequently updated to the mean of the points that belong to the cluster it characterizes. This *partitioning-updating* process is repeated until there is no significant change in the $q(\mathcal{C})$ quantities of the last two iterations.

GNP + K-Means. Let us term by $\mathcal{X} = \{x_1, x_2, ..., x_N\}$ the set of d-dimensional GNP coordinates corresponding to the N nodes in the system. To cluster nodes, we performed the following: we ran K-Means several times on $\mathcal{X}$ and returned the best solution, i.e., the clustering with the lowest quantization error. Multiple runs prevented K-Means to converge to a suboptimal solution due to the random initialization of the cluster centers.

Notice that K-Means will work well when the coordinates are mostly accurate. GNP relies on a small number of landmarks; so a bad choice of the landmarks may negatively affect the accuracy of GNP coordinates.

Key observation. NC systems such as GNP estimate the pair-wise $\mathcal{O}(N^2)$ RTTs among N nodes performing $\mathcal{O}(N)$ measurements. More efficiently is, however, the approximation of Landmark ordering: each node v does not need the $\mathcal{O}(N)$ RTTs to all nodes, since every host in the i^{th} range of v can be directly among the closest nodes to v.

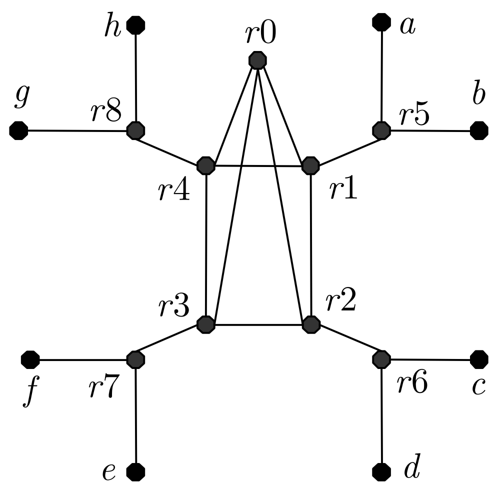


Figure 5.1: *An illustration of the backbone of the star-like graph data set.*

Evaluation

In this section, we simulate the effect of false clustering over Chord. There are several ways to capture this formally, and in our context the most appropriate notion is lookup latency, or the time required to find the node responsible for a key. However, this metric is itself insufficient without a landmark set and an underlying graph which jointly induce a high false clustering rate. In this case, a "clever" decision is to use a symmetric graph such as a tree or a star. For example, in a star graph, all the leaf nodes view the network exactly in the same way. In practice, this implies that many of these nodes may obtain similar RTTs to the landmarks and may be incorrectly assigned to the same cluster.

In our simulations, we used a star-like graph to simulate the effect of false clustering. Fig. 5.1 illustrates the backbone of this graph. In the complete version, there are 800 leaf nodes attached to each of the leaves emanating from the core. Because of the numerous symmetries, observe that preventing false clustering in this topology requires to place at least one landmark in three out of the four trees rooted at r_5, r_6, r_7, and r_8, respectively.

To induce enough false clustering, the landmarks selected for the two implementations were the routers c, g and r_0. As these landmark machines covered only two of the four trees in the star, we explicitly gave rise to a 50% of falsely clustered nodes.

In GNP, we embedded all the nodes of the star into a 2-dimensional Euclidean space. Formally, we embedded all the nodes into $\mathbb{R}^2$ with the standard notion of Euclidean distance $\|x - y\| = \sqrt{\sum_{1 \le i \le 2}(x_i - y_i)^2}$. The landmarks' coordinates were generated after running Simplex solver for 300 iterations, and the leaf hosts' coordinates were computed after repeating the minimization procedure for 30 iterations [60]. The authors of GNP [60] showed that in general 3 iterations are enough to obtain a fairly accurate coordinates. Finally, the clusters were obtained after running 10 times K-Means over the set of GNP coordinates.

The main input parameter of K-Means is k, the desired number of clusters. Since there exists a direct correspondence between clusters and trees in this graph, $k = 4$, a larger k would only slow the algorithms and it would marginally provide new insights about the effects of false clustering upon C-PNS.

Metrics. As a sanity check, we used the *absolute relative error* (RE) [60] to measure the accuracy of GNP coordinates before any comparison was made. For a pair of nodes u and v, their absolute relative error is defined as

$$RE_{uv} = \frac{|\|c_u - c_v\| - m_{uv}|}{\min(\|c_u - c_v\|, m_{uv})} \tag{5.4}$$

where m_{uv} is the measured RTT between u and v, and c_u and c_v are the assigned d-dimensional coordinates of u and v, respectively. Recall that $\|x - y\| = \sqrt{\sum_{1 \leq i \leq d}(x_i - y_i)^2}$.

From [66] we borrowed the average gain ratio (agr) to measure the improvement in latency experienced by Ideal C-PNS relative to the two benchmarking heuristics. As specified in [66], the computation of the agr required us to calculate the following:

- *Average inter-cluster latency*: average latency from a given node to all nodes not in its cluster.
- *Average intra-cluster latency*: average latency from a given node to all nodes in its cluster.

We then calculated the ratio of the inter-cluster latency to the intra-cluster latency for nodes within that cluster. We call this the *node's gain ratio*. At last, we obtained the average gain ratio by averaging over all nodes the individual gain ratios. Intuitively, what the gain ratio represents is that on the average if a node were to communicate with a node from its own cluster instead of a node not in its cluster, then the communication latency would be reduced by a factor equal to its gain ratio.

Results. Fig. 5.2 shows the cumulative distribution (CDF) of RE for GNP over $100,000$ random node-pairs. Specifically, we performed 100 experiments, each with $1,000$ random distances. The RE did not vary significantly across the experiments, which was desirable for this test. The results indicate a poor performance of GNP in the face of landmark misplacements.

In Fig. 5.3, the resultant clustering for GNP is depicted using different marks for each cluster. As can be seen from this plot, GNP clusters are somewhat elliptical in shape. In the center of the figure, there is a cloud of nodes that, albeit close to each other, come from different clusters. These nodes are the bulk of hosts with the largest relative errors.

Fig. 5.4 depicts the CDFs of fcr for Landmark Ordering and GNP. The common characteristic of the two distributions is that more than 50% of the hosts exhibit a fcr superior to 0.5. The reason for this becomes clear when we look at the view from a leaf node to the landmarks. If we assume that each edge represents a distance of 1, it can be easily seen that there are some sets of mutually-distant nodes that have the same RTTs to the landmarks. To make exposition more concrete, let $\vec{d}(v, \mathcal{L}) = \langle d_{l_1}, d_{l_2}, ..., d_{l_\lambda} \rangle$ denote the distance vector of a node v to landmark set $\mathcal{L} = \{l_1, l_2, ..., l_\lambda\}$. Using this notation, it is easy to see that

- node a and its closest leaf node, node b, have $\vec{d}(a, \{c, g, r_0\}) = \vec{d}(b, \{c, g, r_0\})$;
- node e and its closest leaf node, node f, have $\vec{d}(e, \{c, g, r_0\}) = \vec{d}(f, \{c, g, r_0\})$; and
- more importantly, $\vec{d}(a, \{c, g, r_0\}) = \vec{d}(e, \{c, g, r_0\}) = \langle 5, 5, 3 \rangle$,

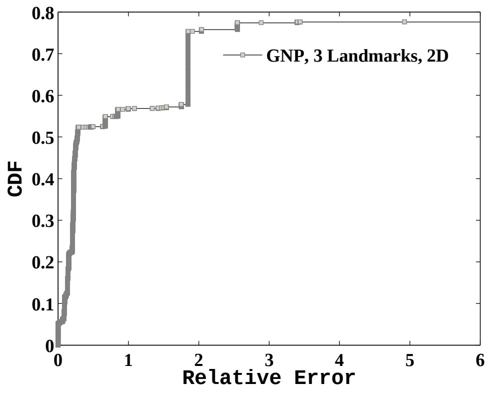


Figure 5.2: *CDF of the absolute relative error for GNP over* 100, 000 *random node-pairs.*

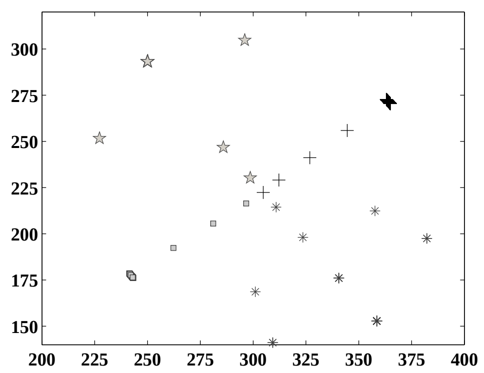


Figure 5.3: 2*-dimensional plot of the resultant clustering for GNP. Each cluster uses a distinct mark.*

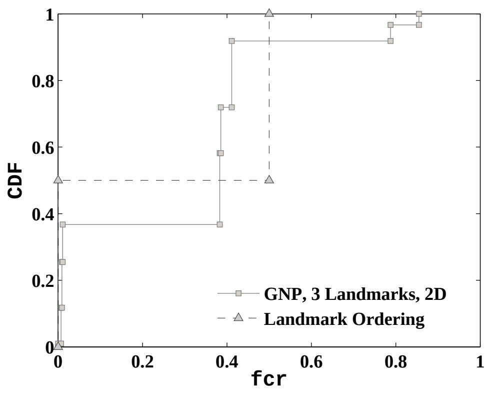


Figure 5.4: *CDFs of the average* fcr *for Landmark Ordering and GNP.*

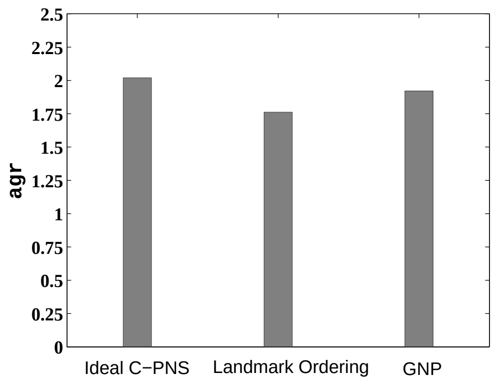


Figure 5.5: agr *for Ideal C-PNS, Landmark Ordering and GNP.*

which means that nodes a and b will be falsely clustered with e and f, even though the RTTs between these two set of nodes are large.

It is worth to note here that perfect clustering is hard to achieve in practice. The reason is that Internet topology changes frequently due to detour paths and multi-homed nodes, i.e., nodes that are connected to more than one ISP. To alleviate this effect, our heuristic has been conceived to deliberately cope with false clustering. Non-surprisingly, it achieves better results than any other heuristic cited in this Chapter (see §5.6).

Fig. 5.5 plots the agr for the three heuristics. From the comparison of Ideal C-PNS with LO, it is clear that agr is negatively affected by the presence of a high fcr. In contrast, the agrs for Ideal C-PNS and GNP are equivalent. This can be explained by the fact that K-Means compensated false clustering through the identification of compact and well-separated clusters. However, when we inspected the distribution of the gain ratio over all nodes, we found some noticeable differences. As an example, the agr for each cluster is shown in Table 5.1. As can be inferred from the table, the agr varies highly from one cluster to another in GNP. Whereas clusters C_1 and C_4 obtained a high agr, clusters C_2 and C_3 only appreciated a slight improvement in their respective agrs. On the contrary, the distribution for Ideal C-PNS was more uniform, which is more desirable, as a "good" clustering should exhibit moderate to high latency savings for all clusters.

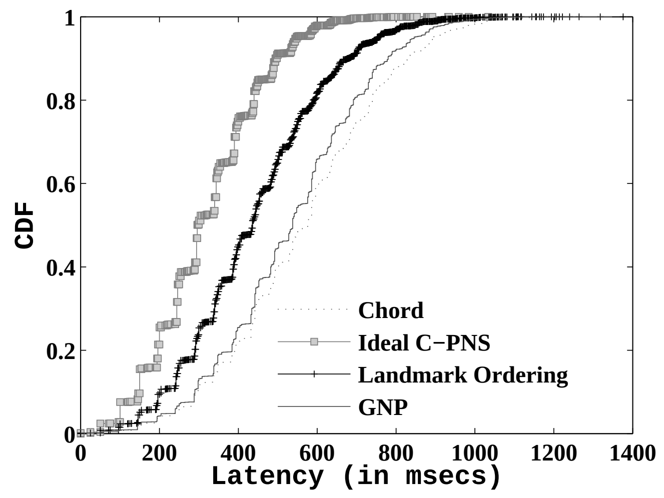


Figure 5.6: *CDF of lookup latency for Landmark Ordering, GNP and Ideal C-PNS over Chord. Plain Chord (labeled Chord in the figure) is used as baseline for comparison.*

Finally, we explored the effect of false clustering upon the presumably improvement in Chord's lookup latency offered by C-PNS. In this experiment, we compared the improvement of LO-based and GNP-based implementations of C-PNS against Ideal C-PNS. The experiment was the following: $100,000$ lookups for random keys were issued by randomly elected hosts. All lookups were recursive and went to the key's successor node. For each lookup, we recorded the time required to locate the key's successor node. The results are shown in Fig. 5.6. As expected, the results agree with our intuition that fcr negatively affects the overall performance of PNS. As fcr increases, the CDF of lookup latency shifts to the right, which indicates that the number of lookups falling into high-latency ranges grows. Observe that Ideal C-PNS exhibits the maximum improvement while plain Chord (labeled Chord in the figure) the minimum.

Table 5.1: Average cluster's gain ratio for GNP and Ideal C-PNS.

	Clusters				
Implementation	C_1	C_2	C_3	C_4	Std. deviation
GNP	2.82	1.27	1.14	2.44	0.84
Ideal C-PNS	2.03	2.03	2.03	1.99	0.02

In summary, this evaluation shows that false clustering may impact negatively proximity techniques and have an important influence in solving the topology mismatching problem.

5.4 TR-Clustering: Principles

TR-Clustering is based on the following two empirical observations:

- The betweenness centrality of Internet routers encountered on traceroute paths towards a set of landmarks; and

- The sampling bias in traceroute-like explorations of Internet.

5.4.1 Internet Routers and Betweenness Centrality

As shown in §5.3.3, landmark set $\mathcal{L} = \{c, g, r_0\}$ is unable to separate the four branches in the star graph of Fig. 5.1. The problem is that RTT measurements from the nodes routing through r_5 and r_7 to each landmark in $\mathcal{L}$ are identical. The result is that the nodes hanging of r_5 and r_7 are clustered together, albeit being far away from each other. To overcome this limitation, the solution consists in adding one more landmark to $\mathcal{L}$ so that the hosts routing through r_5 can be discriminated from the ones going through r_7. This new landmark can be either router r_5 or r_7. The reason is that both nodes are the two remaining routers with the highest betweenness centrality.

First proposed by Freeman [72], the *betweenness centrality* of a vertex is defined as the fraction of shortest paths between pairs of vertices that run through it. Formally,

Definition 53. *For a graph $G = (V, E)$ with n vertices, the betweenness b_v of a vertex v is*

$$b_v = \sum_{s \neq v \neq t \in V, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} denote the number of shortest paths from s to t, and $\sigma_{st}(v)$ denote the number of shortest paths from s to t that run through v. It can be normalized by dividing b_v by the number of pairs of vertices not including v, which is $(n-1)(n-2)$.

Vertex betweenness is thus a measure of the influence of a node over the flow of information between other nodes, specially in cases where information travels over the network by primarily following shortest paths. This property is basic for the high performance of TR-Clustering: if a network contains clusters that are loosely connected to each other by a few intermediate routers, one can expect that all the traffic passes along these routers. The immediate consequence of this is that these routers report the highest betweenness and hence, the problem of detecting the clusters can be solved by equipping the landmark set with such routers. This is the reason why routers r_5 and r_7 can improve the clustering for the star graph. In conclusion, we use betweenness centrality as a useful hint to choose landmarks that have the potential to improve clustering.

To estimate betweenness, we assume the existence of an alias resolution service that strives to identify the multiple interface IP addresses assigned to a single router. In the literature, there exist various techniques to deal with interface resolution. Some examples are the works of Govindan and Tangmunarunkit [73], and Spring et al. in RocketFuel [74]. As our evaluations illustrate, alias resolution is not a requirement. However, we use scriptroute sr-ally [75] to increase the precision of TR-Clustering. More specifically, alias resolution allows us to treat the multiple IP addresses of a router as a single node rather than as multiple separate routers.

Our strategy relies, to a certain degree, on what we term efficient routing, i.e., the assumption that Internet routing policies strive to discover low latency paths between hosts, and that Internet routes from two nearby hosts will be similar when heading to the same destination [63]. This assumption is somewhat optimistic since it is usually violated due to routing policies. However, it appears to be that most Internet routes are stationary over a time period of one day [62].

5.4.2 Sampling Bias in Traceroute Exploration

While underlining the benefits of betweenness centrality in the preciding analysis, astute readers have immediately recognized the assumption that enables the use of betweenness centrality: the assumption that the network is known beforehand, since the computation of vertex-betweenness requires the knowledge of all shortest paths between any two nodes in the network.

To overcome this barrier, we exploit the bias that mapping projects are suffering in the quest of gathering a complete picture of the Internet. Although significant research efforts have been made to construct sufficiently accurate views of the Internet, Internet maps are not complete yet. The reason is that Internet mapping projects rely on a general monitoring strategy that introduces some sampling bias. This strategy can be summarized as:

1. The acquisition of local views of the Internet from a small set of monitors, where a local view is, in practice, a set of $traceroute$ paths that together form a graph; and

2. The merging of local views to obtain a global picture of the Internet.

Recent studies have shown that relying on a relatively small number of monitors introduces some sampling bias. For example, Lakhina et. al. [76] discovered that if a graph has a degree distribution that is very different from a power law (i.e. the probability of a vertex to have degree k is given by $\Pr(k) \propto k^{-\gamma}$), sampling from a small set of monitors will yield a picture of the graph with a node degree distribution following a power law. This owes to the fact that $traceroute$ explorations statistically focus on high vertex-betweennes nodes, sampling more frequently the routers that are closer to the sources.

However, we emphasize that the work of Asta et al. [77] is the more akin to us. They were the first to claim the existence of a correlation between vertex betweenness and the odds of a node to be discovered during a $traceroute$ exploration. Specifically, they investigated sampling biases on:

- *Homogeneous graphs*, in which the degree distribution $\Pr(k)$ is peaked around the mean, with exponential or faster decaying tails.

- *Scale-free graphs*, for which $\Pr(k)$ is characterized by a heavy tail that decays as a power-law $\Pr(k) \propto k^{-\gamma}$.

More importantly, their evaluations shown that the probability to discover a router with high betweenness tends to be 1.0, irrespective of the number of $traceroute$ probes. In our approach, this observation has repercussions on two basic aspects. First, it ensures that $traceroute$ paths from two nearby nodes converge to the same intermediate routers when heading to the same landmarks (i.e., $traceroute$ paths will sample the routers with the highest vertex betweenness). Second, it suggests that it is very likely that one of the landmarks appears on the shortest path from the source to another landmark. We base our clustering heuristic on the latter. Recall that we select as landmarks only those routers that have high vertex betweenness. Hence, it is logical to expect that some landmarks appear on the traces to the other landmarks. Put differently, while such bias is a handicap for Internet mapping projects, it is what allows us to cluster hosts based on the shortest paths to the landmarks. In fact, owing to the way information is processed, we view TR-Clustering as a $traceroute$-based sampling tool which exchanges sources with targets.

5.5 TR-Clustering: Algorithms

The major purpose of TR-Clustering is to have a set of clusters dynamically evolve such that nodes within a single cluster are closer to one another than hosts not in that cluster. Technically speaking, we see TR-Clustering as a clustering algorithm that recursively partitions dense clusters into lower-latency clusters based on the existing similarity found on nodes' traceroute paths to common set of landmarks. It has been is built upon the fact that distant nodes (as measured by traceroute) tend to belong to distinct spanning trees, where for spanning tree we mean the sampled graph induced by the union of all traceroute paths to a given landmark. Strictly speaking, the last statement may not be true. Traceroute paths may intertwine (due to routing policies) and introduce cycles. However, in order to simplify exposition, we will make use of this term henceforth.

Equally important is the interplay between routers with high betweenness and spanning trees. To explain, traceroute probes tend to converge to the routers with high vertex-betweenness when heading to the same landmark, even though they come from far away locations. Motivated by this observation, TR-Clustering progressively reduces false clustering by clustering hosts based on the forks present on traceroute paths. To make this strategy feasible, TR-Clustering is built upon the fact that a small number of routers will appear on almost all traceroute paths. By progressively selecting such routers as landmarks, TR-Clustering will identify new spanning trees that lead to a more optimal clustering of the network (of course, in terms of fcr). More specifically, for each new spanning tree, TR-Clustering creates a new cluster. A small set of landmark machines is set up for bootstrapping. That is, the first clusters are formed by having the hosts in the network partition themselves according to the traceroute probes to this set.

In order to partition a cluster, TR-Clustering performs the following two steps:

1. In the first step, TR-Clustering composes a set of landmarks for assistance in partitioning a cluster. To elect the landmarks, all nodes within the cluster submit their traceroute data, i.e., the paths that were used to create this cluster in the last iteration, to a local repository in the cluster.

   With the traceroute paths from all hosts to the old landmark set, the local repository selects as new landmarks the first λ routers that, besides having the highest betweenness, pertain to different spanning subtrees.

2. In the second step, by scanning the traceroute paths to the new landmarks, each node manages to clusters itself, i.e., without needing to know neither the traceroute paths nor the RTT measurements of the other hosts.

This process of selecting landmarks followed by clustering is repeated until the either there are not more dense clusters, or the desired clustering quality (in terms of fcr) is achieved.

5.5.1 Landmark Selection

Unlike GNP [60], TR-Clustering maintains a separate set of landmarks for each cluster. Ideally, the new landmarks should be elected so that Δfcr after splitting was minimized, where Δfcr is the gain in fcr between the splitting cluster and the new clusters. However, because this practice would be too costly, i.e., it implies verifying all possible combinations of landmarks, we have developed an heuristic based on vertex-betweenness to make landmark selection *computationally* feasible. This heuristic is coded in function `getLandmarks` (see Algorithm 1). For correct operation, it requires two parameters:

I. The desired number of landmarks λ; and

II. The graph, $\widehat{G} = (\widehat{V}, \widehat{E})$, induced by the union of all traceroute paths to the preceding landmark set; $\widehat{V}$ is the set of sampled routers and $\widehat{E}$ is the set of sampled edges.

With this information, `getLandmarks` is able to elect λ non-redundant landmarks to facilitate the splitting. Non-redundancy means that `getLandmarks` tries to identify λ landmarks that can yield the identification of λ lower-latency, far-apart clusters within the current one. This is achieved by iteratively removing the router with the highest betweenness. Recall that the sampling bias on traceroute exploration leads to a power-law distribution of betweenness with a fast decaying tail. Roughly speaking, this implies that a small subset of the routers in $\widehat{V}$ monopolize betweenness.

Let $\mathcal{L}_{new}$ denote the new landmark set. Let $\mathcal{L}_{old}$ be the preceding landmark set, i.e., the landmark set the nodes in this cluster used to be clustered together. Let $\mathcal{B}_{list}$ be the list of sampled routers sorted in decreasing order of vertex-betweenness. Then, `getLandmarks` works as follows:

First, it extracts the router with the highest betweenness from $\mathcal{B}_{list}$. Let us denote by r_{max} this router. Next, `getLandmarks` evaluates two conditions to decide whether r_{max} should be a new landmark or not, i.e., $\mathcal{L}_{new} = \mathcal{L}_{new} \cup \{r_{max}\}$:

- The first condition checks if the degree of r_{max}, `degree`(r_{max}), is greater than 2.
- The second condition verifies if there is a pair of landmarks, l in $\mathcal{L}_{new}$ and $\widehat{l}$ in $\mathcal{L}_{old}$, such that a traceroute path from r_{max} and heading to $\widehat{l}$ goes through l. More specifically, function `isOnPath` returns true when:

$$\exists (l, \widehat{l}), l \in \mathcal{L}_{new}, \widehat{l} \in \mathcal{L}_{old} : l \in \mathcal{T}^*[r_{max} \to \widehat{l}]$$

  where $\mathcal{T}^*[r_{max} \to \widehat{l}]$ denotes the set of all routers that starting at r_{max} appeared on at least one traceroute path to $\widehat{l}$.

The purpose of function `isOnPath` is to make sure that each new landmark lies on a different spanning tree, thus covering the maximum possible area with the new landmark set. Recall that nodes are clustered based on the landmarks they discover on the traceroute paths to other landmarks. In a practical sense this means that if each new landmark belongs to a different spanning tree, the traceroute probes from one tree to another will get routed through these landmarks with high probability.

We observe that function `getLandmarks` ignores the routers with degree equal to 2. The reason is that these routers lie on a path graph. Specifically, a path graph $P = (V_{path}, E_{path})$ is a simple connected graph with $|V_{path}| = |E_{path}| + 1$ vertices and $|E_{path}|$ edges that can be drawn so that all the vertices and edges lie on a straight line. In practice, this means that these routers transfer the betweenness of one neighbor to the other and hence, they do not help to separate each spanning tree from the rest.

Algorithm 1 getLandmarks($\lambda, \widehat{G}$)

```
1: /* $\mathcal{B}_{list}$ is a list with the routers in $\widehat{V}$ sorted in decreasing order of vertex betweenness */
2: $\mathcal{B}_{list} \leftarrow$ new orderedList($\widehat{V}$,"betweenness")
3: $\mathcal{L}_{new} \leftarrow \emptyset$
4: while size($\mathcal{L}_{new}$) $< k$ and length($\mathcal{B}_{list}$) $> 0$ do
5:     /* Extracts the router with highest betweenness */
6:     $r_{max} \leftarrow$ pop($\mathcal{B}_{list}$)
7:     /* If router $r_{max}$ is not redundant, add $r_{max}$ to $\mathcal{L}_{new}$ */
8:     if degree($r_{max}$) $> 2$ and not isOnPath($\mathcal{L}_{old}, \mathcal{L}_{new}, r_{max}$) then
9:         $\mathcal{L}_{new} \leftarrow \mathcal{L}_{new} \cup \{r_{max}\}$
10:    end if
11: end while
12: return $\mathcal{L}_{new}$
```

Communication Cost. In this phase, communication cost strongly depends on the way peers exchange traceroute data with the local repository. The main consequence of this is that a general upper bound cannot be provided for communication cost. In what follows, however, we describe a particular solution to show that TR-Clustering is clearly scalable if sufficient means to distribute communication load are provided. Furthermore, the discussion below will be useful to clarify the communication exchanges that occur between the parties.

At this point is important to highlight that the primary aim of this Chapter is not to investigate what is the best way to exchange traceroute information. Rather, our goal is to provide a generic clustering algorithm that minimizes false clustering, a problem that has not been received enough attention on prior works.

As a trivial solution, we propose that one host in each cluster takes over the responsibility of maintaining the local repository. To ease exposition, we shall refer to these nodes as cluster-heads (CHs). Even partially decentralized, this solution does not solve the whole problem yet. Nodes still need sending their traceroute data to CHs directly, which may not scale for large clusters. To overcome this, a first approximation consists in allowing the nodes to self-organize into a tree for assistance in making a fairer distribution of communication load among the nodes. In this way, CHs can rapidly propagate the new landmark set to its cluster neighbors as well as collecting traceroute information without being overswamped.

For structured overlays, this can be done using a multicast tree rooted at each CH (see Scribe [78]). As an optimization, interior tree nodes could aggregate the traceroute paths of its children. In this way, CHs could be off-loaded from this task. Anyway, the main benefit of this method is that allows TR-Clustering to scale to a large system since each node receives at most f messages, where f denotes the *fan-out* of the tree. In constrast, the propagation of traceroute data incurs $\mathcal{O}(\Delta \log_f \rho)$ time, where Δ denotes the maximum RTT between any two nodes in the cluster.

For unstructured overlays, this responsibility could be assigned to the superpeers. Superpeers are nodes with long lifetime and large capacities (specially, CPU and bandwidth) which act as proxies for *unstable, ordinary* peers. Using gossiping (see [79] for references thereof), superpeers could propagate the traceroute data of its client peers and even aggregate it.

One important feature of TR-Clustering is that it does not require global synchronization in the sense that all clusters in the system must be in the same iteration. Clusters partition themselves, independently of each other, which reduces enormously the amount of communication incurred during clustering.

Space Cost. As the communication cost depends on the way nodes exchange information, we only provide the worst-case space cost for landmark selection. Let $m(t)$ be the number of clusters at iteration t. At iteration t, the space cost at any node is proportional to the number of probes performed by this node, which is $\mathcal{O}(\max\{|\mathcal{L}(C_i)|\}_{1\leq i\leq m(t)})$. Assuming that the maximum number of iterations is T, then the amount of space required at any node is $\mathcal{O}(\sum_{t=1\ldots T}\max\{|\mathcal{L}(C_i)|\}_{1\leq i\leq m(t)})$.

The space cost for the local repository is $\mathcal{O}\left(\rho\left[\sum_{t=1\ldots T}\max\{|\mathcal{L}(C_i)|\}_{1\leq i\leq m(t)}\right]\right)$. Observe that this bound is rather pessimistic, since traceroute paths converge at the routers with high vertex betweenness and often overlap, which saves space. Also note that, in the derivation of this bound, we have assumed a one-to-one mapping between CHs and clusters so that there is no CH being shared for more than one cluster.

5.5.2 Clustering Nodes

Landmark selection is followed by clustering. In this step, we assume that the landmarks for the splitting cluster have been set up and made available to all cluster members. In TR-Clustering, partitioning is a decentralized procedure, in the sense that each node can identify its new cluster with no intervention of the other hosts. Specifically, a node identifies its new cluster after issuing traceroute probes to a few landmarks. For that reason, what we describe here are the steps a single node will follow to be clustered. The protocol steps are described below.

1. The node sends a traceroute probe to each landmark, recording the Internet routers and the round-trip-time to each one. When an Internet router receives a probe packet, it replies with an acknowledgment packet sent back to the originator of the traceroute probe.

2. Upon receiving all acknowledgment packets from the landmarks and routers (if any), the node scans the traceroute paths in order, starting with the routers that immediately follow it on the traces, in order to identify which landmarks come out as routers too. By accounting for these landmarks, the node can identify its new cluster, which is done by calling function `getChildCluster` (see Algorithm 2).

The rationale behind `getChildCluster` is as follows. Since the eligibility of landmarks is based on vertex-betweenness, there is a large likelihood that a node finds a landmark $l \neq \widehat{l}$ on its traceroute path to another landmark $\widehat{l}$. With this in mind, `getChildCluster` can be seen as an iterative algorithm that scans traceroute paths, right to left, looking for the closest landmark to the source that also appears as intermediate router on at least one other traceroute path.

To describe the algorithm, we make use of the following notation. We denote by $\mathcal{T}[h \rightarrow \widehat{l}]$ the traceroute path from a host h to a landmark $\widehat{l}$. Further, we use $\mathcal{L}(C)$ to denote the landmark set used for partitioning cluster C.

We are now in position to sketch how TR-Clustering implements `getChildCluster`. For a host h, first function `firstLandmark` is called passing as input traceroute path $\mathcal{T}[h \rightarrow \widehat{l}]$ and $\mathcal{L}(C)$). This function returns the first landmark $l_{1^{st}} \neq \widehat{l}$ found on the path to landmark $\widehat{l}$. Next, function `networkLatency` is invoked with $l_{1^{st}}$ given as input. As a result of the call, the RTT between h

Algorithm 2 `getChildCluster`(h, C)

```
1: $lat_{min} \leftarrow +\infty$
2: $C_{new} \leftarrow Zero$
3: for each $\widehat{l}$ in $\mathcal{L}(C)$ do
4:     $l_{1^{st}} \leftarrow$ firstLandmark($\mathcal{T}[h \rightarrow \widehat{l}], \mathcal{L}(C)$)
5:     if $l_{1^{st}}$ is $null$ then
6:         continue
7:     end if
8:     /* $lat$ holds the round-trip-time between $h$ and the first landmark other than $\widehat{t}$ found en route to $\widehat{t}$*/
9:     $lat \leftarrow$ networkLatency($h, l_{1^{st}}$)
10:    /* If $l_{1^{st}}$ is closer to $h$ than the current candidate, then $C_{new}$ should represented by $l_{1^{st}}$ */
11:    if $lat < lat_{min}$ then
12:        $lat_{min} \leftarrow lat$
13:        $C_{new} \leftarrow l_{1^{st}}$
14:    end if
15: end for
16: return $C_{new}$
```

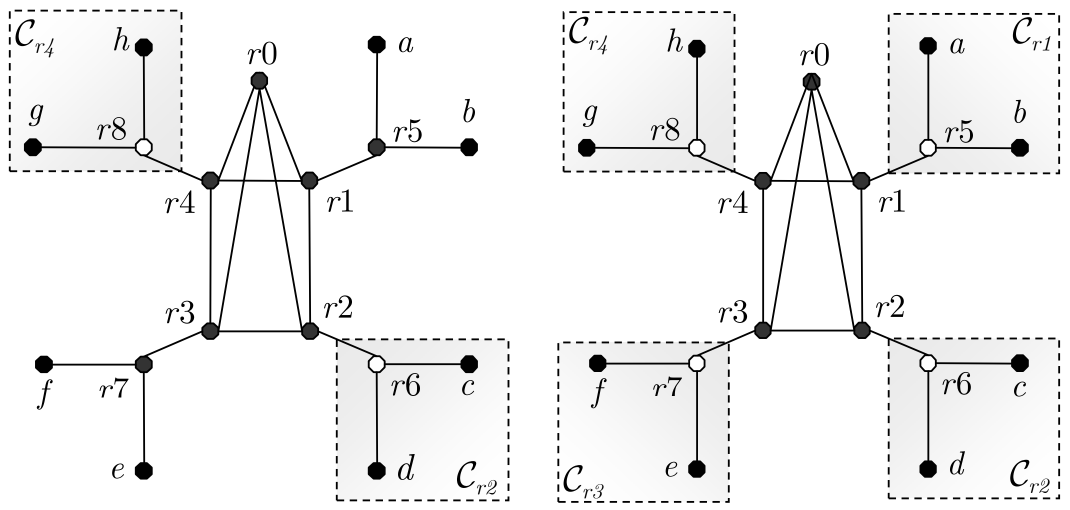
and landmark is $l_{1^{st}}$ is obtained. If then the RTT is smaller than the RTT of the closest landmark found so far, $l_{1^{st}}$ is chosen as the representative for its new cluster C_{new}. Independently if the latter holds, then the same operation is repeated for each landmark in $\mathcal{L}(C)$. In consequence, `getChildCluster` ends up determining the closest landmark to h that was seen on at least one traceroute path. If there exists no such a landmark, `getChildCluster` returns constant $Zero$. This constant signals that h belongs to cluster $Zero$, i.e., the cluster that contains all the hosts that did not find any landmark in their traceroute paths. Note that cluster $Zero$ is therefore prone to have a high fcr. To better understand how this function works, we present an example below.

Example. To be consistent with the exposition of this chapter, we again base our example on the graph illustrated in Fig 5.1. In this case, however, we focus on how this phase is viewed in the eyes of node a when the landmark set is $\mathcal{L} = \{r_1, r_3\}$. As specified above, this phase proceeds as follows. Initially, node a samples the paths $\mathcal{T}[a \rightarrow r_1] = [a, r_5, r_1]$ and $\mathcal{T}[a \rightarrow r_3] = [a, r_5, r_1, r_0, r_3]$ using $traceroute$. Next, a invokes `getChildCluster`, which returns r_1 as the closest landmark to itself. The result of the call is r_1, because r_1 is clearly the first landmark present in $\mathcal{T}[a \rightarrow r_3]$. Consequently, a becomes a new member of cluster C_{r_1}.

Communication Costs. To conclude, we give the communication costs for this phase in terms of number of $traceroute$ probes. At iteration t, we have that the communication cost at any node is $\mathcal{O}(\max\{|\mathcal{L}(C_i)|\}_{1 \leq i \leq m(t)})$, where function $m(t)$ returns the number of clusters at this iteration. The overall communication cost is then $\mathcal{O}(\sum_{t=1...T} \max\{|\mathcal{L}(C_i)|\}_{1 \leq i \leq m(t)})$, where T denotes the maximum number of iterations or partitionings performed by any node.

5.5.3 Putting It All Together: a Clustering Example

In this section, we tie together all the above developments through a simple example. As in other sections, we revisit the graph on Fig 5.1. For bootstrapping, we assume that routers r_2 and r_4 are



a) Using landmark set $\{r_4, r_6\}$, TR-Clustering generates 3 clusters at first iteration: C_{r_4}, C_{r_6} and C_{Zero}

b) At second iteration, C_{Zero} is partitioned into 2 new clusters C_{r_1} and C_{r_3} characterized by landmarks r_1 and r_3, respectively.

Figure 5.7: *A step-by-step clustering example*

the static landmarks in the system. We describe the state of the system at the end of each iteration, where for iteration we mean the partitioning of all the clusters now present in the system.

To begin with, we thus describe the state of the system at the end of the first iteration. In this iteration, 3 clusters are formed: $C_{r_2} = \{d, c\}$, $C_{r_4} = \{g, h\}$ and $C_{Zero} = \{a, b, e, f\}$, which contains the subsets $\{a, b\}$ and $\{e, f\}$ mutually misclustered (see Fig. 5.7a). The reason is that the *traceroute* probes from nodes a, b, e and f to landmark r_2 (resp., landmark r_4) have seen landmark r_4 (resp., landmark r_2) on their paths; thereby introducing ambiguity on clustering decision. The remedy is to incorporate new landmarks to improve decision. These landmarks are routers r_1 and r_3, since they are the two remaining routers in the graph with the highest betweenness. So at iteration two, C_{Zero} is effectively partitioned into two new clusters: $C_{r_1} = \{a, b\}$ and $C_{r_3} = \{e, f\}$ (see Fig. 5.7b). To better understand this, consider node a. Because its *traceroute* path to r_3 follows the itinerary $\mathcal{T}[a \rightarrow r_3] = [a, r_5, r_1, r_0, r_3]$, it discovers that r_1 is on the path to r_3 and hence, it belongs to cluster C_{r_1}. It is interesting note here that the average number of landmarks contacted by each node is exactly 3 as in Fig 5.1, but now with a fcr of 0%.

5.5.4 Traceroute: Implementation Issues

In TR-Clustering, *traceroute* is the network sampling tool used to cluster end hosts. In general, *traceroute* is implemented with Internet Control Message Protocol (ICMP), which means that the probing node sends a series of IP datagrams with increasing time-to-live (TTL) to the destination (in our case a landmark). From the returned ICMP error messages, it obtains intermediate routing information such as router IP address and round-trip-time (RTT). Unfortunately, the *traceroute* behavior explained above is the ideal case. An intermediate router along the path might not reply to probes because the ICMP protocol is not enabled, or because the router simply uses ICMP rate limiting. In order to avoid waiting an infinite time for ICMP replies, the probing node activates a timer after launching each probe. If the time expires and no reply is received, for that TTL, this router is considered non-responding and is termed anonymous. In *traceroute* output, anonymous routers are commonly represented by $'*'$s.

In TR-Clustering, however, the presence of anonymous routers is not as critical as in network mapping projects. To obtain accurate network maps, projects such as Skitter [80] has to deal with anonymous routers explicitly to prevent generating inflated network graphs by considering each $'*'$ as a single router. Since accurately mapping the Internet is not our purpose, we argue that landmark selection can be accomplished without a specific treatment of the anonymous routers. Observe that if we treat each anonymous router as a unique router, its vertex-betweenness will be low and consequently, it will be automatically discarded by function `getLandmarks`. Only if the presence of $'*'$s were large, we might consider to apply methods such as [81] to reduce its number. However, as shown in the literature, it turns out to be that the percentage of anonymous routers found in *traceroute* paths is no more than 15%. For example, in [82], the authors gauged their network inference method by mapping the Internet2 backbone. Using the *traceroute* output from 8 vantage points, they identified 808 unique IP addresses and only 61 $'*'$s. This corresponds to a presence of anonymous routers of less than 7%.

At first glance, another algorithmic aspect of TR-Clustering for which resolving $'*'$s might be critical is the identification of the correct cluster for each node. Recall that each node chooses its new cluster based on the landmark occurrences found on *traceroute* paths. Hence, if for any reason a landmark stopped replying to ICMP probes, hosts might be falsely clustered. However, we believe that this behavior is difficult to show up in practice because the criterion for selecting landmarks prevents it. A landmark is always a router with high betweenness, which means that it had to reply to a large number of ICMP probes in order to become a landmark. Consequently, we do not find unreasonable to assume that landmarks will continue answering ICMP probes in the future.

5.6 Experimental Results

To analyze the performance of our clustering algorithm, we used both synthetic and real Internet topologies. They were the following:

1. RocketFuel (RF) [74]. The main contribution of Rocketfuel is that it shows that the number of disjoint paths between pairs of Points-of-Presence (POP) is frequently low. To give an example, Tiscali, a european ISP, presents on average a single disjoint path among its POPs. This empirical observation reinforces our thesis, since the packets originating at two nearby sources and directed towards the same destination are routed through the same set of intermediate routers. To provide a strong confidence on the results, we ran TR-Clustering on several Rocketfuel POP topologies. We identified each one by its corresponding AS number. They were: the RF-6465 model, with 654 nodes; the RF-1785 model, with 300 nodes; the RF-3356 model, with $1,786$ nodes, and lastly, the RF-1221 model, with $3,515$ nodes.

2. Scale-Free Topologies. (Probably) the most well-known study about the Internet topology was conducted by Faulotsos et. al. [83], which discovered the existence of power-law relationships between several aspects of the Internet AS graph. The key relationship they found was between the out-degree of a node and its rank (i.e., its index in order of decreasing degree). To study the effect of this relationship, TR-Clustering was evaluated on two scale-free topologies. The first one was synthetically generated using the Barabási-Albert (BA) model [84]. The second topology was downloaded from DIMES [46], the largest-scale deployed network tracing technique regarding the number of monitors; more than $8,700$ monitors scattered over five continents.

The degree distribution of BA graphs is a power-law with exponent $\gamma = 3$. For the experiments in this paper, the BA model we created had a core of $5,000$ routers and about $10,000$ links. Such a large number of edges obeyed to the must-have evaluation of TR-Clustering in a network plenty of disjoint paths.

From DIMES, we used the inferred AS graph corresponding to September of 2007. To assign the weights to edges, we used the minimum delay between the first hop in the source AS to the last hop in the destination AS (measured in milliseconds). Further, we removed parallel edges, maintaining the edge with the largest delay. For the simulations, we chose a random set of $1,000$ ASes among the ASes with degree 5 and 6. The reason for doing so was to verify if TR-Clustering is capable to follow primary paths and miss out on backup paths, i.e., paths that are only used when primary paths are not available, for example, due to network congestion. Finally, we attached 100 peers to each AS to simulate a P2P overlay network of $100,000$ peers.

The basic properties of the DIMES topology are the following: $21,902$ AS nodes, $61,399$ AS links, and a power-law exponent $\gamma = 2.14$ in its degree distribution (after removing the parallel edges).

3. GT-ITM (TS) [45]. GT-ITM is a software tool for creation, manipulation, and analysis of graph models for the Internet. It has been used by many networking researchers in a variety of ways, most often to create topologies to use in simulation studies. For this reason, we found convenient to use GT-TIM to generate two synthetic Internet graphs. The first one, the TS-1K network was a network graph with a core of $1,000$ routers arranged in a hierarchical manner. In this graph, there were 3 transit domains at the top level with an average of 4 routers in each. Each transit router presented an average of 3 stubs attached to it, and each stub had an average of 24 routers. For the sake of easy visual examination, the second topology was restricted to a backbone of 45 routers (see Fig. 5.12). The pair-wise latencies between nodes were calculated according to GT-ITM default policy in both graphs.

4. PlanetLab [17]. We also used two different datasets derived from measurements on PlanetLab network test-bed. Both datasets involved 33 PlanetLab nodes, but one was elaborated using scriptroute sr-ally [75] to filter as much router aliases as possible. The 33 hosts were distributed around the world: 16 nodes were in Europe, 13 other nodes were in America and 4 in Asia. To elaborate these datasets, we collected the delays along with traceroute paths between any two PlanetLab hosts over a two-day period: on January, $2-3$, 2007. In fig. 5.8, we show the CDF of the RTT over all pairs of Planetlab nodes. The average RTT for the *alias* and *no-alias* datasets were of 63 and 68 milliseconds, respectively.

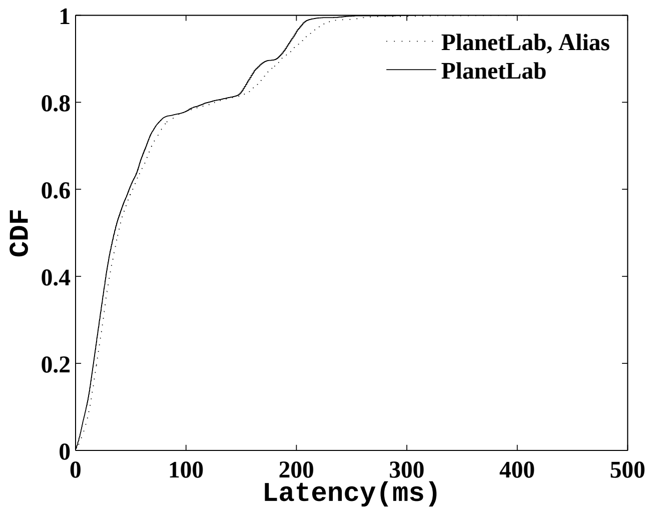


Figure 5.8: *CDF of the RTT over all pairs of PlanetLab nodes.*

5.6.1 Comparison of TR-Clustering against Global Clustering

In this section, we compare the performance of TR-Clustering with K-Means. With regards to TR-Clustering, the question we wanted to address was:

Can a traceroute-based, decentralized clustering algorithm do a reasonable job of placing nearby hosts into the same cluster?

To answer this question, we made use of K-Means because it represented the ideal case where a global site has all the pair-wise $\mathcal{O}(N^2)$ latencies, and can exploit such a "privilege" to achieve optimal clustering.

Setup. For this test, we preferred TS-45 graph over the rest of datasets. The main reason was to illustrate the resultant clustering in Fig. 5.12, so that a a deep understanding of TR-Clustering can be easily gained by the reader. This experiment required the following steps:

- We first ran K-Means on the N-by-N matrix containing the pair-wise latencies between all hosts; and
- Then, we applied TR-Clustering on the traceroute paths from all hosts to the landmark set. We made the nodes join the overlay progressively, triggering a new partitioning when the density of any temporary cluster exceeded ρ.

Because there was no evidence for the optimal number of clusters, we ran both algorithms for different values of k. It is important to note that for a given k, TR-Clustering cannot ensure that the clustering process ends up with exactly k clusters; it depends on the particular value of ρ. The larger the value of ρ, the less the number of clusters. In this test, we set ρ to $\lfloor 0.3N \rfloor$. We emphasize here that there is no any general reason to choose this particular value of ρ. The basic reason to fix $\rho = \lfloor 0.3N \rfloor$ was to provide sufficient traceroute information to make sure that simulations were not inadvertently biased against TR-Clustering.

Metrics. In this experiment, we measured the average fcr, the average gain ratio and the Davies-Bouldin index [85], which according to [86] is among the best indices to evaluate the performance of a clustering algorithm. These metrics were chosen to provide an uncorrelated analysis of the quality of TR-Clustering, as each metric captures a distinct aspect of clustering.

The Davies-Bouldin (DB) index is a measure to assess the quality of a clustering with respect to a benchmarking clustering, either created by distinct algorithms, or even by the same algorithm for different parameter sets. This metric [85] is a function of the ratio of the sum of *within-cluster* scatter to *between-cluster* separation.

Definition 54. *Let $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ be a dataset of n points and $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ its clustering into k clusters. Let $d(x_i, x_j)$ denote the distance between x_i and x_j. Let $\Delta(C_i)$ denote the scatter distance within cluster C_i, i.e., $\Delta(C_i) = \frac{1}{|C_i|}\sum_{x \in C_i} d(x, c_i)$, where c_i is the cluster center (mean). Then, the Davies-Bouldin index is defined in the following way:*

$$DB(\mathcal{C}) = \frac{1}{k}\sum_{i=1}^{k} \max_{j=1..k, i \neq j} \left\{ \frac{\Delta(C_i) + \Delta(C_j)}{\delta(C_i, C_j)} \right\}$$

where $\delta(C_i, C_j)$ is the inter-cluster distance, i.e., the distance between the centers c_i and c_j that characterize clusters C_i and C_j, respectively.

From the above definition, one can easily see that the DB index represents the average similarity between each cluster C_i and its most similar counterpart. As it is desirable for the clusters to have the minimum possible similarity to each other, the DB index is small when the clusters are mostly *compact* and *far away* from each other.

Results. In Fig. 5.9, we plot the average fcr for TR-Clustering and K-Means. The x-axis represents the number of clusters k, while the y-axis shows the average fcr. Non-surprisingly, K-Means achieves a fcr of 0% for many values of k. The reason is as follows. K-Means knows all the pairwise $\mathcal{O}(N^2)$ RTTs between hosts. Thus it can iteratively partition the hosts into clusters so that the sum, over all clusters, of the distances from the hosts to the cluster centers is minimized. In contrast, TR-Clustering can only converge to the optimal solution by recursively partitioning the densest clusters at each iteration. The consequence of this is that TR-Clustering needed at least 14 clusters to reduce the mean fcr to 0%. This shows that if clusters evolve sufficiently, TR-Clustering can be equivalent to K-Means. We outline that a fcr of 0% is possible because TS-45 is synthetic and contains non-overlapping proximity clusters.

Fig. 5.10 depicts the average gain ratio for TR-Clustering and K-Means. From the figure, we can clearly see that the average gain ratio increases linearly, with the same slope for the two curves. This suggests that the temporary clusters identified by our scheme provide savings comparable to those reported by K-Means. In fact, the benefit can be considered superior in TR-Clustering, since each host is clustered only with its own information. This distributes load and clustering process among all hosts in the network, which makes TR-Clustering very suitable for highly decentralized environments such as P2P systems.

In Fig. 5.11, we depict the Davies-Bouldin's index as a function of k. As before, TR-Clustering's results are very similar to those reported by K-Means. This suggests that clustering hosts based on the forks identified on traceroute paths works reasonably well: clusters are far-apart and compact.

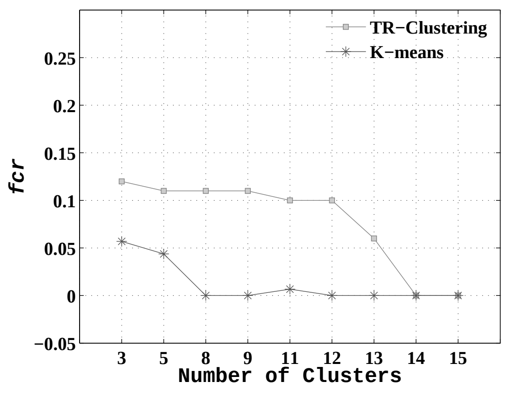


Figure 5.9: *Comparison between TR-Clustering and K-Means on GT-ITM: Evolution of the average* fcr *with an increasing number of clusters*

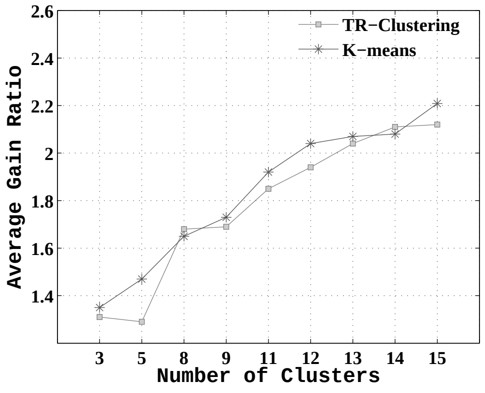


Figure 5.10: *Comparison between TR-Clustering and K-Means on GT-ITM: Evolution of the average gain ratio with an increasing number of clusters*

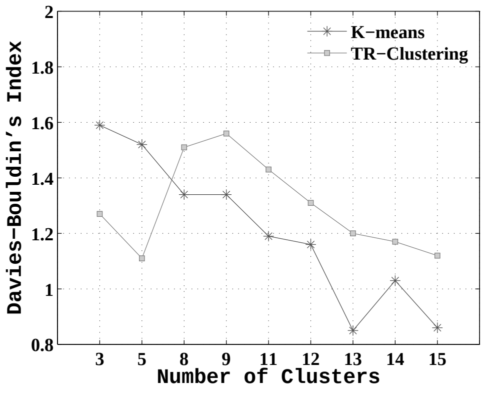


Figure 5.11: *TR-Clustering vs. K-Means on GT-ITM: Evolution of Davies-Bouldin's (*DB*) index with an increasing number of clusters*

However, as we will see for PlanetLab and DIMES, this argument does not longer hold. This claim will be discussed in detail in the following sections.

Finally, we illustrate the resultant clustering for $k = 15$ in Fig. 5.12. The black circles stand for the transit routers. Each cluster is drawn with a shape and a color which distinguishes it from its adjacent clusters. As this figure illustrates, TR-Clustering puts the mutually nearby hosts into the same cluster.

Fig. 5.13 depicts the landmarks that were elected by TR-Clustering. For clarity, the color of the landmarks faints gradually from black to white at each iteration. The arrows signal the landmarks chosen at the next iteration. From visual inspection of the figure, it is clear that the next-iteration landmarks were mainly the routers with the highest betweenness from the remaining ones.

5.6.2 Impact of the Density Threshold

Fig. 5.15 depicts the complementary cumulative distribution (CCDF) of degree (a), and vertex-betweenness (b), for RF-1221 dataset at the end of the first iteration. Clearly, the figure proves our

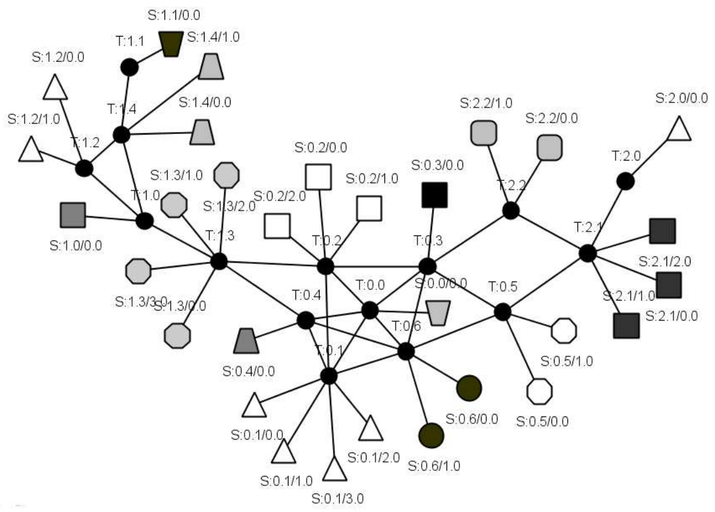


Figure 5.12: *Resultant clustering for TS-*45 *topology. For instance, nodes S2.2/1.0 and S2.2/0.0 are in the same cluster.*

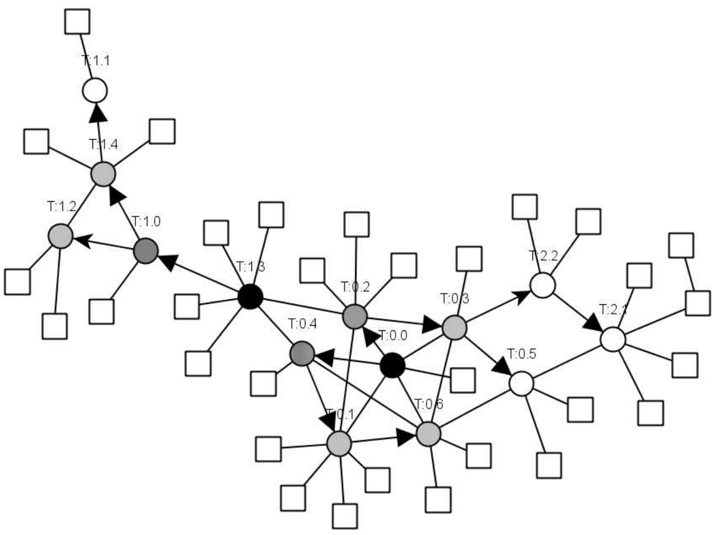


Figure 5.13: *Landmark selection step-by-step in TS-*45 *topology. Black arrows indicate the landmarks chosen at next iteration.*

intuition. Both subplots show a power-law curve with a fast exponential decaying tail (in log-log scale), which is more pronounced in the case of vertex-betweenness.

One important parameter of TR-Clustering is the density threshold. It controls cluster splitting and hence, it may influence the quality of the clusterings. For this reason, we now study the sensitivity of TR-Clustering w.r.t. this parameter. Fig. 5.14 plots the average fcr of six datasets after generating $k = \lfloor 0.3N \rfloor$ clusters. Each group of two bars compares the fcr for the thresholds $\rho = \lfloor 0.01N \rfloor$ and $\rho = \lfloor 0.1N \rfloor$, respectively. We make the following two observations.

- The average fcr is low for all models. For GT-ITM and RocketFuel, the average fcr is clustered around 2%, whereas for BA this value is slightly higher; it stabilizes around 3%. This means that if a host was to communicate with a random host within its cluster, the probability that the recipient was not among the closest hosts to the source would be less than 3%.

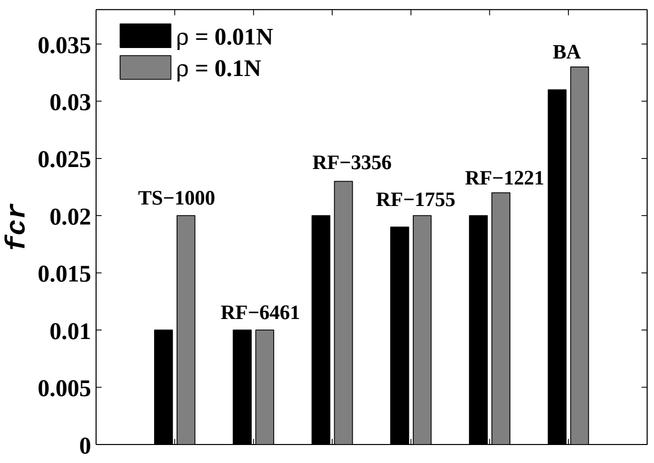
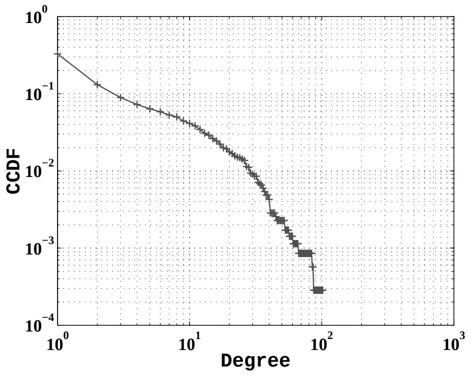
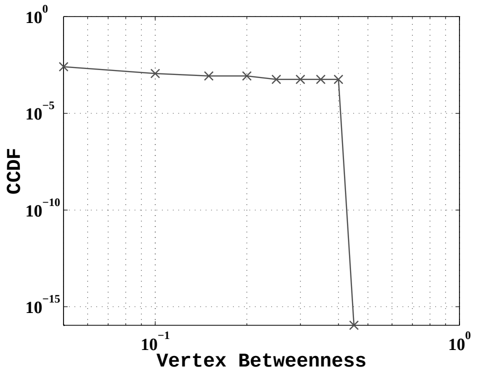


Figure 5.14: *Effect of the density threshold on* fcr*. Each group of two bars illustrates the* fcr *experienced on six models for thresholds $\rho = \lfloor 0.01N \rfloor$ and $\rho = \lfloor 0.1N \rfloor$, respectively. As suggested by this figure, TR-Clustering appears no particularly sensitive to the current value of ρ.*



a) CCDF of degree in RF-1221 dataset



b) CCDF of vertex-betweennes in RF-1221 dataset

Figure 5.15: Sampling bias (power-law behavior) in RF-1221 dataset after first iteration.

- Decreasing the density threshold helps to decrease the average fcr. However, this reduction is not proportional. For instance, the reduction in the BA model is only of 4% while ρ has been decreased ten times. This suggests that TR-Clustering is not particularly sensitive to the value of ρ.

In addition, we verified if the cluster graphs induced by the union of all traceroute paths are affected by some kind of sampling bias. To that purpose, we recorded the degree and betweenness of the routers discovered by traceroute probes on RF-1221 dataset at the end of the first iteration.

5.6.3 Performance Evaluation on PlanetLab

The purpose of this section is to give a quantitative idea of TR-Clustering's performance on PlanetLab, a global, real testbed that supports the development of new network services. To begin

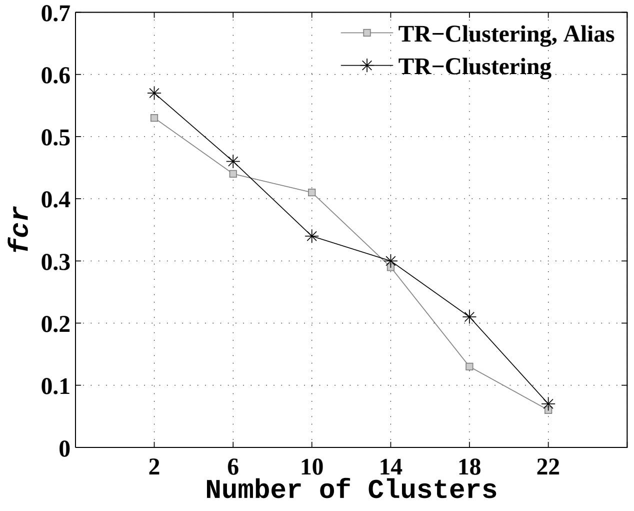


Figure 5.16: *Average* fcr *of TR-Clustering on PlanetLab with alias and without alias resolution.*

with, we study the effect of alias resolution on TR-Clustering. As anonymous routers can greatly inflate traceroute graphs, we considered the following question:

Does false clustering diminish significantly by reducing the number of anonymous routers?

To answer this question, we conducted the following test. We fed TR-Clustering with PlanetLab *alias* and *no-alias* datasets. For each dataset, we recorded the mean fcr as a function of the number of clusters. The results are shown in Fig. 5.16.

At first sight, aliasing information should improve accuracy. However, what Fig. 5.16 suggests is the contrary to what was initially expected: aliasing information does not significantly diminish fcr; the two curves present similar slopes, which clearly demonstrates that aliasing information is not strictly necessary to spot the best results. On the one hand, this result is excellent, as router aliases are hard to collect, but on the other hand, it is also negative, as aliasing information allows TR-Clustering to operate with less landmarks. Specifically, the use of *sr-ally* reduced about 25% the number of landmarks in PlanetLab.

For the rest of experiments, we compared TR-Clustering against GNP and Vivaldi [47]. Briefly, Vivaldi models the network as a collection of springs that push on the coordinates of each node. The idea is that each node is represented by a *unitary mass* connected to each neighbor by a spring with the *rest length* set to the measured RTT. The actual length of the spring is the latency predicted by the coordinate space. Since each spring strives to have its actual length equal to its rest length, each node continuously updates its coordinates. After several updates, the coordinates converge and are ready for latency estimation.

Setup. In order to make a fair comparison, we set GNP and Vivaldi parameters to the "optimal" configuration identified in the original works:

- For GNP, we set the number of dimensions, d, to 5 and 7, which according to the authors yields the best results [60].

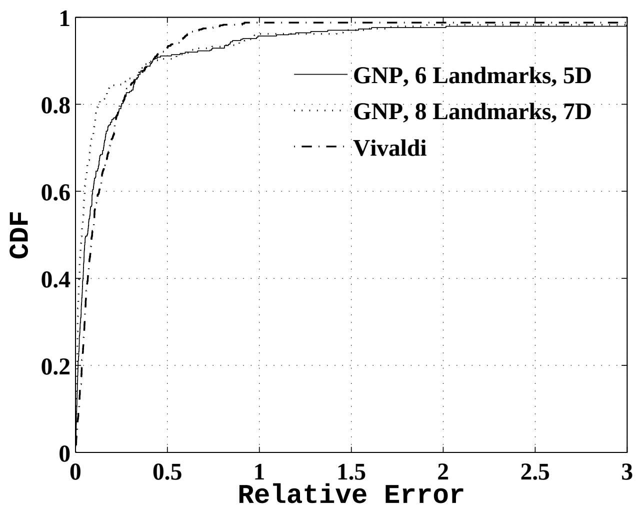


Figure 5.17: *CDFs of RE for Vivaldi and GNP on PlanetLab.*

- For Vivaldi, we made use of the implementation provided by Bamboo [49], which embeds nodes into a 2-dimensional Euclidean space. To stabilize the coordinates, we fed the same set of measurements to Bamboo's Vivaldi implementation 100 times.

The same set of measurements were supplied as input to each scheme.

Results. Fig. 5.17 plots the CDFs of RE for Vivaldi and GNP with $d = 5$ and $d = 7$, respectively. Our results show a draw between the two schemes: the 90th percentile relative error is 0.5, which coincides with the CDF of RE for PlanetLab on Vivaldi's original work. We note that a completely objective comparison between TR-Clustering, Vivaldi and GNP is difficult in this case due to the lack of benchmarks. However, we believe that by computing Vivaldi and GNP coordinates with high accuracy, the confidence on our results will be high enough to convince readers about the utility of TR-Clustering.

In the next experiment, we compared TR-Clustering against Vivaldi, GNP($d = 5$) and GNP($d = 7$). It is customary to run K-Means several times to find a good clustering solution. As in §5.3, we ran K-Means 10 times for each set of coordinates, and we returned the best solution for each one. The measures we chose to express numerically the comparison between the three schemes were again the fcr and the Davies-Bouldin index.

Fig. 5.18 plots the evolution of fcr for an increasing number of clusters. Our observations are as follows.

- For $k < 8$, TR-Clustering performed poorly, with up to a 60% of fcr when hosts were split into two clusters. The explanation for this is that TR-Clustering could not separate the 4 PlanetLab hosts in Asia from the rest of PlanetLab machines. In contrast, Vivaldi was able to do it and found the *optimal* clusters. Fig. 5.19 plots this clustering, with the two cluster centers marked with a cross. The figure illustrates two *well-separated* clusters; the small one characterizes the Asian PlanetLab hosts; whereas the large one contains the nodes in Europe and America.

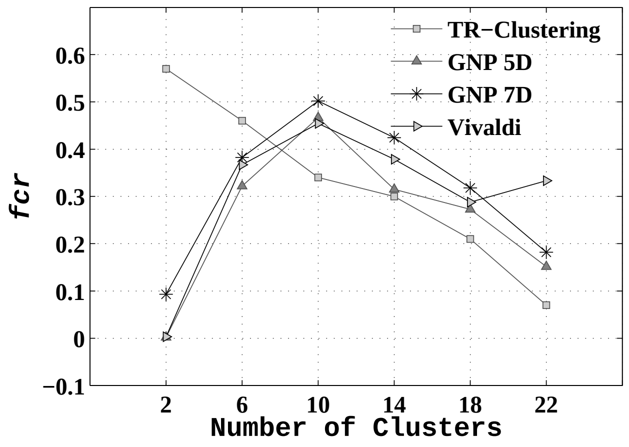


Figure 5.18: *Evolution of the average* fcr *as a function of* k *for TR-Clustering, GNP and Vivaldi on PlanetLab no-alias dataset.*

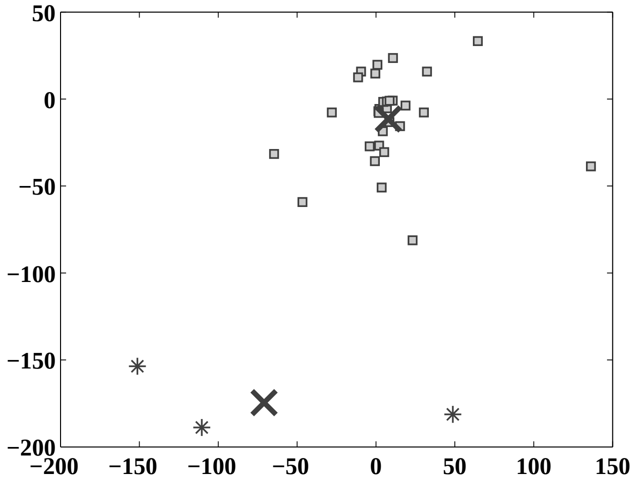


Figure 5.19: 2*-dimensional plot of the resultant clustering for Vivaldi.* $k = 2$.

- For all $k \geq 8$, fig. 5.18 illustrates that TR-Clustering was superior to the other three schemes. This can be explained by the fact that TR-Clustering can generate clusters of *arbitrary* shape if they characterize better the underlying network. This allowed us to split correctly the cloud of the American and European PlanetLab hosts. As can be seen in Fig. 5.19, the positions of these hosts were highly overlapped.

Fig. 5.20 plots the evolution of the Davies-Bouldin's index for an increasing number of clusters. As shown in the figure, when the number of clusters is less than 18, Vivaldi and GNP outperform TR-Clustering. The explanation of this result is the same as in the preceding plot. In order to reduce false clustering, TR-Clustering preferred to generate *compact* clusters of *arbitrary shape* than creating *well-separated*, *spherical* clusters. In stark contrast, GNP and Vivaldi boosted the creation

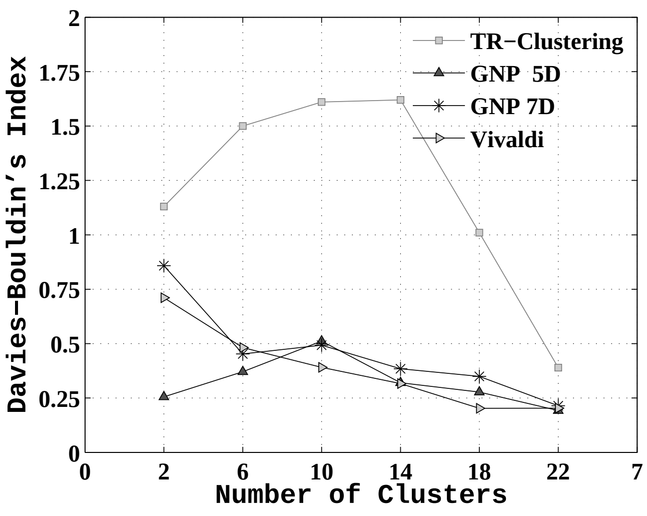


Figure 5.20: *Evolution of the Davies-Bouldin's Index as a function of k for TR-Clustering, GNP and Vivaldi on PlanetLab no-alias dataset*

of well-separated clusters and hence, they obtained better results. However, this was done at the expense of an increase in the fraction of falsely clustered nodes as shown in Fig. 5.19.

5.6.4 Performance Evaluation on DIMES

To test our algorithm with more real data, we also fed our simulator with DIMES. In this case, we compared the performance of TR-Clustering only with Vivaldi. As in the preceding evaluation, we used fcr and the Davies-Bouldin index as the main performance metrics.

Setup. Using Vivaldi embedding, we first mapped the $100,000$ DIMES peers into a 2-dimensional Euclidean space. Then, we executed K-Means exactly 10 times on the set of Vivaldi coordinates and returned the clustering with the minimum quantization error (see Eq.(5.3) for further details).

Because the accuracy of Vivaldi coordinates depends on the number of updates performed by each host, we varied the number of coordinate updates to study the interplay between accuracy, communication cost and false clustering. The obvious relationship is that *"the higher the accuracy, the higher the communication cost"*, but to which extent accuracy helps to reduce false clustering is a question that deserves more attention. While communication increases linearly with the number of updates (each update requires at least one ping measurement with the corresponding neighbor), the key question we tried to answer in this section was:

Does a linear increase in accuracy imply a linear decrease in the false clustering rate?

To answer this question, we generated three sets of Vivaldi coordinates, each one identified by the number of updates performed by each node before K-Means was run. Particularly, we created Vivaldi-100 (100 updates), Vivaldi-1000 (1K updates), and Vivaldi-5000 (5K updates) data sets.

It must be noted that it is unlike to find a set of coordinates with 5000 updates in practice: due to the frequent arrivals and departures of nodes, it is hard to find a set of peers that can update

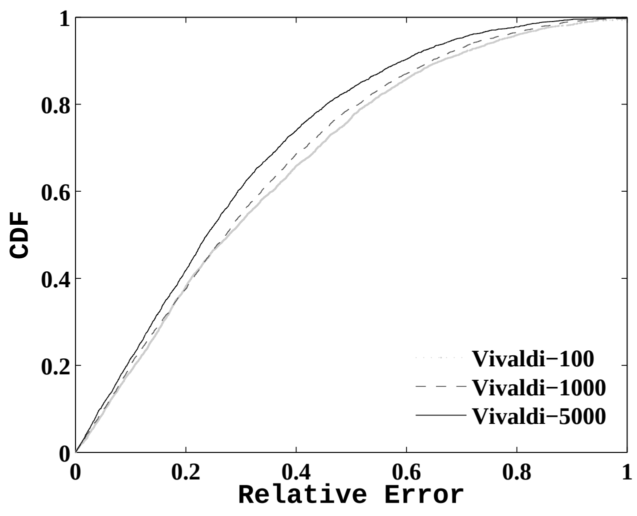


Figure 5.21: *CDF of RE for Vivaldi over all pairs of DIMES hosts.*

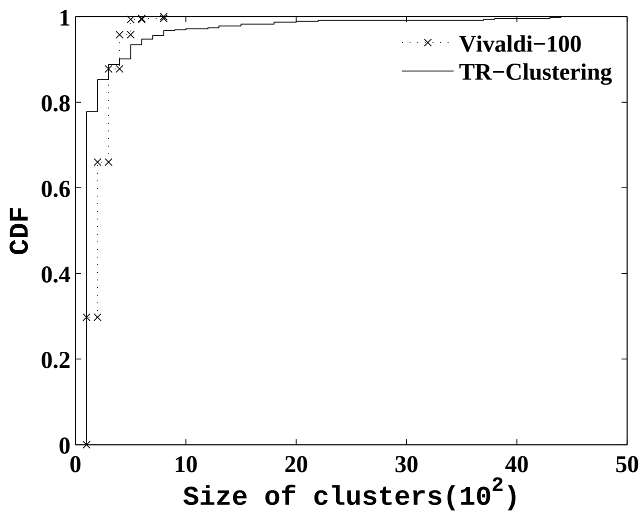


Figure 5.22: *CDF of cluster size when* $k = 450$.

$5,000$ times their coordinates without being penalized *excessively* by the inaccuracies of the joining peers' coordinates. Our evaluation was thus optimistic with respect to Vivaldi's real performance.

Results. In fig. 5.21, we illustrate the CDFs of the RE for the three Vivaldi configurations. As can be seen in the figure, the three configurations perform similarly. To explain, approximately 80% of the hosts exhibit a RE between 0.0 and 0.5, which coincides with the cumulative distribution of the RE found on PlanetLab dataset. Needless to say, as the number of coordinate updates increases, the relative error diminishes (albeit not significantly). More specifically, Vivaldi-5000 performed the best, Vivaldi-1000 the next, and finally, Vivaldi-100.

Fig. 5.22 plots the CDFs of the cluster size (in hundreds of nodes) for TR-Clustering and Vivaldi-100 when $k = 450$. The trends on both curves tell us that TR-Clustering presents a higher

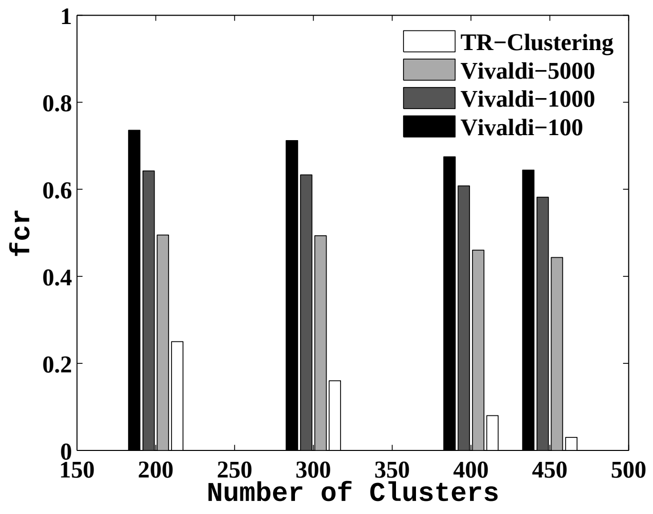


Figure 5.23: *Average* fcr *as a function of* k *for TR-Clustering and Vivaldi.*

variability in cluster size. To be more specific, while in Vivaldi-100 clusters were more uniform and contained approximately 300 nodes on average, in TR-Clustering we found clusters of up to $4,500$ peers. Such a variability comes from the following fact. In DIMES dataset, leaf ASes are commonly connected to the Internet backbone through a small set of transit ASes. Consequently, when clustering hosts, TR-Clustering also grouped into the same cluster the leaf ASes that shared similar routing paths to the backbone, transforming *path diversity* between ASes into *variability* in cluster sizes.

In fig. 5.23, we plot the average fcr for an increasing number of clusters. Our results indicate the following.

- TR-Clustering was clearly the best. For $k = 450$, TR-Clustering decreased the fcr to 3%, leading to a potential performance ≈ 15 times better than the best of three Vivaldi configurations.

- Surprisingly, Vivaldi could not diminish the fcr below 40%, which suggests that it is essential to make no assumptions on the cluster forms. Precisely, the superiority of TR-Clustering can be explained by its strong independence on cluster shapes. In addition, Vivaldi establishes a metric embedding; hence, it is incapable to adapt itself to detour paths as such paths violate the triangle inequality.

Fig. 5.24 illustrates the variation of Davies-Bouldin's index with the number of clusters. Similar to what was observed for PlanetLab, our results show that TR-Clustering and Vivaldi exchanged again trends w.r.t. DB-*index*. We recall that DB-*index* has a geometric (typically centroidic) view of clustering, which means that it works well when the underlying data contains clusters of spherical form, but it is susceptible to data where this condition does not hold. This observation is what explains the tendency of our clusterings to obtain high values for this metric. We note that despite the negative results, TR-Clustering is superior to Vivaldi, since what we seek is that the effects of false clustering are smoothed out. The inclusion of these results owed to the need

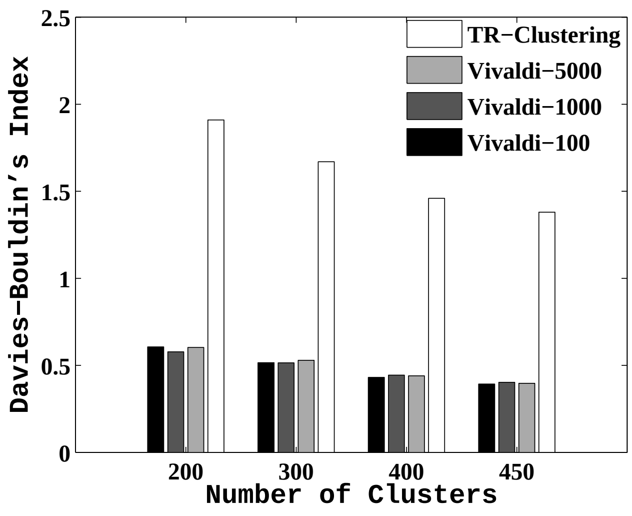


Figure 5.24: *Davies-Bouldin's index as a function of k for TR-Clustering and Vivaldi.*

of verifying if traditional metrics such as Davies-Bouldin's index or Dunn's index are adequate to evaluate the quality of proximity clustering in the Internet.

In summary, although the accuracy of TR-Clustering is not 100%, it provides a better accuracy than coordinate systems when clusters evolve dynamically, a property that we believe it is critical to achieve good clustering results on the Internet.

5.7 Summary and Future Directions

In this Chapter, we have provided an extensive study about false clustering. For this purpose, we have investigated the negative impact that false clustering has on Proximity Neighbor Selection, and have shown that false clustering is a serious problem induced by the Internet itself. To cope with this problem, we have developed a new clustering algorithm called TR-Clustering which uses traceroute to cluster together those hosts that share the same paths to a common set of landmarks. The main contribution of TR-Clustering is that it can form stable clusters with a low rate of falsely clustered nodes because traceroute information is more stable than network coordinates.

Our simulation results show that the utility of TR-Clustering seems to be promising, especially on the Internet where it is difficult to get a consistent set of delay measurements at all times (this affects more NCs). In this context, we believe that TR-Clustering can provide both the accuracy and the stability necessary to facilitate the deployment of real peer-to-peer applications.

Possible directions for future work:

- A traceroute probe incurs message overhead proportional to the length of the probed path. It would be interesting to devise a smarter version of traceroute that does not probe every hop, but reports the same accuracy in terms of false clustering rate. A good start could be as follows. Instead of incrementing the TTL of the ICMP probe packet by 1, "smart-traceroute" could intelligently skip some intermediate hops.

- Concerning the preceding question, it would be interesting to study more extensively the trade-off between measurement overhead and accuracy in order to find the optimal operating point for TR-Clustering.
- Another aspect of our future work could be to apply our technique to real applications such as constructing efficient hierarchical overlay networks. This implies adapting the structure of the overlay to the current clustering, which may induce marginal or even major changes in the routing tables of nodes.

6
Conclusions and perspectives

6.1 Conclusion

The peer-to-peer paradigm has grown in significance in the Internet, both in terms of the number of participating users and the traffic volume. Nowadays, structured overlay networks, also called Distributed Hash Tables (DHTs), are considered as the most scalable substrates in the peer-to-peer paradigm. However, there still remain some critical issues to address before DHTs can experience a wide adoption. Some of them are: (1) offering flexibility to organize users into a domains, either by interest, by affiliation or whatever, in order to ensure routing paths stay entirely in a domain when possible; (2) providing the possibility for optimizing the locality of domains in such a way that the average latency between two peers in a domain is much lower than any two peers outside that domain. These two issues are the main topics of this thesis.

In the first part of this dissertation, in Chapter 3, we have studied the problem of hierarchically constructing DHTs in order to provide fast search operation, load balancing, fault isolation, and administrative autonomy. In order to achieve this, we have developed a generic framework which preserves all the properties of the original DHT such as degree, load balancing and fault-tolerance, but incorporates all the advantages of hierarchical design. We have shown that this can be done by finding the quasiminimal generating set of its Cayley graph representation. Moreover, we have applied our framework to provide a hierarchical version of Chord. We have also given some indicative hints of how to convert another six DHTs into hierarchical overlays. In the course of this work, we verified, theoretically and through simulations, that our construction of Chord is equivalent to Chord in maintenance and construction complexity and in addition, it can obtain significant savings in search latency.

It must be remarked that our hierarchical framework can be applied to solve other distributed problems. Such flexibility is facilitated by the use of clusters which can be visualized as smaller DHTs embedded into a large DHT. This property has many appealing applications. One promising application is in the field of security. There are many applications that may never profit from the possibilities of structured P2P overlays unless distributed security is developed. For instance, DHTs have only one routing path between the source and target peers. As a result, any malicious peer along this path may simply drop the message, even though this message is cryptographically secure. To address this issue, we have used our framework in [14] to provide multi-path routing between the source and target peers by means of multiple independent paths, i.e., paths that do not share any common peer other than the source and the target. Each path is of the same length and is *exclusively* composed of peers lying in the same cluster, of course, with the exception of the source and target peers. Thanks to multi-path routing, the probability of a routing failure can be then reduced by an exponential decay factor in the number of independent paths.

Other applications include Geographic Information Services (GIS) for the Internet such as the one we developed in [15], publish/subscribe systems and even Domain Name System (DNS).

In the second part of this thesis, in Chapter 4, we have formulated the question of what are the salient features of our hierarchical constructions compared with existing hierarchical designs. To answer this question, we have developed an analytic cost-based model to calibrate the potential gains of each design. Our strategy was not only to look at the graph-theoretic properties of each design, but also to compute the communication cost as a function of the locality in communication. Furthermore, we have compared the two main hierarchical DHT designs: the homogeneous design, in which all peers assume equal roles, against the superpeer design, in which a relatively small subset of peers behave as proxies for the other peers in the overlay. From this comparison, we have discovered that there is no a universally better design: superpeer designs could be worse than homogeneous designs and vice versa.

Finally, in Chapter 5, we have studied a central question to all hierarchical overlays: how to divide participating peers into low-latency clusters so that routing performance can be improved. In order to do this, we have seen that it is critical to mitigate false clustering, which occurs when distant peers are clustered together. Since none of the existing tools accounts for this fact, we have proposed: (1) a novel clustering algorithm deliberately aimed at minimizing false clustering; and (2) a novel metric to measure and compare the performance of our clustering algorithm against them. Specifically, our metric measures the fraction of falsely clustered peers in a clustering. We have shown through simulation that our clustering algorithm is superior to existing tools, with a false clustering rate inferior to 5%. These results illustrate that our algorithm can be successfully deployed in real environments because it is scalable and it provides accurate clusterings.

Overall, we believe that this dissertation provides all the necessary mechanisms to construct efficient content distribution networks such as Coral, which includes the adaptation to the underlying physical network, and many other wide-area applications.

6.1.1 Summary of Contributions

The following is the list of original contributions of this thesis.

i. Formal classification of peer-to-peer overlay networks

   - a classification of peer-to-peer overlays into flat and hierarchical structures and discussion of one prominent example of each class.
   - a conceptual model for homogeneous, hierarchical overlays.

ii. Hierarchical construction (*the construction module*)

   - a generic framework for the construction of homogeneous hierarchical DHTs based on the Cayley graph representation of their flat topologies.
   - a hierarchical construction for Chord, which inherits the load balancing property and logarithmic complexity of Chord, while enjoying the advantages of hierarchical design.
   - indicative hints of how six different DHTs — Randomized Chord, Symphony, Kademlia, P-Grid, Tapestry and Pastry — can be converted into their hierarchical versions.
   - Theoretic and experimental results that illustrate the improvement on search latency offered by our hierarchical designs.

iii. Comparative study of hierarchical designs (*the comparative module*)

   - an analytic cost model to identify the optimal hierarchical design for a given workload.
   - an analytic comparison of the two main hierarchical designs: the superpeer design and the homogeneous design.

iv. Addressing locality (*the clustering module*)

- an innovative metric called false clustering rate to assess the quality of proximity-based clustering.
- a deep understanding of the false clustering problem.
- a novel clustering algorithm that organizes peers into clusters so that peers in a cluster are closer — in terms of round-trip-time — to each other than peers not in their cluster.

6.2 Directions for future research

In the course of this work, several interesting questions have arisen, among which the following are of particular interest.

- The design of our framework assumes that all peers are homogeneous, an assumption that does not hold in the real world. A question of enormous interest for future work would be how to extend our framework to incorporate heterogeneity of peers. As noted in Chapter 3, our hierarchical construction of Chord has logarithmic diameter with logarithmic degree. If peers could maintain a number of connections in proportion to their capacities, it would be interesting to explore the trade-off between diameter and load balancing, as both properties are at odds with each other.

- Another unaddressed aspect of hierarchical DHTs is security. In superpeer systems, superpeers process and relay requests on behalf of other peers. This means that for an attacker to infringe the maximum damage all he/she has to do is to compromise as much superpeers as possible. In order to strengthen a system against this attack, how to elect the superpeers and how to establish trust among peers are critical issues. Designing completely distributed protocols to address these problems is a very challenging task.

- We have seen in Chapter 2 that objects are assigned to domains by means of mapping $F_{\mathcal{M}}$. As a consequence, the stability of a domain plays an important role in the scalability of the system. Failures and dynamic creation and deletion of domains could trigger an avalanche of object movements among domains. Moving all the objects from one domain to another might congest network links. For this reason, it would be interesting to include object movement in our cost model and determine which designs are more sensitive to such dynamics.

- Our clustering algorithm uses traceroute as a measurement tool. As mentioned in Chapter 5, a traceroute probe incurs message overhead proportional to the length of the probed path. An interesting future research would be to develop a smarter version of traceroute that does not probe every hop, but achieves equivalent accuracy in terms of false clustering rate. For example, a smarter version of traceroute could skip some intermediate hops. Which hops to skip is a challenging problem for future research.

References

[1] *A&M Records, Inc Vs. Napster Inc, United States District Court, Northern District of California*, 2001.

[2] Gnutella, http://gnutella.wego.com/ (2006).

[3] J. Ritter, *Why Gnutella Can't Scale. No, Really*, http://www.tch.org/gnutella.html (2001).

[4] I. Stoica et. al., *Chord: a scalable peer-to-peer lookup protocol for internet applications*, IEEE/ACM Transactions on Networking **11**, 17 (2003).

[5] A. Rowstron and P. Druschel, *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, in *Middleware'01:Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, 2001.

[6] P. Maymounkov and D. Mazieres, *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*, in *IPTPS'02: Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[7] R. Rivest, *The MD5 Message-Digest Algorithm*, 1992.

[8] r. D. Eastlake and P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, 2001.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, *A scalable content-addressable network*, in *SIGCOMM'01: Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, 2001.

[10] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz, *Tapestry: A Resilient Global-scale Overlay for Service Deployment*, IEEE Journal on Selected Areas in Communications **22**, 41 (2005).

[11] G. Manku, M. Bawa, and P. Raghavan, *Symphony: Distributed hashing in a small world*, in *USITS 2003: Proceedings of 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[12] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, *P-Grid: a self-organizing structured P2P system*, SIGMOD Rec. **32**, 29 (2003).

[13] P. Ganesan, K. Gummadi, and H. Garcia-Molina, *Canon in G Major: Designing DHTs with Hierarchical Structure*, in *ICDCS'04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 263–272, 2004.

[14] M. Sánchez-Artigas, P. García-López, and A. F. G. Skarmeta, *A Novel Methodology for Constructing Secure Multipath Overlays*, IEEE Internet Computing **9**, 50 (2005).

[15] J. P. Ahulló, P. García-López, M. Sánchez-Artigas, and A. F. G. Skarmeta, *Supporting Geographical Queries onto DHTs*, in *LCN'08: Proceedings of the 33rd IEEE Conference on Local Computer Networks*, 2002.

[16] S. B. Akers and B. Krishnamurthy, *A Group-Theoretic Model for Symmetric Interconnection Networks*, IEEE Transactions on Comput. **38**, 555 (1989).

[17] PlanetLab, http://www.planet-lab.org .

[18] H. Weatherspoon and J. D. Kubiatowicz, *Erasure coding vs replication: A quantitative comparison*, in *IPTPS'02: Proceedings of the 1st Intl. Workshop on Peer-to-Peer Systems*, 2002.

[19] J. Kleinberg, *The Small-World Phenomenon: An Algorithmic Perspective*, in *STOC'02: Proceedings of the ACM Symposium on the Theory of Computing*, 2000.

[20] L. H. A. J. B. Zhao, Y. Duan and J. Kubiatowicz, *Brocade: landmark routing on overlay networks*, in *IPTPS'02: Proceedings of the 1st Intl. Workshop on Peer-to-Peer Systems*, 2002.

[21] A. Mizrak, Y. Cheng, V. Kumar, and S. Savage, *Structured superpeers: leveraging heterogeneity to provide constant-time lookup*, in *Proceedings of the IEEE Workshop on Internet Applications*, 2003.

[22] L. Garces-Erice, E. W. Biersack, K. W. Ross, P. A. Felber, and G. Urvoy-Keller, *Hierarchical P2P Systems*, Euro-par 2003: Proceedings of the 2003 ACM/IFIP International Conference on Parallel and Distributed Computing (2003).

[23] L. Garces-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross, *Data Indexing in Peer-to-Peer DHT Networks*, in *ICDCS'04: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, pages 200–208, 2004.

[24] M. J. Freedman, E. Freudenthal, and D. Mazières, *Democratizing content publication with coral*, in *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 18–18, 2004.

[25] R. M. Z. Xu and Y. Hu, *HIERAS: A DHT based hierarchical p2p routing algorithm*, in *ICPP'03: Proceedings of the 2003 Intl. Conference on Parallel Processing*, pages 187–194, 2003.

[26] M. T. Schlosser, M. Sintek, S. Decker, and W. Nejdl, *HyperCuP - Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks*, in *AP2PC'02: Proceedings of the 1st Intl. Workshop Agents and Peer-to-Peer Computing*, pages 112–124, 2002.

[27] H. Shen, C.-Z. Xu, and G. Chen, *Cycloid: a constant-degree and lookup-efficient P2P overlay network*, Performance Evaluation **63**, 195 (2002).

[28] D. Ratajczak and J. M. Hellerstein, *Deconstructing DHTs*, Technical Report IRB-TR-03-042, Intel Research, 2004.

[29] H. Cohen, *A course in computational algebraic number theory*, Springer-Verlag New York, Inc., 1993.

[30] 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.

[31] C. Qu, W. Nejdl, and M. Kriesell, *Cayley DHTs - A Group-Theoretic Framework for Analyzing DHTs Based on Cayley Graphs*, in *ISPA'04: Proceedings of the 2nd Intl. Symposium on Parallel and Distributed Processing and Applications*, pages 914–925, 2004.

[32] M. Lupu, B. C. Ooi, and Y. C. Tay, *Paths to stardom: calibrating the potential of a peer-based data management system*, in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 265–278, 2008.

[33] W. Xiao and B. Parhami, *Cayley graphs as models of deterministic small-world networks*, Information Processing Letters **97**, 115 (2007).

[34] Z. Sun and W. Xiao, *ComNET: A P2P Community Network*, in *APPT'07: Proceedings of the 7th International Symposium on Advanced Parallel Processing Technologies*, pages 271–281, 2007.

[35] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R.Panigrahy, *Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on World Wide Web*, in *STOC'97: Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 654–663, 1997.

[36] M. C. Heydemann, *Cayley graphs and interconnection networks*, G. Hahn, G. Sabidussi (Eds.), Graph Symmetry (Montreal, PQ, 1996), NATO Advanced Science Institutes Series C, Mathematica and Physical Sciences **497**, 167 (1997).

[37] F. R. K. Chung, J. E.G. Coffman, M. Reiman, and B. Simon, *The forwarding index of communication networks*, IEEE Transaction on Information Theory **33**, 224 (1987).

[38] M. C. Heydemann, J. Meyer, and D. Sotteau, *On forwarding indices of networks*, Discrete Applied Maths **23**, 103 (1989).

[39] Y. Manoussakis and Z. Tuza, *The forwarding index of directed networks*, Discrete Appl. Math. **68**, 279 (1996).

[40] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh, *Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience*, in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 395–406, 2003.

[41] J. Xu, A. Kumar, and X. Yu, *On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks*, IEEE Journal on Selected Areas in Communications **22**, 151 (2004).

[42] G. Cordasco and A. Sala, *2-Chord Halved*, in *HOT-P2P'05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 72–79, 2005.

[43] V. King and J. Saia, *Choosing a random peer*, in *PODC'04: Proceedings of the 23th annual ACM symposium on Principles Of Distributed Computing*, pages 125–130, 2004.

[44] G. Plaxton, R. Rajamaran, and A. Richa, *Accessing nearby copies of replicated objects in a distributed environment*, Theory of Computing Systems **32**, 241 (1999).

[45] K. Calvert, M. Doar, and E. Zegura, *Modeling Internet topology*, Communications Magazine, IEEE **35**, 160 (1997).

[46] Y. Shavitt and E. Shir, *DIMES: let the internet measure itself*, SIGCOMM Comput. Commun. Rev. **35**, 71 (2005).

[47] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, *Vivaldi: a decentralized network coordinate system*, in *SIGCOMM'04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 15–26, 2004.

[48] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, John Wiley and Sons, New York, 1990.

[49] B. Karp, S. Ratnasamy, S. Rea, and S. Shenker, *Spurring Adoption of DHTs with OpenHash, a Public DHT Service*, in *IPTPS'04: Proceedings of the 3rd International Workshop on Peer-to-Peer Systems*, pages 195–205, 2004.

[50] Q. Z. B. L. B. Y. Z. R. Tian, Y. Xiong and X. Li, *Hybrid overlay structure based on random walks*, in *IPTPS'05: Proceedings of the 4th Intl. Workshop on Peer-to-Peer Systems*, 2005.

[51] S. Zoels, Z. Despotovic, and W. Kellerer, *On hierarchical DHT systems - An analytical approach for optimal designs*, Comput. Commun. **31**, 576 (2008).

[52] H. Beverly Yang, B.; Garcia-Molina, *Designing a super-peer network*, ICDE'03: Proceedings of the 19th International Conference on Data Engineering. , 49 (2003).

[53] N. Christin and J. Chuang, *A Cost-Based Analysis of Overlay Routing Geometries*, in *INFOCOM 2005: Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, 2005.

[54] V. Ramasubramanian and E. G. Sirer, *The design and implementation of a next generation name service for the internet*, in *SIGCOMM'04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 331–342, 2004.

[55] L. Xiao, Z. Zhuang, and Y. Liu, *Dynamic Layer Management in Superpeer Architectures*, IEEE Transactions on Parallel and Distributed Systems **16** (2005).

[56] M. Sánchez-Artigas, P. García-López, and A. F. G. Skarmeta, *On the Feasibility of Dynamic Superpeer Ratio Maintenance*, in *P2P'08: Proceedings of the 8th International Conference on Peer-to-Peer Computing*, pages 202–209, 2008.

[57] S. Saroiu, K. P. Gummadi, and S. D. Gribble, *A Measurement Study of Peer-to-Peer File Sharing Systems*, (2002).

[58] S. L. Johnson and C. T. Ho, *Optimum broadcasting and personalized communication in hypercubes*, IEEE Trans. Comput. **38**, 1249 (1989).

[59] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, *IDMaps: a global internet host distance estimation service*, IEEE/ACM Trans. Netw. **9**, 525 (2001).

[60] T. S. E. Ng and H. Zhang, *Predicting Internet network distance with coordinates-based approaches*, volume 1, pages 170–179, 2002.

[61] K. Gummadi, S. Saroiu, and S. Gribble, *King: Estimating latency between arbitrary Internet end hosts*, in *Proceedings of the Internet Measurement Workshop*, 2002.

[62] V. Paxson, *End-to-end routing behavior in the Internet*, SIGCOMM Comput. Commun. Rev. **26**, 25 (1996).

[63] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, *iPlane: an information plane for distributed services*, in *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 26–26, 2006.

[64] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee, *Estimating network proximity and latency*, SIGCOMM Comput. Commun. Rev. **36**, 39 (2006).

[65] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan, *Geographic locality of IP prefixes*, in *IMC'05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 13–13, 2005.

[66] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, *Topologically-Aware Overlay Construction and Server Selection*, in *INFOCOM 2002: Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1190–1199, 2002.

[67] J. Ledlie, P. Pietzuch, and M. Seltzer, *Stable and Accurate Network Coordinates*, in *ICDCS'06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, page 74, 2006.

[68] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft, *On the accuracy of embeddings for internet coordinate systems*, in *IMC'05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 11–11, 2005.

[69] J. Dunn, *Well separated clusters and optimal fuzzy partitions*, Journal Cybernetics **4**, 95 (1974).

[70] J. C. Bezdek and N. R. Pal, *Some new indexes of cluster validity, "*, IEEE Trans. Syst. Man, Cyber. -Part B **28**, 301 (1998).

[71] 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.

[72] L. C. Freeman, *A Set of Measures of Centrality Based on Betweenness*, Sociometry **40**, 35 (1977).

[73] R. Govindan and H. Tangmunarunkit, *Heuristics for Internet Map Discovery*, in *INFOCOM 2000: Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1371–1380, 2000.

[74] N. Spring, R. Mahajan, and D. Wetherall, *Measuring ISP topologies with rocketfuel*, SIGCOMM Comput. Commun. Rev. **32**, 133 (2002).

[75] Scriptroute, http://www.cs.washington.edu/research/networking/scriptroute .

[76] A. Lakhina, J. W. Byers, M. Crovella, and P. Xie, *Sampling biases in IP topology measurements*, in *INFOCOM 2003: Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 332–341, 2003.

[77] L. Dall'asta, I. Alvarez-Hamelin, A. Barrat, A. Vazquez, and A. Vespignani, *A statistical approach to the traceroute-like exploration of networks: theory and simulations*, 2004.

[78] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. T. Rowstron, *Scribe: a large-scale and decentralized application-level multicast infrastructure*, Selected Areas in Communications, IEEE Journal on **20**, 1489 (2002).

[79] A. Montresor, *A Robust Protocol for Building Superpeer Overlay Topologies*, in *P2P'04: Proceedings of the 4th International Conference on Peer-to-Peer Computing*, pages 202–209, 2004.

[80] B. H. r, D. Plummer, D. Moore, , and k. Claffy, *Topology Discovery by Active Probing*, in *SAINT-W'02: Proceedings of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*, page 90, 2002.

[81] B. Yao, R. Viswanathan, F. Chang, and D. G. Waddington, *Topology Inference in the Presence of Anonymous Routers*, in *INFOCOM 2003: Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 353–363, 2003.

[82] M. H. Gunes and K. Sarac, *Inferring subnets in router-level topology collection studies*, in *IMC'07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 203–208, New York, NY, USA, 2007, ACM.

[83] M. Faloutsos, P. Faloutsos, and C. Faloutsos, *On power-law relationships of the Internet topology*, SIGCOMM Comput. Commun. Rev. **29**, 251 (1999).

[84] A. L. Barabasi and R. Albert, *Emergence of scaling in random networks*, Science **286**, 509 (1999).

[85] D. Davies and D. Bouldin, *Cluster separation measure*, IEEE/ACM Trans. Pattern Analysis and Machine Intelligence **1**, 224 (1979).

[86] M. Kim and R. S. Ramakrishna, *New indices for cluster validity assessment*, Pattern Recogn. Lett. **26**, 2353 (2005).