UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

POSITION-BASED DISTRIBUTED HASH TABLES

Filipe João Boavida de Mendonça Machado de Araújo

DOUTORAMENTO EM INFORMÁTICA

2005

UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

POSITION-BASED DISTRIBUTED HASH TABLES

Filipe João Boavida de Mendonça Machado de Araújo

DOUTORAMENTO EM INFORMÁTICA

2005

Tese orientada pelo Prof. Doutor Luís Eduardo Teixeira Rodrigues

This work was partially supported by the Fundação para a Ciência e a Tecnologia through the project INDIQoS POSI/CHS/41473/2001 via POSI and FEDER funds and by the European Science Foundation MiNEMA Research Network.

Abstract

In this thesis we want to create scalable, fault-tolerant and self-configuring dictionaries that can be deployed in a wide range of networks, including highly dynamic networks with frequent membership changes, like peer-to-peer overlay networks or wireless *ad hoc* networks.

In recent years, distributed hash tables (DHTs) have emerged as a solution to implement large-scale dictionaries. However, given the existing bandwidth limitations, updating routing information in DHTs remains a challenge. Position-based routing schemes arise as an attractive solution to this problem, due to inexpensive and ubiquitous localization mechanisms. Positional information enables the creation of oblivious (or memoryless) routing schemes, where the coordinates of the current forwarding node, of its neighbors and of the destination, suffice to determine the next hop. Such routing schemes are very suitable to rapidly changing networks, because they require very little control information.

We argue in this thesis that we can use positional information to efficiently support routing and DHT operation in wireless *ad hoc* and in wired networks, whenever position of nodes reflects network topology. To support this claim, we create and evaluate a number of algorithms that simultaneously support routing and DHT operation in both types of networks.

KEY WORDS: Distributed Hash Table, Overlay Network, Position-Based Routing Scheme, Delaunay Triangulation, Long Range Contact.

Resumo Estendido

Um dicionário armazena valores que podem ser acedidos através de chaves associadas. Nos sistemas distribuídos actuais, os dicionários desempenham um papel central. Por exemplo, na Internet, quase todas as aplicações dependem de um dicionário chamado "Domain Name Service" (DNS). Esta tese aborda o problema da criação de dicionários escaláveis, tolerantes a faltas e auto-configuráveis, que possam ser aplicados numa vasta gama de redes, incluindo redes extremamente dinâmicas com mudanças frequentes de participantes. Dentro destas redes, destacam-se, apesar das suas diferenças, as redes lógicas (de *overlay*) entre-pares (com fios) e as rede *ad hoc* sem fios, por serem capazes de se auto-organizarem de forma fortemente descentralizada, mesmo em ambientes onde a participação é altamente dinâmica.

Uma concretização de uma tabela de dispersão distribuída (distributed hash table, DHT) sobre redes entre-pares emerge naturalmente como uma solução para concretizar um dicionário de grande escala. Actualmente há muitas concretizações de DHTs baseadas em redes lógicas entre-pares, especialmente em redes com fios, tais como o Pastry, Tapestry, Chord e CAN, para mencionar apenas algumas. No entanto, as limitações de largura de banda fazem com que a actualização de informação de encaminhamento permaneça muito difícil, não só em redes com fios, mas especialmente em rede *ad hoc* sem fios, onde a construção de uma rede lógica levanta problemas ainda mais complexos de eficiência. É neste contexto que o en-

caminhamento com tabelas de dispersão distribuídas baseadas em informação de localização surge como uma solução atractiva para o problema, devido, em muito, à vulgarização de mecanismos de localização baseados, por exemplo, no "Sistema de Posicionamento Global" (*Global Positioning System*, GPS). Para reduzir a informação de controlo e a memória necessária, o encaminhamento baseado em informação de localização, ou posicional, utiliza a posição geográfica dos nós para encaminhar as mensagens. Um esquema de encaminhamento posicional assume que *i*) os nós conhecem a sua própria posição geográfica, *ii*) os nós conseguem determinar a posição dos seus vizinhos, *iii*) os nós conseguem determinar a posição do destino e *iv*) a proximidade geográfica reflecte a proximidade topológica, de forma a garantir que o endereço dos nós codifica automaticamente parte da estrutura da rede. Em geral, um esquema de encaminhamento geográfico requer que cada nó construa a sua visão parcial do grafo, recorrendo apenas a informação de vizinhos próximos. Por uma questão de economia de recursos, é comum os nós só poderem recolher informação de vizinhos que estejam a um pequeno número de saltos de distância, sendo este número tipicamente de um ou dois saltos. Construído este grafo, que requer uma quantidade muito limitada de memória em cada nó ($O(1)$ em média), os nós podem utilizar um algoritmo de encaminhamento, onde as coordenadas do nó que faz o encaminhamento e do destino chegam para determinar o próximo salto. Este tipo de esquemas de encaminhamento requer muito pouca informação de controlo, sendo, por isso, perfeitamente adequado para redes em rápida mutação.

A tese que este trabalho defende é que, sempre que a proximidade geográfica entre nós reflicta a topologia da rede, é possível utilizar informação de localização na construção de DHTs eficientes, quer em redes com fios, quer em redes sem fios. A utilização de informação de localização de permite resolver simultaneamente o problema do encaminhamento e o suporte da DHT. Por outras palavras, não é sequer necessário que exista qualquer esquema de encaminhamento de suporte à DHT, podendo esta funcionar sem todos os protocolos associados ao IP. É de realçar, no entanto, que quando existe um esquema de encaminhamento subjacente, uma DHT com noção de localização pode enriquecer aplicações entre-pares com serviços como difusões ou interrogações limitadas geograficamente. Além disto, em redes com fios onde o encaminhamento IP esteja disponível, é possível enriquecer uma DHT baseada em informação de localização com contactos de longa distância, para reduzir o comprimento dos percursos. Por estas razões, o encaminhamento baseado em informação de localização pode ser usado como um componente fundamental para uma DHT eficiente, tanto em redes *ad hoc* sem fios, como em redes com fios. Para validar esta tese, foram criados e avaliados um conjunto de algoritmos, que são descritos brevemente de seguida, capazes de suportar simultaneamente o encaminhamento e a DHT em ambos os tipos de rede:

- uma triangulação de nome "Fast Localized Delaunay triangulation" (FLDT), que cria e mantém um grafo bem conhecido de nome "Planar Localized Delaunay Triangulation", $PLDel(V)$, em redes *ad hoc* sem fios dinâmicas. O grafo $PLDel(V)$ apresenta um número de características favoráveis: pode ser construído de forma localizada, o seu custo de construção é baixo e suporta algoritmos de encaminhamento que garantem a entrega de mensagens. Embora apresente um custo de comunicação semelhante ao de trabalhos anteriores ($O(n \log n)$ no total, sendo n o número de nós existentes), a constante escondida no algoritmo FLDT é muito mais baixa, além de necessitar apenas de uma única mensagem por nó, em vez de quatro ou mais. O baixo custo do nosso algoritmo torna a utilização deste grafo viável em sistemas reais, em substituição de outros grafos como o grafo de Gabriel (GG) ou o grafo de vizinhanças relativas (RNG), que não permitem um bom desempenho do algoritmo de encaminhamento;

- uma DHT que utiliza um mecanismo de agrupamento baseado em posição, para redes *ad hoc* sem fios, chamado “Cell Hash Routing” (CHR). A maior inovação do CHR está no facto de utilizar um grafo virtual extremamente regular para fazer o encaminhamento, com base em grupos de nós. Este agrupamento resolve automaticamente muitos dos problemas associados a esquemas de encaminhamento posicionais, por exemplo, eliminação de arestas que se intersectam. A concretização da DHT CHR utiliza um esquema de encaminhamento localizado e permite equilibrar a distribuição de de carga pelos nós. Isto torna o CHR altamente escalável relativamente não só ao tamanho da rede, mas também à densidade dos nós (*versus* alcance da comunicação). Estas razões fazem-nos pensar que o CHR é uma adaptação promissora do conceito de DHT às rede *ad hoc* sem fios;

- GeoPeer, um sistema entre-pares com noção de localização, que permite concretizar uma DHT baseada em informação de localização para redes com fios. Esta DHT constrói uma triangulação de Delaunay completa. Por esta razão, pode ser vista como a arquitectura complementar para redes com fios da triangulação criada pelo algoritmo FLDT. Esta DHT pode ser utilizada para suportar aplicações com exigências de QoS, além de outros serviços baseados em posição, tais como difusões ou interrogações limitadas geograficamente;

- o mecanismo “Hop Level”, que cria e mantém contactos de longa distância numa rede com fios. Este mecanismo procura reduzir o número de saltos que é necessário percorrer numa rede lógica. O mecanismo “Hop Level” melhora consideravelmente o desempenho da rede GeoPeer em redes lógicas com mudanças frequentes dos seus elementos, desde que estas funcionem em cima duma rede IP. Um dos aspectos mais interessantes do

mecanismo “Hop Level” é que pode ser utilizado em sistemas de armazenamento distribuídos (Distributed Storage Systems, DSSs) para suportar de forma eficiente interrogações em gamas multi-dimensionais de atributos. Por esta razão a sua utilidade vai um pouco além da funcionalidade típica duma DHT;

O trabalho realizado nesta tese levanta perspectivas interessantes, que poderão ser abordadas como temas de trabalho futuro. Por exemplo, um dos aspectos a considerar é a utilização de modelos de comunicação mais realistas para redes *ad hoc* sem fios. Em particular, os mecanismos que servem de base à DHT CHR poderão ser facilmente estendidos com vista a esse fim. A utilização de posição poderá servir também de base a arquitecturas com suporte de parâmetros de qualidade de serviço, tais como latência e largura de banda ou para seleccionar melhores localizações para outros recursos partilhados por muitos nós da rede (por exemplo, “núcleos” de árvores de difusão ou nós de “contacto” para sistemas editor/assinante). Finalmente, outra possibilidade interessante criada por este trabalho consiste na possibilidade de construir uma arquitectura integrada que combine DHTs para redes com e sem fios. Como elo de ligação entre os dois tipos de redes poderão ser utilizadas as soluções baseadas em triangulações de Delaunay para ambos os tipos de redes. A utilização da posição numa arquitectura como esta simplificaria consideravelmente a integração e utilização transparentes da DHT, independentemente do tipo de rede de acesso dos nós.

PALAVRAS-CHAVE: Tabela de Dispersão Distribuída, Rede Lógica, Esquema de Encaminhamento Baseado em Informação de Localização, Triangulação de Delaunay, Contacto Distante.

Acknowledgments

My first words of gratitude are to my advisor, Professor Luís Eduardo Teixeira Rodrigues of the Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa (DI-FCUL). He has always been able to encourage me with his high standards of quality and his endless motivation. I owe him everything that I learned about scientific creation during my PhD. More than an advisor, he has also been a friend that allowed me to do my PhD and still keep on with my familiar obligations.

I am grateful to the DIstributed ALgorithms and Network Protocols (DIALNP) research group, to the Laboratório de Sistemas Informáticos de Grande Escala (LaSIGE) and to the DI-FCUL for the outstanding conditions that I enjoyed during this work. I am also grateful to the people of DIALNP, LaSIGE and DI-FCUL, in particular, to Nuno Carvalho and to my office mates, António Ferreira, Carlos Duarte and Hugo Miranda, for the opinions and valuable discussions that we had.

Devo também uma palavra especial de agradecimento ao meu tio Luís Alberto e à Teresa. Sem a sua ajuda e generosidade nunca poderia ter conjugado este trabalho de doutoramento com a minha vida familiar da forma que o fiz.

À Carlinha, ao João e ao Zé, agradeço e peço desculpa por tantas e tantas ausências. Pelas dificuldades que passaram a eles dedico esta tese.

Agradeço também a ajuda dos meus pais, dos meus sogros, da avó, da Alcina, do Camilo, do Luís e do Rui pelas tantas vezes que nos ajudaram a cuidar dos meninos.

Finalmente, não posso deixar de agradecer aos meus pais tudo aquilo que sempre fizeram por mim.

Lisboa, Setembro de 2005

Filipe João Boavida de Mendonça Machado de Araújo

À Carlinha, ao João e ao Zé.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

A dictionary stores values that can be accessed by associated keys. Dictionaries play a role of paramount importance in modern distributed systems. For instance, in the Internet, most applications depend on a dictionary called "Domain Name Service" (DNS). Implementation of a dictionary should be scalable, fault-tolerant and self-configuring. The peer-to-peer interaction model naturally emerges as a response to these demanding requirements. A dictionary implemented as a peer-to-peer distributed hash table (DHT) can be made scalable, self-configuring and can have low congestion. Furthermore, fault-tolerance and availability are almost inherent to this model. Currently, there are many implementations of DHTs, like Pastry (Rowstron & Druschel, 2001), Tapestry (Zhao *et al.*, 2001), Chord (Stoica *et al.*, 2001) and CAN (Ratnasamy *et al.*, 2001), just to name a few. As a consequence of their nearly optimal or even optimal path length/node degree trade-off, these DHTs are effectively able to deliver most of the promises held by the peer-to-peer interaction model. As a consequence, we can expect DHTs to have an increasingly important role in the construction of distributed applications, like naming services, file sharing applications or publish/subscribe systems.

Unfortunately, while there are many DHTs targeted for wired networks, there are only a few targeted for wireless *ad hoc* networks. This results from the difficulty of routing in wireless *ad hoc* networks. Without efficient routing it is impossible to support a DHT. Moreover, DHTs for wired networks use overlay logical networks. These overlays work under an independent logic of the physical network, thus being unsuitable for wireless *ad hoc* networks. Interestingly, there are some similarities between peer-to-peer overlay networks and wireless *ad hoc* networks, because they both are decentralized and self-organizing. Additionally, both of them have rapidly changing topologies. Hence, despite their differences, solutions for both types of networks may share some principles.

The faster the topology changes, the worse it is to make routing work, given the same computing power, communication latency and available bandwidth. In this scenario, cheap localization mechanisms, based, for instance on Global Positioning System (GPS) receivers (Kaplan, 1996), turned position-based routing schemes into an attractive solution to the routing problem. Position-based routing schemes assume that *i*) nodes are aware of their own geographical position, *ii*) nodes can determine the position of their neighbors and *iii*) nodes can determine the position of the destination. Position-based routing also stands on the additional assumption that geographical proximity reflects topological proximity. Given these conditions, nodes forward messages by comparing the position of the destination with their own position and with the positions of the neighbors. In these routing schemes, the coordinates of the current forwarding node, of its (few) neighbors and of the destination, suffice to determine the next hop. This makes such schemes very suitable to rapidly changing networks, because they require little control information.

Positional information allows us to simultaneously route messages and support the DHT operation. Additionally, position-based schemes scale very well, because the scope of the control information does not need to grow with the size of the system. Hence, following from the work of Ratnasamy *et al.* (2002) and Bose & Morin (1999), we argue that we can use positional information to implement efficient DHTs. Although this solution is particularly well suited for wireless *ad hoc* networks, from a theoretical point of view, it can also benefit wired networks, in networks where position of nodes strongly matches topology. A position-based DHT does not strictly require the heavy weight routing protocols associated with the IP network, because knowledge of distant nodes is not necessary. Nevertheless, even when using IP, the notion of location can enrich peer-to-peer applications with services like geographically-scoped multicasts or queries. Also, to reduce the distances in terms of hops in a wired network, we can augment a position-based DHT with a set of long range contacts. This takes advantage of the underlying infrastructure to reduce path lengths between distant neighbors. Therefore, we can use position-based routing as a fundamental component of an efficient DHT, not only in wireless but also in wired networks.

1.1 Problem Statement and Objectives

The goal of this thesis is to create DHTs for fast changing networks (wired or wireless), where routing between distant nodes is not available and network as well as node resources are scarce, given the pace of change of the network. Such DHTs should own the following properties: *i*) high scalability of memory requirements, path lengths and congestion up to millions of nodes for wired networks, but possibly two or more fewer orders of magnitude in the wireless case; *ii*) high fault-tolerance and availability;

and finally *iii*) self-configurability, as human intervention should be unnecessary. In this thesis we shall focus our work on networks where the following properties hold:

- nodes are able to determine their own geographical position. Furthermore, nodes can obtain positional information of neighbors and of the destination. To know their own location, nodes may either use a GPS-like system (Kaplan, 1996)[1] if they are outdoors, or they can use a triangulation or trilateration scheme (Hu & Evans, 2004; Niculescu & Nath, 2004; Haeberlen *et al.*, 2004). Alternatively, Rao *et al.* (2003) and Wattenhofer *et al.* (2005) follow approaches where nodes use logical instead of geographical positions, thus precluding the need for external positioning devices. In any case, to determine the position of the neighbors, nodes must exchange positional information among them. Determination of the position of the destination is orthogonal to our problem. In the context of a DHT, destination is determined by the hash function;

- the geographical position of the nodes reflects the topology of the network. Although this assumption is controversial in wired networks, Padmanabhan & Subramanian (2001) show that in the Internet there is a strong correlation between physical and topological distances. Furthermore, authors believe that this correlation will increase as the Internet is expected to become more richly connected.

[1]See also the web page of U.S. Naval Observatory (USNO) GPS Operations: http://tycho.usno.navy.mil/gps.html.

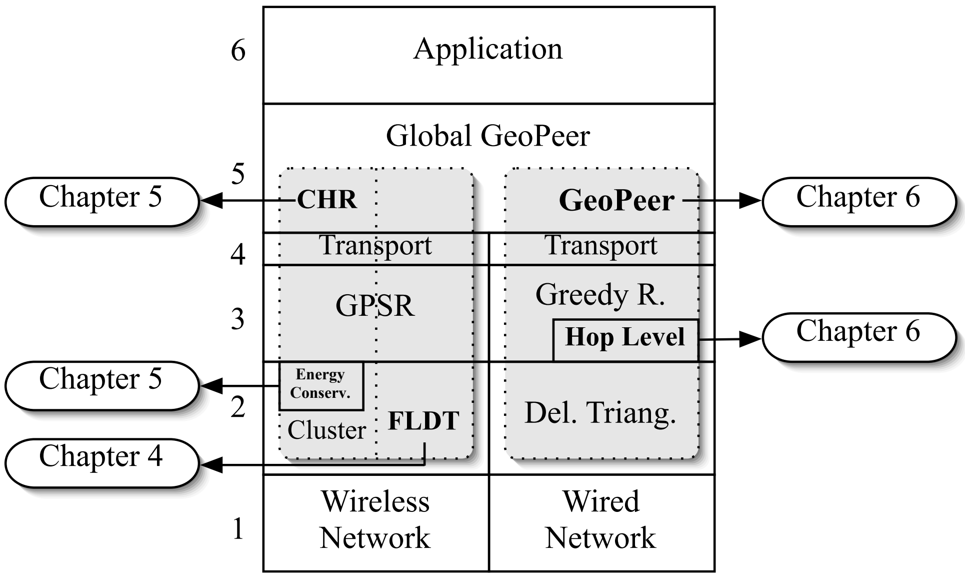


Figure 1.1: Overview of the proposed architecture

1.2 Results

In this thesis, we present a complete multi-platform architecture for both wireless and wired networks. The global view of this architecture is depicted in Figure 1.1, which is divided in six levels. In this thesis we focus on the routing layer (levels 2 and 3 of the figure) and on the application layers (levels 5 and 6 of the figure). In the figure, we include the chapter where we present each of the contributions.

For wireless *ad hoc* networks, we developed two different DHTs: a triangulation algorithm, called "Fast Localized Delaunay Triangulation" (FLDT), and an algorithm that clusters nodes in a regular grid, which is part of a DHT called "Cell Hash Routing" (CHR). In both cases, we use either the Greedy Perimeter Stateless Routing (Karp & Kung, 2000) (GPSR) or a variation as the routing algorithm. In wired networks, we developed a DHT called GeoPeer. This is also a position-based DHT, which uses a

Delaunay triangulation underneath and the simple greedy routing algorithm (Finn, 1987). This DHT can also make use of a mechanism called "Hop Level" that works on top of IP and that creates and maintains long range contacts in overlay networks. An interesting open possibility that results from the work of this thesis is the integration of wired and wireless DHTs based on Delaunay triangulations into a single DHT. We call "Global GeoPeer" to this DHT and we include it in the figure to complete the global view of our architecture. We depict in gray color the aforementioned results of this thesis. Contributions, to be described in the next section, are written in bold.

1.3 Contributions

The main contributions of this thesis are:

- a triangulation algorithm called "Fast Localized Delaunay triangulation" (FLDT) that, unlike previous work, requires a single control message in dynamic wireless *ad hoc* networks, to create and maintain a well-known graph called "Planar Localized Delaunay Triangulation", *PLDel*;

- a position-based clustering mechanism for wireless *ad hoc* networks, called "Cell Hash Routing" (CHR). The main novelty of CHR is the use of a logical graph of clusters, where routing takes place. This automatically solves many problems associated with position-based routing algorithms, like elimination of edges that intersect;

- a technique to conserve energy of nodes under the presence of failures in networks clustered like CHR;

- GeoPeer, a position-based DHT for wired networks, which can be used to support QoS applications as well as other position-based services, like geographically-scoped multicasts and queries;

- the "Hop Level" mechanism that creates and maintains long range contacts in wired networks. Hop Level aims to reduces the number of hops jumped in an overlay network. We can use Hop Level to improve performance of GeoPeer in overlay networks with highly dynamic membership that already have operational routing underneath. Furthermore, it can efficiently support multidimensional range queries in Distributed Storage Systems (e.g. Harvey *et al.*, 2003; Aspnes *et al.*, 2004; Bharambe *et al.*, 2004; Karger & Ruhl, 2004).

1.4 Outline of the Thesis

Chapters 2 and 3 motivate the work by introducing the two central issues of this thesis and by pointing out the limitations of current work. In Chapter 2, we overview position-based routing and, in Chapter 3, we overview current work on DHTs.

In Chapter 4, we present the "Fast Localized Delaunay Triangulation" algorithm, for wireless *ad hoc* networks. FLDT is a position-based preprocessing algorithm that creates a graph that lies underneath the routing algorithm. We include the formal proof of correctness and experimental results that show the validity of our approach.

In Chapter 5, we present the "Cell Hash Routing" (CHR) DHT. CHR is a position-based DHT for wireless *ad hoc* networks. It uses clustering and a virtual graph to greatly simplify the routing scheme. In addition, we study techniques to increase battery lifetime in similarly clustered networks.

In Chapter 6, we present a complete position-based DHT for wired networks. This chapter contains two main parts. A peer-to-peer network based on a complete Delaunay triangulation, called "GeoPeer" and a complementary mechanism called "Hop Level" to create and maintain long range contacts (LRCs). Although the utilization of Hop Level is not limited to GeoPeer, we use GeoPeer to do the experimental evaluation.

Finally, Chapter 7 outlines conclusions and points directions for future work.

1.5 Related Publications

We have previously published parts of this work in the following conferences and workshops:

- Filipe Araújo and Luís Rodrigues. GeoPeer: A location-aware peer-to-peer system. In *The 3rd IEEE International Conference on Network Computing and Applications (NCA '04)*, pages 39–46, Cambridge, MA, USA, August 2004.

  This paper first presented GeoPeer.

- Filipe Araújo and Luís Rodrigues. Fast localized Delaunay triangulation. In *The 8th International Conference On Principles Of Distributed Systems (OPODIS '04)*, pages 81–93, Grenoble, France, December 2004. Springer-Verlag, LNCS 3544.

  This paper first presented the FLDT algorithm.

- Filipe Araújo, Luís Rodrigues, Jörg Kaiser, Changling Liu, and Carlos Mitidieri. CHR: a distributed hash table for wireless ad hoc networks. In *The 25th IEEE International Conference on Distributed Computing Systems Workshops (DEBS '05)*, Columbus, Ohio, USA, June 2005.

This paper first presented CHR.

- Filipe Araújo and Luís Rodrigues. Long range contacts in overlay networks. In *Euro-par 2005*, pages 1153–1162, Lisbon, Portugal, August 2005. Springer-Verlag, LNCS 3648.

  This paper presents the Hop Level mechanism.

- Filipe Araújo and Luís Rodrigues. On the monitoring period for fault-tolerant sensor networks. In *Second Latin-American Symposium on Dependable Computing (LADC '05)*, October 2005. (to appear).

  This paper studies the problem of increasing the tolerance to faults in schemes that aim to maximize lifetime of a sensor network, given that energy consumption is significant and nodes can also fail for a number of other reasons. We do this analysis in a context of a clustering similar to CHR.

1.6 Additional Publications

Besides the work that is central to this thesis, we also explored the possibility of using DHTs to create publish/subscribe systems with quality of service (QoS) parameters, such as latency and bandwidth. To support these publish/subscribe systems, the DHTs must have a notion of QoS. For lack of space we do not detail the work related to this idea, but the interested reader is referred to the following publications:

- Filipe Araújo and Luís Rodrigues. On QoS-aware publish-subscribe. In *The 22nd IEEE International Conference on Distributed Computing Systems Workshops (DEBS '02)*, pages 511–515, Vienna, Austria, July 2002.

  This position paper addresses the issue of supporting QoS parameters in distributed publish/subscribe systems. It advocates that QoS parameters should be handled using the same constructs as other event information, such as their type or content.

- Nuno Carvalho, Filipe Araújo, and Luís Rodrigues. IndiQoS: um sistema publicação-subscrição com Qualidade de Serviço. In *6a Conferência sobre Redes de Computadores (CRC '03)*, Bragança, Portugal, Setembro 2003.

  This paper describes an approach to create a publish/subscribe system with QoS, called IndiQoS.

- Nuno Carvalho, Filipe Araújo, and Luís Rodrigues. Scalable QoS-based event routing in Publish-Subscribe Systems. In *The 4th IEEE International Conference on Network Computing and Applications (NCA '05)*, Cambridge, MA, USA, July 2005.

  This paper describes an implementation of the IndiQoS system using multiple rendezvous nodes on top of a DHT.

2 Survey on Position-Based Routing

2.1 Overview

The goal of routing is to deliver a packet from a source node S to destination node D in a network of nodes (which we can represent as a graph). To solve the routing problem, nodes of the network execute a distributed "routing scheme". A routing scheme is comprised of two parts (see, for instance, Fraigniaud & Gavoille, 2002): *i*) a distributed algorithm, here known as the routing algorithm, running at every node, which is responsible for determining the output port (i.e., the next hop) of a packet; and *ii*) a pre-processing algorithm that, given the initial connection graph G, must create whatever information is necessary to the routing algorithm (e.g., routing tables or a subgraph of G). In the worst case, optimal routing schemes may require as much as $O(n \log n)$ memory space at each node. For this reason, there are many "compact routing schemes", which try to reduce these requirements, for instance, by re-arranging the identification of nodes (van Leeuwen & Tan, 1995). Position-based routing can be seen as a form of compact routing in which nodes receive identifications that depend on their geographical positions. Usually, in literature, authors assume that the identification of a node is precisely the position that it occupies in the $\mathbb{R}^2$ plane. Since a packet includes the geographical position of the destination, nodes will know in which direction to forward it. On the contrary, in IP networks, nodes perform routing using protocols that derive from Dijkstra (1959) or Bellman-Ford algorithms (Bellman, 1957; Jr. & Fulkerson, 1962).

2.1.1 Advantages of Position-Based Routing

For the sake of scalability, we will focus on single-path position-based routing schemes, i.e., we do not allow nodes to create additional instances of the packet they are forwarding. Additionally, we assume that position of nodes encodes topological information, i.e., in general, nodes that are geographically close are also topologically close. This assumption allows nodes to avoid extensive topology updates, thus saving precious resources. This is especially important in fast changing networks, like wireless *ad hoc* networks or peer-to-peer networks. For instance, in wireless *ad hoc* networks, it is difficult to determine the underlying topology for two basic reasons: *i*) it may change too fast, thus generating too many global update messages that will consume available battery power and bandwidth; and *ii*) nodes have limited memory and usually they will not be able to store all the topology even if they could collect it. According to several experimental work (e.g. Jain *et al.*, 1999; Li *et al.*, 2000), routing schemes that do not use positional information and that are based on the exchange of routing tables, like DSDV (Perkins & Bhagwat, 1994), AODV (Perkins, 1997) and DSR (Johnson & Maltz, 1996), do not scale. Additionally, Jain *et al.* (1999) showed that routing table size grows linearly for the DSDV algorithm, while it grows logarithmically for a comparable position-based routing scheme (see also Stojmenovic, 2002). These limitations may also affect wired nodes, namely in peer-to-peer networks, where topology also tends to change very fast.

2.2 Assumptions

Position-based routing stands on top of a number of assumptions, including the following. The most important one is that nodes can determine their own position. To get positional information, nodes can use a Global Positioning System (GPS) receiver, if they are outdoors[1]. In alternative, wireless nodes can use techniques based on signal strength information, available in the standard IEEE 802.11 technology (Niculescu & Nath, 2004; Haeberlen *et al.*, 2004). For wired nodes there are also some attempts to provide a mapping service capable of returning a position given the IP address of a node (Padmanabhan & Subramanian, 2001). A second assumption is that nodes can determine location of their neighbors. This usually implies the exchange of a small number of packets between neighboring nodes to make their own positions available to others. Final assumption states that nodes can determine the position of the destination. Reasonability of this assumption depends on the concrete network. Usually, the problem of determining the network address of the destination is separated from the routing problem (take the Domain Name Service, DNS, for example). However, in practice, it may be difficult to use a different layer to provide a location service atop of, for instance, a wireless network. For this reason, there are solutions that integrate the routing with the location problem, e.g., the Grid Location Service (Li *et al.*, 2000). Interestingly, in the case of distributed hash tables (DHTs), this service can be provided by the

[1]A completely different approach is followed by Rao *et al.* (2003) and Wattenhofer *et al.* (2005) that use logical instead of geographical positions.

DHT itself and therefore, the assumption is perfectly reasonable. Henceforth, we focus on the problem of sending a packet from source node S to destination node D, considering that both of them have known locations.

2.3 Definitions

2.3.1 Notation

We will use the following conventions for notation:

- nodes (also designated as vertices) will be represented with capital letters, for instance, node A;
- the set of all nodes of the graph is designated as V;
- edges are represented by the two nodes that define them, for instance, node A and node B may define edge AB;
- distance between two nodes A and B will be represented by $||AB||$;
- a triangle defined by nodes A, B and C is represented by $\triangle ABC$;
- the circumcircle of $\triangle ABC$ is represented as $\bigcirc ABC$;
- the circle whose diameter is defined by two nodes A and B is represented by $d(A,B)$;
- the circle centered at A with ray r is represented as $b(A,r)$ (ball centered at A with ray r);
- an angle between line segments AB and AC defined at A is interchangeably represented by $\angle BAC$ or $\angle CAB$ — the vertex where the

angle is measured stays in the middle, while the position of remaining vertices is arbitrary. Unless stated otherwise, this angle is always $< \pi$.

Additionally, we will introduce in the following sections a number of models and definitions which are necessary to understand existing position-based pre-processing and routing algorithms.

2.3.2 Unit Disk Graph

Given the set of wireless nodes V, the Unit Disk Graph model (UDG) assumes that *i*) communication range of nodes is a perfect circle in $\mathbb{R}^2$; and *ii*) all nodes have the same communication range. Hence, the Unit Disk Graph of V, $UDG(V)$, is comprised of all edges not longer than maximum communication range between all pairs of nodes. Assuming that communication range is 1, graph $UDG(V)$ includes *all* possible edges whose length is at most 1 (also known as short edges as opposed to long edges which are longer than 1).

Given this definition, every node Q that is inside $b(P,1)$ is a neighbor of P, i.e., $Q \in N(P)$. Alternatively, Q is a 1-hop neighbor of P. If $Q \in N(P) \vee \exists C \in V : C \in N(P) \wedge C \in N(Q)$, Q is said to be a 2-hop neighbor of P, or simply 2-neighbor of P, and vice-versa. This definition can be extended to define neighborhoods between nodes that are separated by an arbitrary number of hops. Hence, two nodes that can reach each other in k or fewer hops are considered to be k-neighbors.

Henceforth, we will usually assume that we are using the UDG model. This model is more adequate to describe wireless *ad hoc* networks than to describe wired networks. Hence, the algorithms that depend on this model can only be used in wireless environments. This is especially important in the case of the pre-processing algorithms, which we present in Sections 2.5 and 2.6. The reader can find a related connectivity model of Barrière *et al.* (2002).

2.3.3 Localized Routing Scheme

To conserve resources, one often requires routing schemes to be localized. A routing scheme is localized if there is a constant n such that a node only uses *i*) its own information, *ii*) information of neighbors that can be reached in up to n hops and *iii*) information of a constant k number of additional nodes (see Kranakis *et al.*, 1999). A routing scheme is said to be n-localized if n is the smallest constant that satisfies the above condition. Occasionally, to simplify, we refer to "localized routing algorithms", but what we really mean is that the pre-processing algorithm and the distributed routing algorithm are both localized.

2.3.4 Spanning-ratio

Although creating economical routing schemes is very important, ensuring good performance is not less important. Hence, one criterion used to evaluate the pre-processing step of a routing scheme is the *quality* of the subgraph created. We will call G to the initial connectivity graph and H to the subgraph created by the pre-processing algorithm. As we will show ahead, this subgraph H allows the routing algorithm to converge. H is said to be a "topological t-spanner of G" if and only if:

$$\underset{\forall S,D\in V}{max} \left\{ \frac{\|\Pi_H(S,D)\|}{\|\Pi_G(S,D)\|} \right\} \leq t$$

This means that for all nodes S and D, shortest path between S and D in H, $\Pi_H(S,D)$, is at most t times longer than in G, $\Pi_G(S,D)$. t is known as the "length stretch factor". In a sense this factor indicates the quality of the subgraph. The smaller it is, the better the subgraph is.

When the graph G is the complete Euclidean graph determined by V, the above expression defines an "Euclidean t-spanner".

2.3.5 Competitive-ratio

The reader should notice that the spanning-ratio of a subgraph is only a bound to the performance of a routing scheme. We still need to develop routing schemes that select paths which are *close* to the shortest path. The "competitive-ratio" is used as an accurate measure of the quality of the routing scheme (RS) and is defined as follows:

$$\text{competitive-ratio(RS)} = \max_{\forall S,D \in V} \left\{ \frac{\|A_G(S,D)\|}{\|\Pi_G(S,D)\|} \right\}$$

$\|A_G(S,D)\|$ is the length of the shortest path between S and D, found by the routing scheme A in graph G; $\|\Pi_G(S,D)\|$ is the length of the shortest path, between the same pair of nodes, existing in G. A routing scheme is said to be "t-competitive" if its competitive-ratio is t. Again, this is a worst-case definition, because the competitive-ratio is determined by the pair of nodes for which results are worst. Often, the name "stretch factor" is also used instead of "competitive-ratio".

One interesting known fact is that no localized routing scheme can be t-competitive for any constant t. Kuhn *et al.* (2002) showed that if c is the cost of the best path for a given pair of nodes, the cost for any localized position-based routing scheme can grow to $\Omega(c^2)$. More precisely, any de-

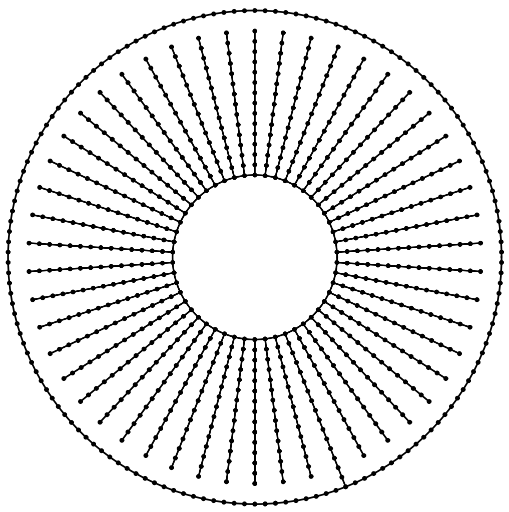


Figure 2.1: Variation of the lower bound graph

terministic (randomized) position-based routing scheme has a (expected) cost of $\Omega(c^2)$. Furthermore, this applies to the number of links traversed, to Euclidean distance or to energy spent transmitting the packet. To understand the reason for this, the reader should refer to Figure 2.1, which shows a variation of the "lower bound graph". The dots represent the nodes, while the lines represent the links between the nodes. In a wireless network, the distance between nodes of the inner circumference is precisely 1. Therefore, these nodes are connected. However, the other nodes of the radial lines can only communicate with the immediate neighbors in their own radial. Another key aspect of this graph is that there is only one radial that gives access to the outer circumference. If the shortest path (e.g., in number of links) to some node in the outer circumference is c, a localized routing scheme may end up using $\Omega(c^2)$ links to reach the only node that gives way to that outer circumference.

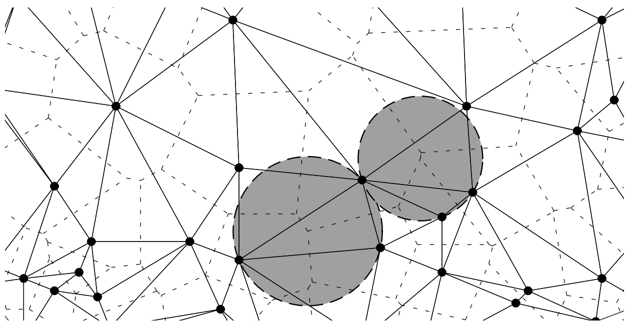


Figure 2.2: Delaunay Triangulation, Voronoi tessellation and the empty circumcircle

2.3.6 Delaunay Triangulation

The *Delaunay triangulation* (DT) of a node set V, also represented as $Del(V)$, is the set of edges satisfying the "empty circle" property: edge AB belongs to the triangulation if and only if there is a circle containing A and B, but not containing any other node. An important property of $Del(V)$ that we exploit in the thesis states that the circumcircle of a triangle does not contain any node of V. To this property we call the "empty circumcircle" property. The DT has an associated dual concept called the "Voronoi tessellation". The Voronoi tessellation partitions the space into convex polytopes in the following way. Given a node set V, the polytope of node N is comprised of the points that are closer to N than to any other node of V. In this thesis we will restrict ourselves to two-dimensional spaces. Therefore, we call simply "cell" to the Voronoi polygon of node N. Figure 2.2 shows the relation between the Voronoi tessellation (dashed lines) and the Delaunay triangulation (solid lines): two nodes share a Delaunay edge if and only if their Voronoi cells have a common border. We can also see the empty circumcircle property for two triangles, as no fourth node is inside the depicted circumcircles.

Although Delaunay triangulations have many other applications (e.g.

Computer Graphics), we are interested in their advantages for routing purposes (Bose & Morin, 1999; Li *et al.*, 2002; Gao *et al.*, 2001; Wang & Li, 2002; Lan & Wen-Jing, 2002).

2.4 Routing Algorithms

In this Section we briefly overview some of the most important position-based routing algorithms. All of these algorithms assume that nodes have only a partial view of the graph, such that they must relay packets to the neighbors, to reach a distant and unseen destination. At this moment we do not care about the pre-processing algorithm that nodes use. We postpone this issue to the following sections, as the pre-processing algorithms are strongly dependent of the kind of network that nodes use, either wired or wireless.

2.4.1 Basic Position-Based Routing Algorithms

In the following descriptions we assume that the algorithms are executed at node S, packets are destined to node D and there is a graph that determines the neighbors of node S. This graph must be created by a pre-processing algorithm, before the node starts forwarding packets. In the case of wireless networks, this resulting graph is typically a subgraph of UDG.

Greedy

Greedy routing algorithm (Finn, 1987) is a memoryless algorithm (only requires information about destination). When using greedy forwarding, a node selects for the next hop, the neighbor closest to destination. It is

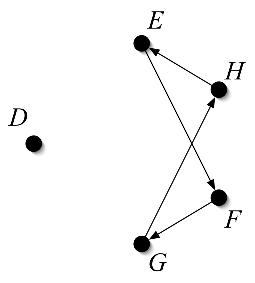


Figure 2.3: Compass routing can create a loop between nodes E, F, G and H

easy to come up with examples where this algorithm does not converge, due to local minima that occur in regions void of neighbors.

Compass

Consider the angle formed by line segments *SN* and *SD*, where *S* is the forwarding node, *N* is a potential next hop and *D* is the destination. The compass routing algorithm (Kranakis *et al.*, 1999), forwards packets to the neighbor *N* that forms the smallest angle $\angle NSD$ with the destination. Compass routing algorithm is also memoryless.

In the work of Stojmenovic & Lin (2001), we can find Figure 2.3, which shows an example where the compass routing algorithm may fall in a loop, between nodes E, F, G and H. Consider that the packet is at node E. Neighbors of E are F and H, being F the node that has minimizes the angle with D at E, i.e., $\angle FED$. Then, F forwards the packet to G, which, in turn, forwards the packet to H, which sends it back to E, thus forming a loop. Hence, the compass routing algorithm is not loop-free.

Randomized Compass

Randomized Compass routing algorithm (Bose & Morin, 1999) is a variation of the compass algorithm that avoids local minima with random decisions. It is also a memoryless algorithm. Consider as in the compass routing the line defined by forwarding node and destination. At each node, two options are considered to route a packet: the neighbor with smallest angle above that line and the neighbor with smallest angle below that line. One of those neighbors is randomly chosen to be the next hop.

Most Forwarding within Radius

Consider that line SD is the x-axis, where S is the forwarding node and D is the destination node. In the Most Forwarding within Radius (Takagi & Kleinrock, 1984) (MFR) node S forwards the packet to the node A that maximizes progress along x-axis. A is therefore the node that minimizes the dot product $\overrightarrow{DS} \cdot \overrightarrow{DA}$, assuming that it is positive (if it is negative $\angle SDA > \pi/2$, which means that S and D must be neighbors). This algorithm is also loop-free and memoryless.

Geographical Distance Routing

GEographical DIstance Routing (Stojmenovic & Lin, 2001) (GEDIR) resembles greedy algorithm, with a subtle difference. Packets are sent to the neighbor A that is closest to destination D, despite the distance of the current node, S, to the destination. This means that a packet can be sent to some node A that is actually more distant from D than the sending node S. The rationale for this is that A may have some neighbor that is closer to D than S is. The only kind of loop that may occur in this algorithm is between two consecutive nodes and, therefore, one can make it loop-free (Stojmenovic & Lin, 2001).

Figures 2.4 and 2.5 try to illustrate the routing algorithms described before. Forwarding node S is aware of all the nodes depicted except of

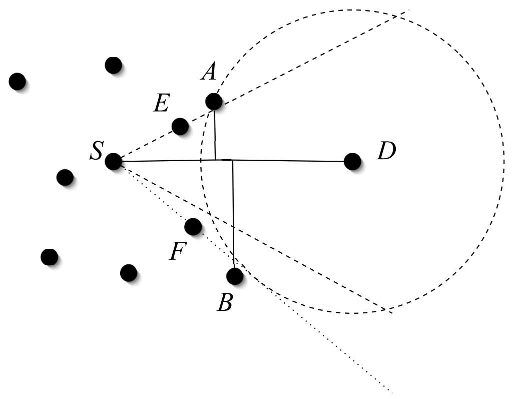


Figure 2.4: Next hops of several routing algorithms

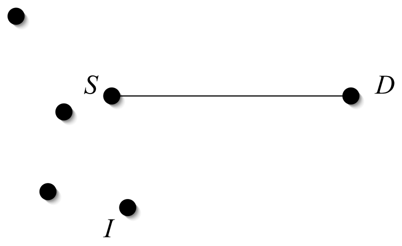


Figure 2.5: Next hop of GEDIR

destination D. Greedy algorithm will choose node A, as this node is closest to destination. Compass will choose E as this has the smallest angle, while randomized compass also allows the selection of F, as F defines the smallest angle in the opposite side. MFR will select B. Finally, GEDIR (Figure 2.5) attempts to deliver the packet even when S is a local minimum. In this case the next hop is node I.

2.4.2 Right-Hand Routing Algorithms

None of the deterministic algorithms presented in Section 2.4.1 can ensure routing convergence. For each one of these algorithms, there is always a graph where they will fail to find a path to destination, even when that path exists. However, it is unacceptable to successively fail to find a path if such path exists. Therefore, they are not adequate to environments where the shape of the graphs is not controlled and is pretty much arbitrary. Fortunately, there are algorithms that can overcome this problem. These algorithms are based on the right-hand rule (Bondy & Murty, 1976). This rule states that all the walls of a maze can be visited if the visitor never lifts his/her right-hand of the wall. If instead of a maze we have faces of a connected planar graph, such rule allows a packet to visit all the edges of a face. This means that a packet routed under the right-hand rule always returns to the source node. Algorithms based on the right-hand rule require $O(1)$ memory, because packets need the sender information, S, to determine which edges intersect line segment SD. Next, we present two algorithms that are based on the right-hand rule. The key point here is that these algorithms must be executed on planar graphs, otherwise they may fail to converge. Ahead in the text, we will focus on the problem of making a graph planar.

FACE-1

A very simple routing algorithm based on the right-hand rule is Compass II (Kranakis *et al.*, 1999), later renamed as FACE-1 by Bose *et al.* (1999). In this algorithm, nodes forward the packet from face to face, always getting closer to destination. The packet goes through faces that intersect line segment SD (known as r in the Algorithm 2.1). Assume that the packet

Algorithm 2.1 Algorithm FACE-1

```
constant $S \leftarrow$ source
constant $D \leftarrow$ destination
constant $r \leftarrow$ line segment $SD$
$P \leftarrow S$
while $P \neq D$ do
  $f \leftarrow$ first face that intersects line segment $PD$ from $P$ to $D$
  for all edges $e$ of $f$ do
    if $e$ intersects $r$ at point $X$ and $X$ is closer to $D$ than $P$ then
      $P \leftarrow X$
    end if
  end for
  traverse $f$ again until reaching edge where $P$ was found
end while
```

is inside face f. We set the variable P to the initial source node S. Then, packet goes from edge to edge (either clockwise or counterclockwise) until it reaches some edge e that intersects r. The packet must keep track of which edge of face f intersects r closest to destination D. Then, when the packet returns to that edge it is certain that there is no other intersecting point P closer to D. At this point the packet may switch from face f to the next face, closer to D. Algorithm 2.1 describes FACE-1.

To see why FACE-1 needs a planar graph, Figure 2.6 shows a case where FACE-1 fails to converge in the presence of intersecting edges. Most algorithms based on the right hand rule will also fail to converge in this case.

FACE-2

A slightly different version of this algorithm, FACE-2, was proposed by Bose *et al.* (1999) (see Algorithm 2.2 executed at graph G). FACE-2 tries to avoid the multiple traversals of the same face that may severely affect performance of FACE-1. FACE-2 switches face whenever the packet reaches a new intersection between r and a different edge, which is closer to desti-

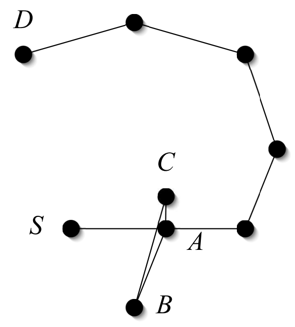


Figure 2.6: Packet from S will never reach D

Algorithm 2.2 Algorithm FACE-2

```
constant $S \leftarrow$ source
constant $D \leftarrow$ destination
$P \leftarrow S$
while $P \neq D$ do
  $f \leftarrow$ face of $G$ with $P$ on its boundary that intersects $PD$
  traverse $f$ until reaching an edge $UV$ that intersects $PD$ at some point $P' \neq P$
  $P \leftarrow P'$
end while
```

nation. Then, the process is repeated for the new face f. Note that f may actually be the same face, if f is not convex. While FACE-1 reaches destination node D in at most $3|E|$ hops, where $|E|$ is the number of edges of graph, FACE-2 has a worse theoretical upper bound, but works better in practice. The experimental results of Bose *et al.* (1999) show that, in spite behaving really badly for some input graphs, FACE-2 generally outperforms FACE-1. Nevertheless, neither one of these algorithms may be used as a standalone routing method, due to their bad performance. As we discuss next, we use these algorithms in complement with better performer, but unreliable algorithms, like greedy or compass, for instance.

2.4.3 Hybrid Position-Based Routing Algorithms

Greedy Perimeter Stateless Routing

One algorithm that explores the duality between efficient and reliable approaches is the Greedy Perimeter Stateless Routing, GPSR (Karp & Kung, 2000). GPSR is based on the original idea of Bose *et al.* (1999). GPSR is a well-known and fairly simple routing protocol for planar graphs. It uses the greedy routing algorithm and the right-hand rule, which is called perimeter routing in this context. More precisely, GPSR is similar to the FACE-2 algorithm when in perimeter mode. The main idea is to use greedy algorithm whenever possible and switch to perimeter routing whenever greedy gets stuck at a local minimum. Then, routing proceeds using perimeter routing and switches back to greedy as soon as it finds some node closer to destination than the previous minimum. The intuition is that greedy algorithm performs better, but it is unreliable, while perimeter algorithm always works if the underlying graph is planar.

AFR and GOAFR$^+$

Kuhn *et al.* (2002) have taken the idea of GPSR a step further and presented an algorithm called "Adaptive Face Routing" (AFR). AFR is guaranteed to achieve a worst case cost of $O(c^2)$, if c is the cost of the best path. Like GPSR, AFR combines greedy routing with face routing to ensure routing convergence without compromising performance. As we have seen in Section 2.3.5, since the worst-case cost for any localized position-based routing algorithm is $\Omega(c^2)$, AFR is asymptotically optimal. Following this work, authors presented another algorithm, GOAFR$^+$ (pronounced as "gopher-plus") that improved performance in random graphs, while maintaining the asymptotically optimal properties of AFR (Kuhn *et al.*, 2003).

2.5 Pre-processing Algorithms for Wireless *Ad Hoc* Networks

Before nodes can run any routing algorithm, they need to apply a pre-processing algorithm to create some kind of underlying graph. Usually, nodes start by exchanging beacon messages to create an initial connectivity graph (e.g., UDG). However, the properties of this initial graph may not be adequate for all routing algorithms. For instance, right-hand and hybrid routing algorithms, like FACE-1 or GPSR, need to use an underlying planar graph. This requires the pre-processing algorithm to remove some of the existing edges in a process known as "topology control". Since the details of the pre-processing algorithms widely differ between wireless and wired networks, we separate the presentation of the two cases. In this section we focus on the wireless case, while we defer the wired case to the next section.

A wireless *ad hoc* network is comprised of nodes that run on batteries and communicate with each other directly via radio using wireless links. A distinctive feature of wireless *ad hoc* networks is the lack of a fixed infrastructure of support. As a consequence, nodes must self-organize in a decentralized way. These characteristics turn these networks eligible for use in a number of circumstances, including the following:

- casual meetings, like conferences or sports events;
- catastrophic situations, where fixed infrastructures are no longer run-

ning;

- for rescue or military teams in inaccessible or hostile regions;
- self-managed networks of sensors, where new sensors can be added or sensors can go down due to battery exhaustion, at any moment. These networks may be used to monitor the environment (e.g., to detect fire or flooding), for health, home or commercial applications.

Wireless *ad hoc* networks can be further characterized by having few resources, both at the node and at the network level. Nodes have limited memory, processing power and possibly worst of all, short batteries. On the other hand, the existing network bandwidth must be divided by all the nodes within reach. Additionally, topology tends to change very fast and radio communication is typically very expensive in terms of energy. Therefore, there is a strong motivation to use simple and economical pre-processing algorithms.

We assume that nodes have already exchanged beacon messages to know of each other and as a result they have already formed an initial graph according to the UDG model. The goal of the following pre-processing algorithms is to create a final planar graph. All these algorithms are restricted to the UDG model. In fact, eliminating intersecting edges in more general models is a quite difficult problem that we do not try to solve in this thesis. Nevertheless, in Chapter 5, where we present "Cell Hash Routing" (CHR), we can easily use a model that is more general than UDG.

2.5.1 Gabriel Graph

Consider nodes X and Y from the initial node set V. If circle whose diameter is AB, i.e., $d(A,B)$ is empty of other nodes from V then edge AB

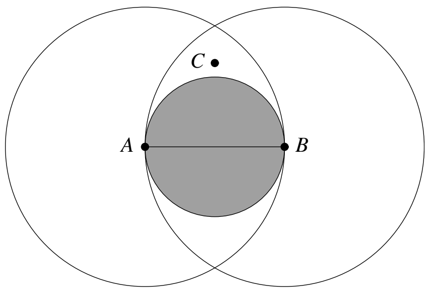


Figure 2.7: In a *GG*, the gray area must be clear

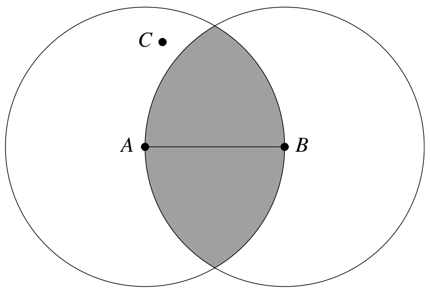


Figure 2.8: In a *RNG*, the gray area must be clear

belongs to the Gabriel graph. This is depicted in Figure 2.7, where the gray color shows the area that must be empty of nodes. The set of all Gabriel edges defines the Gabriel graph (*GG*). The *GG* is a subgraph of the *DT*. This property directly results from the empty circle property of the *DT*. In our context we are more interested in the constrained *GG*, which only has edges with length at most 1. In general, we will always mean *constrained Gabriel graph* even when we do not use the word *constrained*. The constrained *GG* is a subgraph of *UDG*.

2.5.2 Relative Neighborhood Graph

In the relative neighborhood graph (*RNG*), an edge *AB* exists when there is no third node $C \in V$ such that edges *AC* and *BC* are both shorter than *AB*. Again, in our context, we are interested in the constrained *RNG* graph, which only has edges with length at most 1. Both *GG* and *RNG* are planar. Both are connected as long as initial graph is also connected. Figure 2.8 shows the gray area that must be free of nodes. The "exclusion zone" of the *RNG* contains the corresponding zone of the *GG*, which means that the restricted *RNG* is a subgraph of the restricted *GG* and of the *UDG*. The (unconstrained) *RNG* is also a subgraph of the *DT*.

2.5.3 Delaunay Triangulations for Wireless Networks

The problem with both *GG* and *RNG* is that neither one of them is a good spanner of the initial connection graph (Eppstein, 2000), which means that, in particular, they are not good spanners of UDG. One way to create better spanner graphs is to use a triangulation. However, under the UDG model, a complete Delaunay triangulation may not exist, because some edges may be longer than 1. Furthermore, even if all the Delaunay edges were shorter than 1, creating a Delaunay triangulation would make a routing scheme fail the criteria of being localized. This happens because ensuring the empty circumcircle property may require information of nodes that may be close in terms of Euclidean distance, but that may be very far away in terms of number of hops. Even though, this triangulation still owns some attractive properties. For instance, if nodes have similar views of their neighborhood, they can deterministically compute the same triangulation. This may save many steps to reach a form of agreement among the nodes. Additionally, it is possible to create variants of the Delaunay triangulation that are good spanners of UDG. Next, we review some of these graphs.

Localized Delaunay Triangulations

The most obvious variation of the Delaunay triangulation is probably the Unit Delaunay triangulation ($UDel$). $UDel$ results from the intersection of the Delaunay triangulation with the UDG graph. $UDel(V) = Del(V) \cap UDG(V)$, which means that $UDel$ is the Delaunay triangulation with edges that have length at most 1 [2]. $UDel$ is still impossible to build in a localized fashion and therefore, we will use another definition proposed by Li

[2]Other possible name for this would be the “constrained Delaunay triangulation”.

et al. (2002) of "k-localized Delaunay graph over a node set V", $LDel^{(k)}(V)$. $LDel^{(k)}(V)$ is comprised of two types of edges (not longer than 1):

- all edges from the *GG*;
- edges of all triangles ABC for which there are no nodes inside $\bigcirc ABC$ reachable by A, B or C in k or fewer hops.

Li *et al.* (2002) proved that $LDel^{(k)}(V)$ is planar for $k \geq 2$, but edges may intersect for $k = 1$.

$PLDel(V)$ (Li *et al.*, 2002; Lan & Wen-Jing, 2002), which stands for "Planar Localized Delaunay Triangulation", is defined as a planar graph comprised of all triangles of $LDel^{(1)}(V)$, *except* intersecting triangles that do not belong to $LDel^{(2)}(V)$. Li *et al.* (2002) proved that $UDel(V)$ is a $(4\sqrt{3}\pi)/9$-spanner of $UDG(V)$ and that $LDel^{(k)}(V) \supseteq UDel(V)$. Hence $PLDel(V)$ and $LDel^{(k)}(V)$, for all k, are also $(4\sqrt{3}\pi)/9$-spanners of $UDG(V)$.

Restricted Delaunay Graph (*RDG*)

Gao *et al.* (2001) presented an algorithm that creates a graph called the Restricted Delaunay Graph (*RDG*). *RDG* is simply any planar super-graph of $UDel$. Hence, $RDG(V) \supseteq UDel(V)$, which means that *RDG* is also a $(4\sqrt{3}\pi)/9$-spanner of $UDG(V)$. Their *RDG* graph is relatively simple, although possibly expensive in terms of communication. After exchanging information of its own position with its neighbors, each node uses an additional communication step to broadcast its own view of the Delaunay triangulation. This serves to make the triangulation between the nodes converge and to eliminate possible intersections. Let $UDel(A)$ be the Unit Delaunay triangulation computed by node A. Algorithm 2.3 shows the pseudo-code that the nodes must execute after this final communication

Algorithm 2.3 Algorithm that creates *RDG*

```
$E(A) \leftarrow \{AB | AB \in UDel(A)\}$
for all edge $AB \in E(A)$ do
    for all $C \in N(A)$ do
        if $A, B \in N(C)$ and $AB \notin UDel(C)$ then
            Delete edge $AB$ from $E(A)$
        end if
    end for
end for
```

step. Consider that node A executes this algorithm. Basically, A deletes edge AB if there is some node C that simultaneously knows A and B and that does not include AB in its triangulation. This means that AB is not a Delaunay edge.

Algorithms that create $PLDel(V)$

While the *RDG* is a good spanner of *UDG* (it is also a $(4\sqrt{3}\pi)/9$-spanner), its communication cost is $O(n^2)$, because each of the n nodes may see $O(n)$ nodes and $O(n)$ edges in the Delaunay triangulation. To overcome this problem Li *et al.* (2002) proposed an algorithm where nodes only announce *some* of their own edges. Especially in dense networks, this is considerably more economical. This algorithm builds $PLDel(V)$, which is, in fact, just a concrete case of an *RDG*. To simplify the presentation, we will follow the approach of the authors and maintain two separate parts, ran at node A. The first part, Algorithm 2.4, builds $LDel^{(1)}(V)$. We do not include a first step necessary to broadcast the identification and position of a given node. We assume that nodes already know the location of their neighbors. Although $LDel^{(1)}(V)$ is already a $(4\sqrt{3}\pi)/9$-spanner of *UDG*, it is not planar and, therefore, authors need Algorithm 2.5 to planarize $LDel^{(1)}(V)$. This algorithm removes all triangles that are intersected and that do not belong to $LDel^{(2)}(V)$.

Algorithm 2.4 Algorithm that creates $LDel^{(1)}(V)$

```
Node $A$ computes its Delaunay triangulation, $Del(N(A))$ with the neighbors that it is aware of, including itself
for all Edge $AB \in Del(N(A))$ do
  if $AB$ is a Gabriel edge then
    Mark $AB$ as final
    {$AB$ will never be deleted}
  end if
end for
for all Triangles $\triangle ABC \in Del(N(A)) \mid ||AB|| \leq 1 \wedge ||AC|| \leq 1 \wedge ||BC|| \leq 1$ do
  {Ignore any triangle that has one or more long edges}
  if $\angle BAC \geq \pi/3$ then
    Broadcast a message proposal$(A,B,C)$ to form a 1-localized Delaunay triangle $\triangle ABC \in LDel^{(1)}(V)$
  end if
end for
Receive messages from the other nodes
for all Message proposal$(A,B,C)$ received do
  if $\triangle ABC \in Del(N(A))$ then
    {Accept the triangle}
    Broadcast accept$(A,B,C)$
  else
    {This triangle does not exist. Reject it}
    Broadcast reject$(A,B,C)$
  end if
end for
for all Triangles $\triangle ABC \in Del(N(A))$ do
  if $B$ and $C$ sent accept$(A,B,C)$ or proposal$(A,B,C)$ then
    Add edges $AB$ and $AC$ to the triangulation
  end if
end for
```

First thing to notice is that as soon as nodes know the position of their neighbors, marking the Gabriel edges is costless in terms of communication. Hence, communication cost comes from announcing the triangulation. When node A computes $Del(N(A))$ it can have an arbitrarily large number of wrong triangles, i.e., triangles like $\triangle ABC$ that B and C known not to exist. If A were to announce all these triangles, communication cost could grow to $O(n^2)$. To avoid this, node A only announces the triangles for which $\angle BAC \geq \pi/3$. As a result, nodes can only announce up to 6 triangles in a first step, thus limiting the communication cost to $O(n \log n)$.

Algorithm 2.5 Algorithm that planarizes $LDel^{(1)}(V)$

```
Broadcast Gabriel edges incident on $A$ and the triangles $\triangle ABC \in LDel^{(1)}(V)$
Receive the messages from other nodes
{Assume that $A$ gathered information of triangles and Gabriel edges from all its neighbors}
for all Intersecting triangles $\triangle ABC$ and $\triangle XYZ$ known by $A$ do
  if $X, Y$ or $Z \in \bigcirc ABC$ then
    Remove triangle $\triangle ABC$
  end if
end for
Broadcast Gabriel edges and triangles $\triangle ABC$ incident on $A$ that were not removed in the previous step
Listen to the messages from other nodes
for all Edge $AB$ do
  if $AB$ is a Gabriel edge or $\exists \triangle ABC | A, B, C$ announced $\triangle ABC$ in the previous step then
    Keep edge $AB$
  end if
end for
```

Hence, there are some triangles that are not announced by A. However, this is not a problem, because at least in one of the vertexes, the angle must be greater than $\pi/3$. As a result, the other two nodes are forced to agree or disagree on the triangle (all without compromising the theoretical communication cost). In the end of this algorithm, a triangle $\triangle ABC$ can only exist if the three nodes have included it in their localized Delaunay triangulations ($Del(N(A))$ in the case of A). Hence, Algorithm 2.4 creates $LDel^{(1)}(V)$. Now, node A needs to ensure that it gets a planar graph, so it executes Algorithm 2.5. This algorithm uses two steps of communication first to remove possible intersections and then to check the edges that remain in the graph.

Besides the work of Li *et al.* (2002), there is the work of Lan & Wen-Jing (2002) that also builds $PLDel(V)$. However, although not precisely stated, the communication cost of their algorithm should raise to $O(n^2)$ if no optimizations are used. Figure 2.9 summarizes the relations between well-known graphs, including these triangulations. Being on top of an-

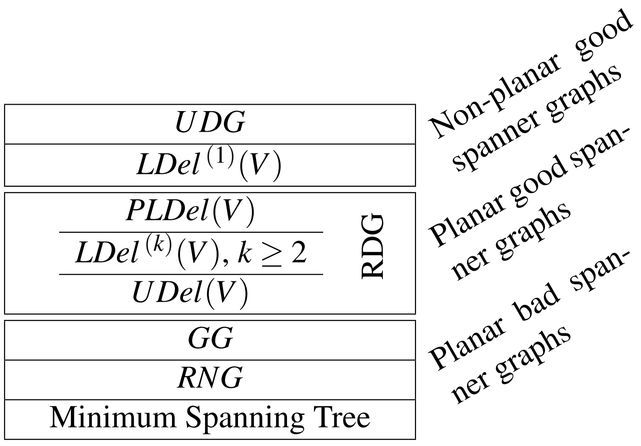

UDG
$LDel^{(1)}(V)$
$PLDel(V)$
$LDel^{(k)}(V), k \geq 2$
$UDel(V)$
RDG
GG
RNG
Minimum Spanning Tree
Non-planar good spanner graphs
Planar good spanner graphs
Planar bad spanner graphs

Figure 2.9: Relation between some well-known graphs

other graph means to contain such graph.

2.6 Pre-processing Algorithms for Wired Networks

When applied to wired networks, position-based routing uses different assumptions. To start with, wired nodes and networks have typically much more resources than their wireless counterparts. This implies a shift in the concerns. Namely, there is no longer a strong reason to use a localized routing scheme. Another important difference is that more often than not, wired networks do not have a communication range that looks like a circle, nor do they have easy access to a broadcast communication channel. As a consequence, the UDG model makes little sense. Therefore, in a wired network, nodes can do more than creating the pre-processing algorithms presented in Section 2.5. In fact, nodes can use more elaborate, non-localized pre-processing algorithms. For example, Liebeherr *et al.* (2001) creates a complete Delaunay triangulation to support multicast at the application layer. Dobkin *et al.* (1990) showed that the Delaunay triangulation is a

$(1+\sqrt{5})\pi/2$-spanner of the complete Euclidean graph. This bound was later improved by Li *et al.* (2002) to $(4\sqrt{3}\pi)/9$, which is a small number ($\approxeq 2.42$).

Moreover, a non-localized pre-processing algorithm offers some important advantages to the routing algorithm. In the case of a complete Delaunay triangulation, it is no longer necessary to use an algorithm like GPSR (Karp & Kung, 2000), because Bose & Morin (1999, 2001) showed that both the Greedy and Compass converge in a Delaunay triangulation (however, they can be defeated by random triangulations). Another advantage is that, in wired networks, it is reasonable to build a c-competitive (non-localized) routing scheme. Bose & Morin (1999) presented an $O(1)$ routing algorithm called "Parallel Voronoi Routing" that is c-competitive in Delaunay triangulations. Later, they extended this result for a broader class of triangulations (Bose & Morin, 2001). Unfortunately, the same authors have also shown that, in practice, performance of Parallel Voronoi Routing is worse than that of greedy or compass routing (Bose & Morin, 1999). Therefore, although these latter algorithms can achieve poor results in some pathological cases, in the normal random case their performance is satisfactory. For this reason, in wired networks, we advocate the use of a simpler algorithm, such as the greedy routing algorithm.

2.7 Comparison of Routing Schemes

2.7.1 Comparison of Pre-processing Algorithms

As a result of the differences between wired and wireless networks, the kind of graph that is more appropriate to each environment varies. For instance, in wireless environments it is crucial to use few messages and

Table 2.1: Comparison of pre-processing algorithms for wireless and wired networks

	RNG	*GG*	Gao *et al.* (2001)	$PLDel(V)$	$LDel^{(k)}(V), k \geq 2$	*DT*
Wireless	S	S	S	S	P	I
Wired	P	P	N	N	N	P

to use localized pre-processing algorithms. Unlike this, in wired environments there is no broadcasting facility and a higher communication overhead is admissible. In Table 2.1, we evaluate the difficulty of creating each of the graphs presented in the previous sections, for wired and wireless environments. We use the following abbreviations: "S" — possible and simple; "P" — possible but may take several rounds of messages or have a high communication cost; "I" — impossible; "N" — makes little sense. Some of the entries of the table are to some extent subjective. Hence, according to this logic, all algorithms take several rounds in wired networks (when compared to wireless networks). We also classify the $LDel^{(k)}(V)$, $k \geq 2$ algorithm for wireless networks with a "P", because good algorithms to do this graph tend to be more complex than other triangulations (Calinescu, 2003; Wang & Li, 2003). Naturally, new, simpler algorithms could make this classification change. Finally, it is to some extent pointless to create most of the triangulations in a wired environment, because it is not obvious whether those would be simpler than creating the complete *DT*.

2.7.2 Comparison of Routing Algorithms

Since the main goal of any routing algorithm is to deliver packets, the most important evaluation criterion is unquestionably the delivery success rate. However, guaranteed delivery cannot be achieved at any price, for instance, by flooding each packet throughout the entire network. Hence, another relevant criterion is the communication effort of the algorithm. In particular, one important distinction is whether the algorithm uses flooding to deliver packets or not. Memory requirement of the algorithm is also important, because some algorithms require information about past packets or require nodes to store large routing tables. It is also important to know if the algorithm is localized, according to the definition of Section 2.3.3.

Table 2.2 resumes a comparison between routing algorithms. We include "Shortest Path Algorithms" (SP algorithms), i.e., algorithms that are not based on position. These algorithms inherit from the techniques of the wired IP networks. The values of the table are valid for the Destination-Sequenced Distance-Vector (DSDV) of Perkins & Bhagwat (1994); *Ad hoc* On-Demand Distance Vector Routing (AODV) of Perkins (1997) and Dynamic Source Routing (DSR) of Johnson & Maltz (1996). We assumed a worst-case scenario where all nodes have routing entries to all other nodes, thus requiring $O(n \log n)$ memory, per node, where n is the number of nodes. In general, this upper bound is unreachable. The memory entry of the table for SP algorithms includes this value, followed by the space required by the algorithm in each packet. Position-based algorithms only include the latter, because space needed to store the graph strongly depends on the pre-processing algorithm used (but this is typically $O(\log n)$). Regarding the space used in each packet, all the algorithms need at least $O(\log n)$ bits to identify the destination. However, to make a clearer distinction between them, in Table 2.2, we refer to the number of other nodes in the packet, besides the destination ($O(1)$ means that the algorithm needs the source, while *memoryless* means that the source is not needed). Of the

Table 2.2: Comparison of routing algorithms

Algorithm	Guarant. del.	Local.	Memory	Flood based	c-comp.
Greedy, Compass	No/DT	Yes/No	Memoryless	No	No
MFR	No	Yes	Memoryless	No	No
Rand. compass	Yes	Yes	Memoryless	No	No
GEDIR	No/DT	Yes/No	$O(1)$	No	No
Voronoi	*DT*	No	$O(1)$	No	No
Parallel Voronoi	*DT*	No	$O(1)$	No	Yes
FACE-1, FACE-2, GPSR, AFR, GOAFR$^+$	Planar G.	Yes	$O(1)$	No	No
SP Algorithms	Yes	No	$O(n \log n)$, $O(1)$	Routing Tables or Route Requests	Yes

routing algorithms included in the table, Greedy, MFR, GEDIR, Randomized Compass, Voronoi as well as shortest path algorithms are loop-free. The face and hybrid algorithms are also loop free in the sense that partial loops occur in a controlled way, in planar graphs.

Of the localized algorithms included, only randomized compass guarantees delivery in arbitrary graphs. Some algorithms guarantee delivery if the graph is a Delaunay triangulation, while face algorithms guarantee delivery if the underlying graph is planar. Remaining localized algorithms do not guarantee delivery. Of the position-based algorithms, only parallel Voronoi is c-competitive (see section 2.3.5). We assume that SP algorithms can find the optimal path, under favorable circumstances. Algorithms may use hop count, energy or distance as their metric. Finally, of the algorithms that we include in the comparison, only shortest path algorithms use flooding. Depending on the particular case, these algorithms use flooding to propagate route requests or to propagate routing tables.

2.8 Cluster-Based Algorithms

Clustering a network consists of dividing that network into groups of nodes. Usually, each cluster will have a "cluster-head" that will act as the representative of that group of nodes. In a sense, a cluster (often by means of its cluster-head) will act as a kind of super-node that represents all the nodes of the group. This allows to create a network that is much sparser. As a consequence, the management of position and non-position-based routing algorithms becomes much simpler and most nodes send fewer control packets, thus reducing collisions and battery consumption. The drawback of clustering is that, often, some unlucky nodes will have more service than others. Additionally, a sparser network with fewer nodes, may reduce the routing options for most algorithms. In the case of position-based routing algorithms, Greedy, MFR, GEDIR, Compass, Random compass, among many others, should have worse behavior on sparser networks, because they have fewer options there. On the contrary, face and hybrid routing algorithms benefit from clustering. Since they operate in planar graphs, the sparser the graph is (only in terms of nodes, not in terms of edges), the longer are the edges. Longer edges will result in fewer hops to reach destination. The algorithm GOAFR$^+$ (Kuhn *et al.*, 2003) explores this idea. This result is also clearly shown in Chapter 5.

Division into cells

There is a very large body of work that focus on clustering (e.g. Ni *et al.*, 1999; Das & Bharghavan, 1997; C.R.Lin & Gerla, 1997; Gao *et al.*, 2001; Wang & Li, 2002; Chen & Stojmenovic, 1999; ITSIRadioEquipment, 1996; Calinescu *et al.*, 2001; Qayyum *et al.*, 2000; Wu & Li, 1999; Stojmenovic

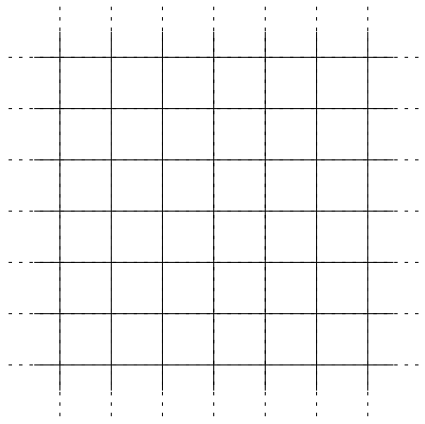


Figure 2.10: Division of the space into cells of fixed size

et al., 2002; Ni *et al.*, 2001). However, in this thesis we only make use of a very simple clustering technique that is based on positional information. Xu *et al.* (2001) use this technique in the "Geographical Adaptive Fidelity" (GAF) algorithm to conserve energy. Similar techniques that could be used to create DHTs exist in the work of Gupta *et al.* (2001) and of Li *et al.* (2004b). The goal of GAF is to put as many nodes as possible to sleep and maintain only one node active in each cluster. After a predetermined period of time, sleeping nodes must wake up to substitute active nodes. GAF divides the space into equally-sized cells that have the shape of squares (assuming that the entire space is a square), as the grid depicted in Figure 2.10. Division of the space into cells allows a simple definition of the network of clusters: all nodes inside the same cell belong to the same cluster. The size of the cells is limited by the communication range of the nodes, because nodes in a cell must always listen to any other node either in its own cell or in any adjacent cell. This restriction ensures that in most circumstances, the clustered network stays connected, as long as the initial network is also connected. If we assume that nodes have a communication range of R, the resulting square side is at most $R/\sqrt{5}$ or $R/\sqrt{8}$, if 4 or 8 cells surrounding the node are considered to be adjacent, respectively (this can be seen in Figure 5.1 for 8 adjacent cells).

This division into cells is, in fact, very convenient to create a position-based DHT. Not only it allows to improve solutions for routing, but it also simplifies the division of keys among the nodes. Therefore, this cell structure will be the starting point for the DHT that we present in Chapter 5, called "Cell Hash Routing".

2.9 Summary

In this chapter, we surveyed position-based routing schemes. A routing scheme is comprised of a pre-processing algorithm plus a routing algorithm. In wireless networks, a popular routing scheme consists of combining the greedy perimeter stateless routing algorithm (GPSR) with a planar graph, like the Gabriel graph. Triangulations can replace the Gabriel graph to achieve better performance, but they are much more difficult to create. In wired networks, position-based routing is a simpler problem, because nodes can usually afford to create a complete Delaunay triangulation. This enables the use of routing algorithms even simpler than GPSR, such as greedy. We address some of the greatest challenges of position-based routing in the wireless distributed hash tables that we create in Chapters 4 and 5: efficient creation of a triangulation and clustering of nodes to improve performance in densely populated networks.

3 Survey on Distributed Hash Tables

A dictionary stores values which can be accessed by associated keys. A hash table is a dictionary in which keys are mapped to array positions by a hash function. Informally, a hash table stores (key, value) pairs[1]. A user of the dictionary must use a key to store or to retrieve the corresponding value. A centralized solution, where a single node holds the entire dictionary, is prone to a number of problems, like lack of tolerance to faults, high congestion and low scalability. The natural answer to these problems lies in the decentralization of data provided by a distributed hash table (DHT). In a DHT, nodes must cooperate to maintain the coherence of the data. Nodes use a consistent hash function (Karger *et al.*, 1997) to determine the peer that holds a given value (often, some sort of file, a music, or a pointer to the location of those items). Usually, nodes self-organize to form a communication graph which has optimal or near-optimal path length/node degree trade-off (in wired networks, both are typically logarithmic with respect to the number of nodes). In this way, although nodes have a small number of neighbors (logarithmic or best) the DHT needs only a small number of hops (logarithmic or best) to satisfy requests from

[1]See for example the web page of National Institute of Standards and Technology — Dictionary of Algorithms and Data Structures: http://www.nist.gov/dads.

clients. When compared to a centralized solution this is a fair price to pay, because distribution offers some considerable advantages, like improved fault-tolerance. In fact, a carefully designed DHT may resist to single or even multiple node failures, loosing only the data that was in the departing nodes. Another advantage of a distributed solution is the reduced congestion. While a centralized server needs to reply to all requests, most DHTs need to reply only to a small fraction of the requests, like $O(\log n/n)$, where n is the number of nodes. Furthermore, DHTs scale better than a centralized solution, both in terms of communication and storage requirements.

3.1 DHTs in Overlay Networks

An overlay network is a network operated on top of another underlying network, but organized under an independent logic. For this reason overlay networks are also deemed as "logical" or "virtual" networks. The growing interest of users in peer-to-peer applications, like Napster[2] and Gnutella[3] ignited the creation of many peer-to-peer overlay networks that implement DHTs, like Pastry (Rowstron & Druschel, 2001), Tapestry (Zhao *et al.*, 2001), Chord (Stoica *et al.*, 2001) and CAN (Ratnasamy *et al.*, 2001). Although different in their details, most of these DHTs share a number of common features. According to Gummadi *et al.* (2003a), in a DHT, one can often distinguish between the "routing scheme"[4] and the "routing geometry" of the overlay network. The routing scheme is the set of mechanisms that determines the neighbors and the next hops (this basi-

[2]See http://www.napster.com.
[3]See http://www.gnutella.com.
[4]In the work of Gummadi *et al.* (2003a), this is called the "routing algorithm".

cally corresponds to the formal definition of *routing scheme*). Often, the routing scheme compels the nodes to create an underlying graph with a predetermined organization, which is called the "routing geometry". The notion of *routing geometry* is quite useful, because it allows us to pinpoint some of the most relevant differences among existing DHTs. Depending on the particular geometry, nodes may have some flexibility to choose their neighbors and to choose paths for the messages, given their actual neighbors. As we shall see, some routing geometries give their nodes degrees of freedom in both aspects (neighbors and paths), others only in one, while the remaining give none. Flexibility to choose neighbors allows the DHT to select closer peers, thus improving routing latency, while flexibility to choose paths also impacts latency (less) and increases tolerance to faults caused by nodes that have departed. To shorten presentation of the DHTs, we will use a "generic DHT" with no specific routing geometry. We use this generic model to present the components of the routing scheme that are common to most DHTs. Then, we briefly overview and compare some of the most well-known DHTs that exist.

3.1.1 The Routing Scheme of a Generic DHT

A DHT stores (key, value) pairs, e.g., ("Bob", 32), where "Bob" is the key, while 32 is the value we want to store, e.g., Bob's age. The DHT must have a globally known hash function capable of converting the key to a pseudo-random value, e.g., an integer or a position in a virtual space. One of the crucial aspects is that the hash function should balance the distribution of keys throughout the space. We first consider that the DHT creates an underlying graph $\mathcal{H}$, which is a function of the set of existing nodes. Nodes of the DHT receive a (pseudo) random identification that evenly spreads the nodes in the space of identifications. The space of identification of the nodes must coincide with the output space of the hash function (to ensure this, some DHTs, e.g., Bamboo (Rhea *et al.*, 2003), hash the identification of the node plus port of the application). If nodes and keys are evenly distributed in space, each node will store a fair share of the keys.

The routing scheme must create graph $\mathcal{H}$ in a way that a path between two arbitrary nodes always exists (at least in steady state conditions). To retrieve (or to store) the value corresponding to a key, the DHT must find the node that stores that key. Take again the example above and assume that the key "Bob" hashed to 81. In general, node 81 will not exist (because nodes are sparse in the identification space) and some other node will become responsible for that key. Assume that it is node 90. Any node, say node 13 looking for the age of Bob will use the key "Bob", which hashes to 81. In general, nodes only have a partial view of the network and node 13 will not know whether node 81 exists or not. Nevertheless, the DHT will try to route the message to node 81 and will always end up in node 90. One of the problems in most DHTs is that their overlay network bears only a limited (if any) relation with the underlying network. Consequently, each hop in the DHT may represent a very long distance in the Internet and successive hops may make the message travel back and forth several times.

Entrance and departure of nodes from the DHT is often very similar from one DHT to the next. To illustrate the process, we will take as example Pastry (Rowstron & Druschel, 2001):

- to enter the network, a node with identification N must ask some node P_0 already in the network to send a special JOIN message to node N (which hopefully does not exist in the network). This JOIN message will be routed to the node responsible for the identification

N in the network, say P_f. At this moment, P_f will know that there is a new neighbor coming in and it divides its own space and its keys with the newcomer. The remaining actions to take strongly depend on the concrete DHT;

- ideally, departure of nodes involves redistribution of the keys of the leaving node. However, in practice, nodes may leave abruptly, rendering this option impossible.

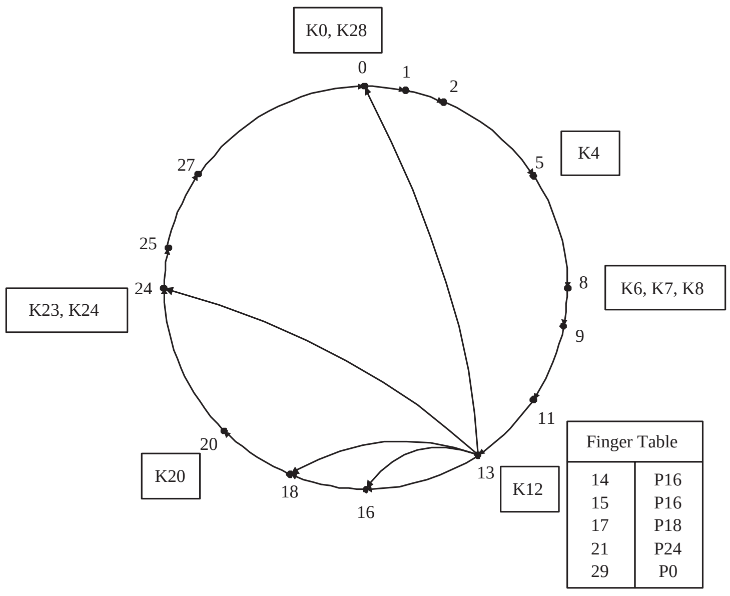
Next, we review some of the most important DHTs.

3.1.2 Chord

Overview

In Chord (Stoica *et al.*, 2001), nodes organize into a logical ring ordered by their growing identification. To close the ring, the smallest node follows the largest one. To maintain the ring, each node keeps a pointer to the node that follows it. The ring allows to define the notion of *successor node of a key*. For some key k, successor(k), is the node of the ring with smallest identification, not smaller than k. To improve routing performance, Chord nodes use "finger tables" with m entries, where 2^m is the number of possible identifications. This scheme requires $O(\log n)$ memory at each node, but ensures delivery of messages in $O(\log n)$ hops with high probability [5]. The i-th entry of the finger table at node P keeps a pointer to the first node S that succeeds P by at least 2^{i-1}, i.e., $S = \text{successor}(P + 2^{i-1})$ (*modulo* 2^m). These pointers correspond to long range contacts (LRCs) carefully selected along the ring. If Chord reaches a steady state, where nodes have updated

[5] With high probability, this means that an event will occur with a probability of at least $1 - O(1/n)$.



Finger Table	
14	P16
15	P16
17	P18
21	P24
29	P0

Figure 3.1: Chord ring

finger tables, each hop successively eliminates half of the possible remaining identifications.

Figure 3.1 depicts an example of a Chord ring with $m = 5$. Nodes hold the keys represented as Kx. We also show the finger table of node 13 (the rows of the first column result from the computation $13 + 2^{i-1}$ *modulo* 32, while the second column is the successor of such node). As an example consider that destination node is $D = 27$ and forwarding node is $P = 13$. In this case, the first hop of the message will go through node successor$(21) = 24$.

Stoica *et al.* (2001) claim that simplicity, provable correctness and provable performance are the main features of Chord. This is further explored by Lynch *et al.* (2002).

Routing Geometry

The routing geometry of Chord is a ring. This ring is augmented with LRCs that nodes store in their finger tables. When populating their routing table, Chord nodes have many options. Except for their immediate neighbors, Chord nodes can select their LRCs in a rather flexible way. The more distant the LRC is the more options there are. Routing also has a large flexibility, because there are many possible routes with $O(\log n)$ path lengths. Ideally, there are $O((\log n)!)$ different ways of arranging the paths with length $O(\log n)$. To see this, consider an average path with a sequence of hops like $n/4, n/8, n/16, \ldots, 4, 2, 1$. We can rearrange this sequence in an arbitrary way and still get the same length. In fact, for k terms, we can have $k!$ different orderings.

3.1.3 Content-Addressable Network (CAN)

Overview

Content-Addressable Network (CAN) divides a virtual (imaginary) d-dimensional torus into d-dimensional zones. There is a one-to-one correspondence between CAN nodes and the d-dimensional zones. The hash function deterministically maps the keys to coordinates of the virtual torus and each process manages all the keys that hash inside its own zone. A visual representation of CAN for anything above two dimensions is not very intuitive and therefore, Figure 3.2 represents a CAN square, for only two dimensions. In fact, this is not a square, but a torus, because coordinates wrap and the virtual space has no borders. Also, note that physical coordinates bear no relation to virtual space.

To route messages, CAN uses a greedy algorithm, where each node

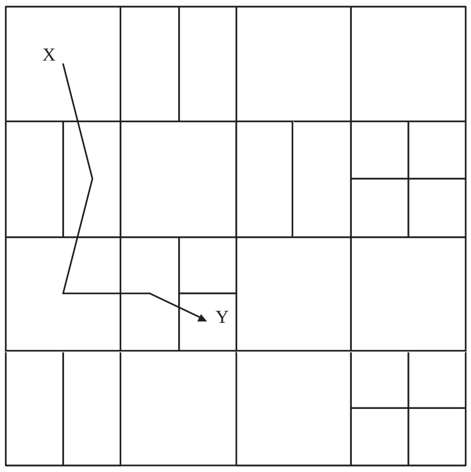


Figure 3.2: Content-Addressable Network (CAN)

sends the message to the neighbor that is closest to destination. In CAN, nodes are only aware of neighbors with which they share a common border of the space. Figure 3.2 depicts an example that illustrates the routing process between X and Y.

Ratnasamy *et al.* (2001) claim that average path length in a d-dimensional CAN with n nodes is $(d/4)n^{1/d}$, i.e., $O(dn^{1/d})$, and that individual nodes maintain only $2d$ neighbors. However, we notice that it suffices to look at Figure 3.2 to see that the number of neighbors may be higher than $2d$. Additional methods are required to avoid very unfavorable space partitions where some node might need to store information of an arbitrary number of neighbors. Authors also state that CAN can achieve $O(\log n)$ hops by setting $d = (\log_2 n)/2$. However, for typical configurations of CAN, where the number of nodes in the system is not known beforehand, the path length/node degree trade-off of CAN is very unfavorable, as paths tend to be very long. CAN can significantly benefit from some enhancements to its basic design (Ratnasamy *et al.*, 2001).

Routing Geometry

The routing geometry of CAN becomes a hyper-cube instead of a torus if we consider that $d = \log n$ and that all possible identifications are taken by existing nodes. In this case, each node will have precisely $\log n$ neighbors each one differing in a single bit. Therefore, routing will take $\log n$ hops, as each one of the hops will correct one of the bits. However, a practical implementation of CAN can hardly correspond to this idealized hyper-cube. Not only the number of nodes of the system is unknown when it boots, but it is likely to vary. Consequently, either the dimensionality is too large or too small. If it is too large, the number of neighbors per node will be above $O(\log n)$. If it is too small, most of the nodes will not lie in the vertices of the hyper-cube and crossing each dimension needs more than a single hop, more than a logarithmic number of hops and in fact $O(n^{1/d})$.

From the two flexibility criteria, CAN only owns one, which is the routing flexibility. Since there is only one way of arranging the hyper-cube, there is no flexibility at all to select the neighbors. On the other hand, nodes can route messages using several of their neighbors. If the forwarding node and the destination have b different bits, the forwarding node can select any of the b neighbors that are closer to destination. This means that there are $O((\log n)!)$ possible paths from source to destination.

3.1.4 Expressways Content-Addressable Network (eCAN)

Overview

The CAN DHT is not very efficient in practice, because path lengths tend to be very long. To overcome this problem Xu & Zhang (2002) proposed an extension to CAN, called "expressways CAN" (eCAN). Refer to Figure 3.3.

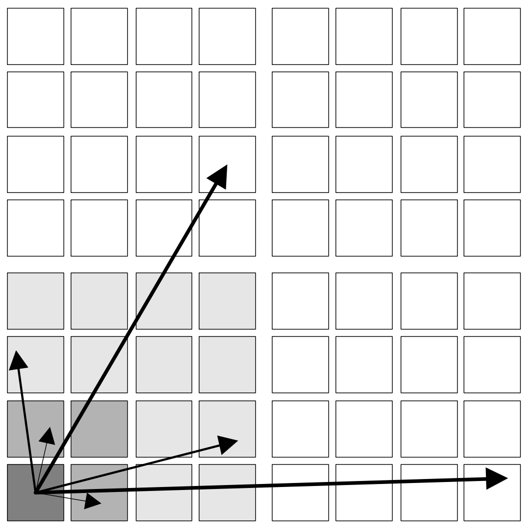


Figure 3.3: eCAN-like LRCs

To simplify presentation we will consider a two-dimensional space, but this mechanism works for an arbitrary number of dimensions. The idea in eCAN is to make a first level division of the entire space into four big squares[6]. Each node keeps LRCs to the two neighboring squares. Then, the four big squares are further divided into other four smaller squares. This time, nodes inside squares have a total number of four LRC (above, below, right and left). This process is repeated for as many levels as wanted. Figure 3.3 illustrates the eCAN-like LRC scheme, for a node in a corner. Wrapping pointers are not shown.

Routing Geometry

If the routing geometry of CAN were truly a hyper-cube with $O(\log n)$ dimensions, eCAN would be pointless. However, in practice, CAN works like a torus and eCAN reduces path lengths in that torus. The LRC mech-

[6]Division in 3×3, 4×4 or any other number of squares is also possible.

anism of eCAN gives a lot of flexibility to nodes in what concerns selection of neighbors. Each LRC has to fall within a hyper-cube with possibly many nodes. However, eCAN is not flexible in route selection, because, in general, there is little gain in progressing until the message crosses the border of its current largest hyper-cube. This means that a node has only one good option to reach destination. Although eCAN has some similarities with the finger table of Chord, the division of the space in hyper-cubes makes it less worthwhile to do small steps forward.

3.1.5 Pastry

Overview

Pastry and Tapestry are two descendants from the work of Plaxton *et al.* (1997). Pastry routes messages as follows. Consider that some node $S = 14543_{10}$ sends a message to node $D = 84944_{10}$. To succeed, S must know some node that starts with an 8. Consider it to be $R_1 = 83135$. Now, R_1 must know about some node that starts with an 8 and has 4 in the second position, say $R_2 = 84899$. This reasoning goes on for nodes $R_3 = 84996$, $R_4 = 84945$ and finally D. Implementations of Pastry use a numeration base of 2^b for some b. b is a parameter of the system, typically set to 4, which means that the numeration base is 16. Rowstron & Druschel (2001) show that under accurate routing tables and in the absence of recent node failures, $O(\log n)$ hops suffice for a lookup operation, w.h.p., while the number of entries in the routing information of each node is $O(\log n)$. n is the number of nodes in the system.

Each node divides its routing information in three parts. The first part is the "routing table", which includes information of peers needed to route

NodeId 23002			
Leaf Set			
Smaller		Greater	
23001	22333	23022	23033
22321	22312	23100	23101
Routing Table			
-0-1023	-1-2131	2	-3-0231
2-0-021		2-2-032	3
0		23-2-33	
0		230-2-2	
		2	
Neighborhood Set			
02132	32100	00213	10023
31102	22311	02310	01213

Figure 3.4: State of node 23002

messages according to the description made before. The "leaf set", L of node P includes a set of nodes with identifications close to P. Nodes use this set to divide keys and to route to nodes with close identifications. Nodes also store a list of nodes that are physically close called "neighborhood set". The purpose of this list is to improve locality properties of routing. By forwarding messages to nodes that are topologically closer, routing becomes more efficient. Figure 3.4 (see Rowstron & Druschel, 2001) shows the information stored in some example Pastry node, for $b = 2$ and $|L| = 8$ in a system that uses 5 digits for the node identification. Node identifications are split in three parts: equal prefix, current digit and different suffix. First row keeps addresses of nodes that have no common prefix with current node. Second row keeps addresses of nodes that share the first digit with the current node and so on. At each row, the cell whose digit matches the node's digit has a gray background. The routing table of each node has an average of $\lceil \log_{2^b} n \rceil$ rows (see Rowstron & Druschel, 2001).

Rowstron & Druschel (2001) claim that Pastry has good locality properties, in the sense that more often than not, nodes will select to neighbors nodes that are close to them. This property is ensured by the way that Pastry nodes build the network.

Routing Geometry

The routing geometry of Pastry is hybrid (Gummadi *et al.*, 2003a). The routing tables of the nodes are organized as a tree, while the leaf set of the nodes forms a ring (with some redundancy). The ring part of the graph is easy to understand and, therefore, we shall focus on the tree. Each node is the root of its own tree. Any node that shares all but the last digit of that identification is a candidate to be a child node. Exactly which nodes exist on that level of the tree depends on things like the nodes that the "root" node used to join the network. The grandchildren nodes share all but the two last digits of the root's identification and so on, until only the first digit coincides in the lowest level of the tree. Figure 3.5 depicts such a tree for node 23002. Routing for node 23002 is straightforward. The path length between two nodes is determined by the subtree that connects them. Transitive network states can cause the tree to be incomplete and force the ring to come into play.

Selection of neighbors in Pastry is quite flexible, especially for entries of the routing table that correspond to nodes which only share a few bits in their identification. For instance, in Figure 3.5, node 21330 could choose many different nodes starting by 23*xxx*, while node 23021 only has a few choices for nodes starting by 2300*x* (possibly only one). On the other hand, there is no routing flexibility in a tree, because there is only one neighbor that yields good routing results.

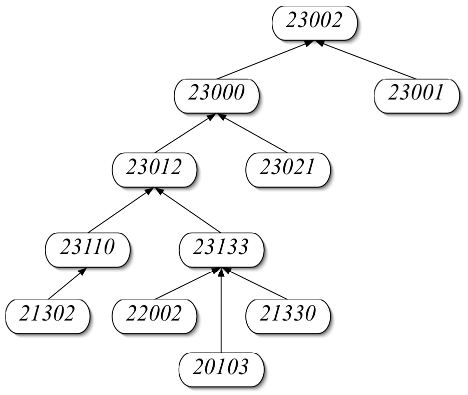


Figure 3.5: Pastry tree rooted at node 23002

3.1.6 Tapestry

Routing Geometry

Pastry and Tapestry's main mechanisms are very similar and therefore, to conserve space, we omit the details of Tapestry. The main difference between the routing geometry of these DHTs is that the routing geometry of Tapestry is a pure tree, because Tapestry does not have any ring. Figure 3.5 also represents this structure, despite the fact that Tapestry resolves digits in the opposite order. Similarly to Pastry, Tapestry is very flexible in the construction of the routing table, while the same is not true for the route selection.

3.1.7 Viceroy

Overview

Viceroy (Malkhi *et al.*, 2002) implements a DHT with constant out-degree and logarithmic diameter. Additionally, insertion or removal of a node requires a number of link changes that is constant in expectation and logarithmic with high probability. The Viceroy network is based on the butter-

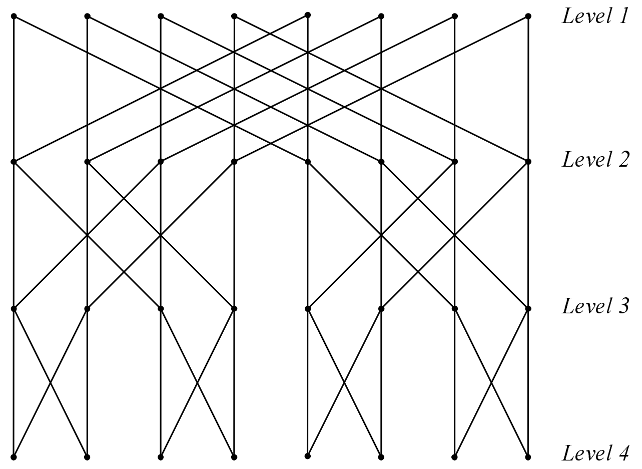


Figure 3.6: The butterfly network

fly network. Figure 3.6 illustrates a butterfly network comprised of thirty two nodes distributed by four different levels. Note that, despite having only a constant number of connections, nodes can reach each other in a logarithmic number of steps.

The principle of the Viceroy network is to enhance the Chord basic ring with carefully selected LRCs inspired in the butterfly configuration. Viceroy nodes try to divide themselves into $\log n$ different levels, being n the number of nodes. A Viceroy network is comprised of three different types of structures: *i*) a ring like Chord; *ii*) the butterfly, that connects nodes from the general ring using an emulation of the butterfly network; and *iii*) level rings that connect nodes from the same butterfly level in a ring structure. To organize the Viceroy network, nodes randomly select identifications in the range $[0,1)$, to determine their positions along the 2π radians of the general Chord ring. A node has the following constant number of connections: two connections with predecessor and successor nodes in the general ring; right and left connections to the next level of the butterfly, a connection to the upper level of the butterfly; and two connections to the predecessor and successor nodes in the level ring. The left and right connections of the butterfly to the next level intend to divide ring space, according to the following rule: for the left connection of node N of level l is selected the neighbor in level $l+1$ that is closest to N in the clockwise direction (i.e., the "clockwise-closest" neighbor) around the general ring; for the right connection is selected the neighbor of level $l+1$ clockwise-closest to $N+1/2^l$. Connections from level l to level $l+1$ are expected to be about $1/2^l$ far apart in the ring. The reader may confront this with the perfect butterfly graph depicted in Figure 3.6, considering that the distance from the leftmost to the rightmost lines is 1. For the up-link, the clockwise-closest neighbor of level $l-1$ is chosen. All the butterfly connections are used for routing purposes.

The level of a node in the butterfly is randomly selected from the set $\{1,\ldots,\lfloor\log n\rfloor\}$, where n is the number of nodes. Being impossible to determine the exact number of nodes, node N estimates this to be $n_0 = 1/d(N,$ successor$(N))$, where $d(N,$ successor$(N))$ is the distance from N to N's successor in the general ring. For instance, if $N = 0.44$ and successor$(N) = 0.64$, N will assume that network has 5 nodes and will select its level between 1 and 2. If successor of N changes, N must re-select its level.

Routing is performed in three sequential steps, called "proceed to root", "traverse tree" and "traverse ring". In the first step, nodes send the message to the first level of the butterfly through their up-links. In the second step, nodes traverse the tree from the first level to the destination. At node N, level l, distance to destination D, $d(N,D)$, is at most $1/2^{l-1}$. Hence, if $d(N,D) < 1/2^l$, left link is chosen, otherwise, right link is chosen. This process will either reach some node without further down links or a node beyond destination D. Assume that $X \neq D$ is the node reached in the end of the second step. The last step will make use of the level ring to achieve destination in $O(\log n)$ steps w.h.p. [7]. In the last step of the routing algorithm, if $X < D$, nodes select either the successor in the level ring, Y, if $Y \leq D$, or the successor in the general ring if $Y > D$. If $X > D$, the reasoning is similar. For further details on Viceroy see the work of Malkhi *et al.* (2002).

Routing Geometry

The routing geometry of Viceroy is a variation of the butterfly graph represented in Figure 3.6. The Viceroy DHT offers no flexibility in the selection of neighbors or in the selection of paths.

3.1.8 D2B

Overview

One of the most interesting features of D2B (Fraigniaud & Gauron, 2003a,b) is that nodes have constant node degree and, still, path lengths are logarithmic. D2B network is based on the *de Bruijn graph*. A de Bruijn graph, $B(2,k)$, for $k \geq 1$, has 2^k nodes with k-bit identifications. In-degree and out-degree of nodes is 2 and diameter of the graph is k. Node with identification $x_1x_2\ldots x_k$ has an arc directed to nodes $x_2\ldots x_k\alpha$, for $\alpha \in \{0,1\}$ and has incoming arcs from $\beta x_2\ldots x_{k-1}$, for $\beta \in \{0,1\}$. Figure 3.7 depicts an example of $B(2,3)$. Routing from $x_1x_2\ldots x_k$ to $y_1y_2\ldots y_k$ is done by selecting intermediate nodes $x_2\ldots x_ky_1$, $x_3\ldots x_ky_1y_2$, etc., until destination is reached. For instance, routing from 101 to 000 is done through 010 and 100.

[7]Malkhi *et al.* (2002) present two versions of the routing algorithm. The simplest version uses the general Chord ring instead of the level rings, but may require $O(\log^2 n)$ steps (more precisely, the expected number of steps is $O(\log n)$ and it is $O(\log^2 n)$ w.h.p.).

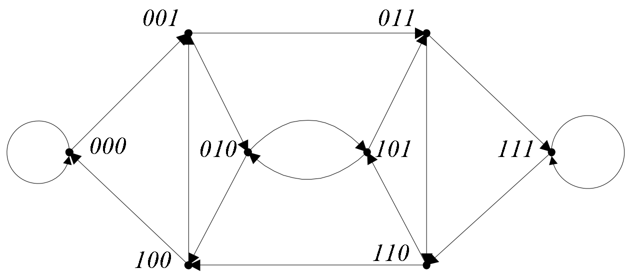


Figure 3.7: Example of a de Bruijn graph, $B(2,3)$

D2B must incorporate some additional features to *i*) support the DHT functionality, *ii*) support sparsity of nodes and *iii*) cope with the dynamic behavior of a network made of nodes that vary in number. The key space of D2B is $\mathcal{K} = \{0,\ldots,2^m-1\}$, for a binary string length of m. The key $\kappa_1\kappa_2\ldots\kappa_m$ is managed by existing node $x_1x_2\ldots x_k$ if and only if $x_1x_2\ldots x_k$ is a prefix of $\kappa_1\kappa_2\ldots\kappa_m$. This raises the notion of "universal prefix set", S, which is a set where for any possible integer with infinite length w there is one and only one prefix of w in S. For example, $\{0,100,1010,1011,11\}$ is a universal prefix set. The empty set is also a universal prefix set. D2B must ensure that at any given time, for any identification, there is one and only one node with a corresponding prefix in the network. In other words, identifications of nodes of D2B must form a universal prefix set.

The sparsity of the identification space implies that, unlike the static de Bruijn graph, node $x_1x_2\ldots x_k$ may have either a single child $x_2\ldots x_j$, $j \leq k$ or several children with identities $x_2\ldots x_ky_1\ldots y_l$, $1 \leq l \leq m-k+1$. If several children exist, the set of sequences $y_1\ldots y_l$ forms a universal prefix set. For instance, node 110 may have more than one children with the following identifications, 100, 10100, 101010, 101011, 1011, or may, otherwise, have a single child with identification of 1 or 10. Given the childhood relation, the parenthood relation is symmetrical. If node A has a single child B with

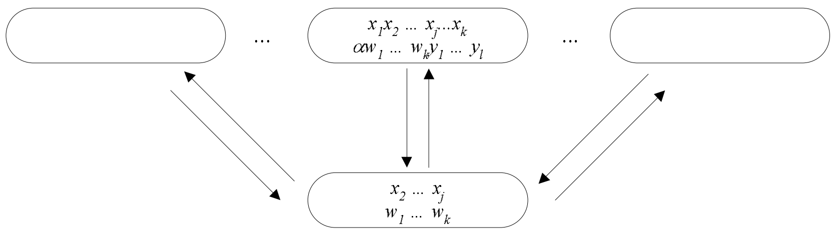

$x_1x_2 \dots x_j \dots x_k$
$\alpha w_1 \dots w_k y_1 \dots y_l$
$x_2 \dots x_j$
$w_1 \dots w_k$
...
...

Figure 3.8: Multiple parent may correspond to a single child

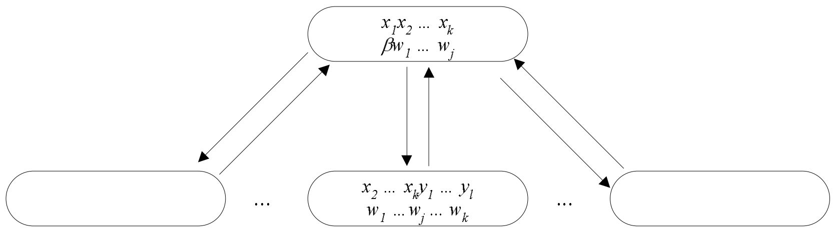


Figure 3.9: One parent may have several children

identification $w_1w_2 \dots w_k$, then, B may have several parents with identifications $\alpha w_1 \dots w_k y_1 \dots y_l$, where $\alpha \in \{0,1\}$ and $y_1 \dots y_l$ forms a universal prefix set. On the other hand, if node A has several children including B, with identification $w_1w_2 \dots w_k$, then B's single parent is A and the identification of A is $\beta w_1 \dots w_j$, where $\beta \in \{0,1\}$. Figures 3.8 and 3.9 illustrate childhood and parenthood relations. To keep coherency with the text we used two different identifications for nodes in the figures.

Routing Geometry

The routing geometry of D2B is a variation of the de Bruijn graph represented in Figure 3.7. D2B is an inflexible architecture, both from the point of view of the neighbor selection and from the point of view of the route selection. This is quite evident for neighbor selection, due to the highly structured de Bruijn graph. Additionally, there is also only one optimal option to route the message and therefore this does also not meet the routing flexibility criterion.

3.1.9 Koorde

Overview

Koorde (Kaashoek & Karger, 2003) is a DHT that augments the basic Chord ring with a de Bruijn graph. Koorde network achieves the following optimal results: *i*) for $O(1)$ node degree, Koorde diameter is $O(\log n)$ w.h.p.; *ii*) for $O(\log n)$ node degree (which enhances fault tolerance), Koorde diameter is $O\left((\log n)/\log\log n\right)$. A Koorde node requires information about 2 nodes: its successor in the ring and the predecessor of $2m\ modulo\ 2$, known as D (unlike a pure de Bruijn graph, there is no pointer to $(2m+1)\ modulo\ 2$, as this would probably be redundant).

The Koorde DHT uses the notions of "virtual node" I and "real node" N. I corresponds to the current routing node in the de Bruijn graph. Since it may happen for identification I not to exist in the Chord ring, node N is a real node that represents I in the ring, such that $I \in [N, \text{successor}(N))$. To route toward key k, node X invokes the function $lookup()$ in Algorithm 3.1 with arguments $X.lookup(k,k,X)$. We used the original notation of Kaashoek & Karger (2003): $kshift$ is used to extract successive bits, starting from the most significant, from key k; function $topBit(x)$ returns the most significant bit of x; finally, $x \circ b$, left-shifts x and introduces the bit b at the right of x. The algorithm first checks if key belongs to the successor. If it does not, but node N represents I, request is forwarded to node D and operator $\circ$ is used on I. Otherwise, request for key is forwarded to the successor in order to search for node I. In Figure 3.10, we depict the Koorde network corresponding to Figure 3.1. The pointer D at each node is shown

Algorithm 3.1 Lookup algorithm at node N

```
Function lookup(k,kshift,I)
  if $k \in (N, \text{successor}(N))$ then
    return successor(N)
  else
    if $I \in [N, \text{successor}(N))$ then
      return call function $D.lookup(k, kshift << 1, I \circ topBit(kshift))$
    else
      return call function $\text{successor}(N).lookup(k, kshift, I)$
    end if
  end if
```

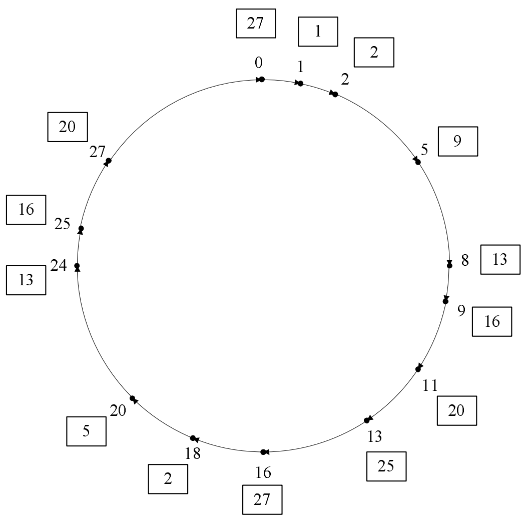


Figure 3.10: A Koorde network

inside a box. Figure 3.11 exemplifies the $lookup()$ function of a request for key $k = 21 = 10101_2$, starting at node $N = 18 = 10010_2$.

The number of hops can be reduced if instead of using N as initial I, the most significant bits of the key k are inserted from the beginning in I, such that I does not overflow the range of N that stops at (not including) successor(N). For instance, in the previous example, we could immediately do $I = 18 = 10010_2$ and $kshift = 101$. Although there is no improvement in this particular case, Kaashoek & Karger (2003) prove that this enables Koorde to achieve $O(\log n)$ path lengths with high probability.

$kshift$	10101	0101	0101	101	01	01	01
I_2	10011_2	00101_2	00101_2	01010_2	10101_2	10101_2	10101_2
I_{10}	19	5	5	10	21	21	21
N	18	2	5	9	16	18	20

Figure 3.11: Request for key $21 = 10101_2$ starting at node $18 = 10010_2$

Koorde has a second variant where it can use a base-$O(\log n)$ (say base-k) de Bruijn graph. In this variant, node m uses pointers to k predecessors of km. This represents a node degree of $O(\log n)$, but, on the other hand, it improves the path lengths to $O((\log n)/\log\log n)$, which is still optimal.

Routing Geometry

The routing geometry of Koorde is comprised of two parts: a basic Chord ring (without the finger tables) plus a variation of the de Bruijn graph represented in Figure 3.7. Neither of the variants of Koorde has any flexibility concerning either the selection of paths or the selection of neighbors. However, comparison might not be entirely fair. In the second variant of Koorde, with $O(\log n)$ neighbors, the effect of choosing alternative neighbors or alternative routes is not very clear. Although this would affect the optimality of Koorde, fact is that most remaining DHTs are already suboptimal.

3.1.10 TOPLUS

Overview

In most peer-to-peer systems there is a mismatch between the logical and the physical networks. As a consequence, paths selected by routing schemes in peer-to-peer networks may be considerably longer than the corresponding paths in the physical network. The TOPLUS DHT (Garcés-Erice *et al.*, 2003) addresses this problem. TOPLUS organizes peers in groups according to their IP addresses. Groups are then organized into a new higher-order group, which is then organized into a new even higher-order group, and so on until, at the highest level, there is only one group, which includes all possible nodes. To increase correlation between group and network structure, TOPLUS gets topological information from Border Gateway Protocol (BGP) tables.

Let $H_N(X)$ be the lowest-order group including node X; let $H_{N-1}(X)$ be the second lowest-order group including $H_N(X)$, until $H_0(X)$, which includes all groups and all nodes. Node X must know all the nodes of group $H_N(X)$; at level $N-1$, node X must know at least one node from each group that is sibling of $H_N(X)$ (i.e., some node Y such that $H_{N-1}(X) = H_{N-1}(Y) \wedge H_N(X) \neq H_N(Y)$); the same applies for all other levels until level 0. The collection of all IP addresses known to X is the routing table of X.

To route, nodes use a metric derived from a simple longest-prefix matching known as the "XOR metric", which is also used in Kademlia (Maymounkov & Maziéres, 2002). A distance between two identifications j and k under the XOR metric is defined as $d(j,k) = \sum_{v=0}^{31} |j_v - k_v| \cdot 2^v$, where j_v is the v-th bit of j. The difference of the XOR metric to the longest-prefix match is that the former always breaks ties when prefixes have the same length (see Garcés-Erice *et al.*, 2003, for details). Given the routing table structure of node X, it is trivial to prove that routing will always converge. In fact, node X always knows some node that is closer to destination D under the XOR metric (e.g., some node F belonging to some sibling group $H_1(F)$).

While capable of achieving a stretch as low as 1.17 compared to the IP routing, TOPLUS has a number of drawbacks (Garcés-Erice *et al.*, 2003). The most obvious one is that consistent hashing is no longer able to balance load among the nodes, because nodes may not be evenly spread in the identification space. As a consequence, some nodes may receive an unfair share of the load. Another practice that might be used in other peer-to-peer systems would be to assign several virtual nodes to the same powerful peer. This is also difficult in TOPLUS. Another issue is the possibility of correlated node failures that may bring down an entire set of related IP addresses, thus reducing the effectiveness of using peers with neighboring addresses to increase availability.

Routing Geometry

The routing geometry of TOPLUS is hybrid in the sense that TOPLUS creates a tree of groups of nodes. Nodes have pointers to groups at higher levels of the tree as well as pointers to sibling groups at each level. TOPLUS only requires nodes to have complete knowledge of the lowest level group of the node. Selection of neighbors in TOPLUS is very flexible. The same does not happen with route selection. The tree structure of TOPLUS allows only a single option for the optimal route.

3.1.11 Comparison of the DHTs

To compare the DHTs, we evaluate their most significant features:

performance and scalability as the network size increases up to thousands or millions of nodes, scalability depends heavily on the behavior of the following factors: ability of nodes to share the load, growth of path lengths versus degree of nodes and congestion at the nodes.

While most DHTs assume that load sharing is ensured by hashing (and possibly randomization), figures for path lengths and node congestion may vary. To evaluate performance and scalability, we will use the theoretical bounds as they are presented by their authors;

tolerance to faults to evaluate tolerance to faults, we base our evaluation on the interesting work of Gummadi *et al.* (2003a) that defines the concept of "static resilience" of a DHT. Although tolerance to faults depends on many different mechanisms, static resilience measures the ability of the DHT to resist to node failures prior to any attempt of reconstruction. Static resilience strongly depends on the particular DHT and on the routing geometry. We evaluate the static resilience of the different DHTs in Section 3.1.11. Other recovery mechanisms may be applicable to more than one DHT and consequently they are less prone to comparisons;

self-configurability to evaluate the self-configurability of the DHTs, we focus on the ability of the network to choose better paths and on churn[8]. However in the case of churn, there is little data available to compare the DHTs and additionally, many techniques are applicable to all the DHTs. Therefore, we will focus on generic measures to improve resistance to churn and, as a consequence, self-configurability. To evaluate the ability of the network to choose better paths, we outline conclusions of Gummadi *et al.* (2003a), while to infer behavior under churn and to mitigate this problem, we outline conclusions of Rhea *et al.* (2004); Liben-Nowell *et al.* (2002); Li *et al.* (2004a).

[8]The site http://searchcrm.techtarget.com defines the "churn rate" as the number of costumers who discontinue a service during a specified time period divided by the average total number of costumers over the same time period.

Performance and Scalability

Path lengths are measured in terms of hops traveled by a message in the overlay network, while the degree of a node is the number of its overlay neighbors. These parameters are tightly connected and cannot be simultaneously reduced, as any n-node network meets its fundamental limits in the inequality $d^{h+1}/(d-1) \geq n-1$. $d^{h+1}/(d-1)$ is an upper bound for the number of nodes within h hops for a given maximum node degree d. In fact, for $O(1)$ node degree, expected path lengths can be, at best, $O(\log n)$, while for $O(\log n)$ node degree, expected path lengths cannot be shorter than $O(\log n/\log\log n)$ (Kaashoek & Karger, 2003). The path length/node degree trade-off is one of the central criterion used to evaluate a DHT.

Besides path lengths and node degree, it is also common to find a theoretical derivation of congestion at nodes in the papers where authors present their DHTs. Table 3.1 summarizes these figures for the DHTs presented before, when available (see also Fraigniaud & Gauron, 2003a; Malkhi *et al.*, 2002). For comparative purposes, we have also included networks with small-world characteristics. A small-world has a constant node degree and poly-logarithmic diameter. Some notable works that study the creation of small-worlds are those of Kleinberg (2000); Duchon *et al.* (2005); Barrièrere *et al.* (2001). One important aspect that we must be aware of is that there may be no correspondence between these theoretical bounds and the results observed in practice, because the $O()$ notation hides a constant factor that may vary widely. For instance, experimental results of Gummadi *et al.* (2003a) tend to demonstrate that, for instance, the butterfly network is very inefficient when compared to networks with $O(\log n)$ node degree.

Table 3.1: Comparison between *expected* performance of several peer-to-peer systems

P2P system	Node degree	Network diameter	Node congestion
small-worlds	$O(1)$	$O(\log^2 n)$	$O\left((\log^2 n)/n\right)$
Chord	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
CAN	$O(d)$	$O(dn^{1/d})$	$O(dn^{1/d-1})$
Pastry	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
Tapestry	$O(\log n)$	$O(\log n)$	$O((\log n)/n)$
D2B	$O(1)$	$O(\log n)$	$O((\log n)/n)$
Viceroy	$O(1)$	$O(\log n)$	$O((\log n)/n)$
Koorde cfg. 1	$O(1)$	$O(\log n)$	$O((\log n)/n)$
Koorde cfg. 2	$O(\log n)$	$O((\log n)/\log\log n)$	$O((\log n)/(n\log\log n))$

Table 3.2: Flexibilities of the different DHTs architectures

Property	Optimal paths	Neighbor selection
Ring (Chord)	$O(\log n)$	$n^{\log n/2}$
Hyper-cube (CAN)	$O(\log n)$	1
Torus (eCAN)	1	$n^{\log n/2}$
Hybrid (Pastry)	1	$n^{\log n/2}$
Tree (Tapestry, TOPLUS)	1	$n^{\log n/2}$
de Bruijn (D2B, Koorde)	1	1
Butterfly (Viceroy)	1	1

Tolerance to Faults

With respect to tolerance to faults, we make an analysis of the static resilience of a DHT, which depends on the routing geometry. If a given routing geometry offers more alternative paths to a destination, one may expect that a DHT based on this geometry will resist to a larger number of broken links. Note that this is largely independent of two mechanisms that can be included by virtually all DHTs: *i*) data replication to prevent data loss and *ii*) active recovery mechanisms to repopulate the routing tables. As noted by Gummadi *et al.* (2003a), if DHT A has better static resilience than DHT B, then DHT B needs to use quicker, more expensive, active recovery schemes, to offer a comparable resilience.

Table 3.2 summarizes the flexibility considerations that we made in the previous sections (Gummadi *et al.*, 2003a). Experimental results of Gummadi *et al.* (2003a) tend to confirm the hypothesis that more paths provide better resilience. This result is consistent with the "optimal paths" column of Table 3.2. In particular, authors have observed that the tree and the butterfly have a low resilience, while the ring and the hyper-cube have the highest resilience of the analyzed DHTs[9].

Self-Configurability

Path Latency As we referred before, there is often a mismatch between the theoretical bounds for the trade-off between path lengths and node degrees and the observed performance. This difference may result from two facts. First, the constant hidden in the $O()$ notation may vary widely. Second, this trade-off does not convey any information about latency. In a DHT where nodes pick their identifications in a random way, there are two main techniques to reduce latency (Gummadi *et al.*, 2003a): *i*) Proximity Neighbor Selection (PNS): pick nearby neighbors when creating the routing tables (if possible) or *ii*) Proximity Route Selection (PRS): when more than a single path is available to a destination, try to select paths through neighbors with a small latency. In general, PNS consistently yields better results than PRS. According to Gummadi *et al.* (2003a), this is a consequence of the fact that PNS has more alternatives than PRS. Many DHTs allow a large number of candidates for a single routing entry[10], while,

[9] Gummadi *et al.* (2003a) have analyzed the following geometries: tree (Tapestry), hyper-cube (CAN), ring (Chord), butterfly (Viceroy), xor (Kademlia (Maymounkov & Maziéres, 2002)) and hybrid (Pastry).

[10]However, it is not possible to measure the latency to each one of the nodes and therefore, nodes must pick a neighbor from a small sampling subset.

at routing time, the candidates for the next hop are restricted to the few options available in the routing table of the node. Interestingly, a combination of the two methods is also possible and yields good results.

Now, the question that we want to answer is which DHTs allow PNS and which do not. This is obviously related to the "neighbor selection" column of Table 3.2. Whenever more than one choice for the routing table is available a node may try to pick a neighbor with a small latency. Therefore, Chord, eCAN, Pastry, Tapestry and TOPLUS can select among multiple different neighbors, while the remaining cannot. One of the conclusions of Gummadi *et al.* (2003a) is that the routing geometry (i.e., tree, ring, etc.) does not seem to have any influence in the gains of PNS. The only thing that matters is whether or not it is possible to implement it. Concerning PRS, the DHTs that allow multiple paths, e.g., Chord or CAN, are also more prone to support "runtime" selection of neighbors.

Resistance to Churn Churn is one of the most crucial problems of peer-to-peer systems. Informally, churn refers to the rapid membership changes that may occur. This was first observed in the context of peer-to-peer applications like Napster, Gnutella or FastTrack[11]. However, given the lack of widespread large-scale applications based on DHT (Rhea *et al.*, 2003), there is little or no consistent and thorough analysis of churn in DHTs. For this reason, we restrict ourselves to outline the conclusions of previous work of Rhea *et al.* (2004); Liben-Nowell *et al.* (2002); Li *et al.* (2004a). Some of this work points to conclusions that we consider in the mechanism to create long range contacts, which we present in chapter 6.

The basic problem with churn is that there is a communication cost as-

[11]See http://en.wikipedia.org/wiki/Fasttrack.

sociated with the maintenance of the DHT. On the other hand, it has been observed that nodes can have very small up times (also known as "session times") in a DHT (Saroiu *et al.*, 2002; J. Chu, 2002; Sen & Wang, 2002; Bhagwan *et al.*, 2003; Gummadi *et al.*, 2003b). Some of these figures point to session times as low as a single minute for 50% of the nodes. This may occur if nodes that join a DHT supporting a file sharing system leave immediately after they find or give up finding a file the user was trying to retrieve. Hence, the basic trade-off is: either spending bandwidth updating outdated routing table entries or accepting a large latency resulting from not updating those entries. Li *et al.* (2004a) experimentally compared four DHTs (Chord, Pastry, Kademlia (Maymounkov & Maziéres, 2002) and Kelips (Gupta *et al.*, 2003)) to find the feasibility limits of this trade-off between "average live bandwidth" and "average lookup latency". They concluded that given the right parameter settings this trade-off is identical for all the analyzed DHTs. Qualitatively, this trade-off curve looks like the function $y = 1/x + k$, where x is the live bandwidth, y is the lookup latency and k is the minimum achievable latency. This means that it is useless to spend more bandwidth beyond a certain point as this will not reduce latency below k. On the other hand latency will sharply increase if the system tries to save bandwidth below a critical point of the function.

Although interesting from a theoretical perspective, the basic problem is to make the system converge to the intended pace of substitution of stalled routing table entries. Rhea *et al.* (2004) clearly show that some systems start failing most lookup requests when the churn rate increases. Additionally, they show that this results from an inadequate policy of replacing stalled entries. These systems replace entries in a *reactive* way, i.e., whenever a node detects that a given neighbor failed it tries to replace that neighbor. This may create a positive feedback cycle where the node congests its own access link trying to refresh its routing table and consecutively declaring more and more neighbors as dead as it cannot communicate with them. Hence the authors identify the three most important factors that may improve the response of a DHT to churn:

- periodic instead of reactive recovery from failures;
- calculation of message timeouts during lookups;
- choice of nearby over distant neighbors.

The first of these ideas is to recover at previously scheduled moments, instead of immediately reacting to failures. On the other hand, correctly estimating the timeouts is also a central aspect of resistance to churn. If a timeout is too short, a new request might be issued before the first one is able to reach the requester, thus congesting the network even more. On the other hand, an unnecessarily longer timeout will have a negative impact on latency. Finally, the third issue is nothing more than PNS. As we have seen, choosing nearby, instead of distant neighbors, is only possible with certain routing geometries. Although Rhea *et al.* (2004) use the Bamboo (Rhea *et al.*, 2003) DHT[12] to support their claims, the use of mechanisms based on these conclusions may also increase the resistance of most other DHTs to churn. In this thesis we will focus on the first of the three factors. In chapter 6 we will develop a mechanism that uses a lazy recovery of lost neighbors. Instead of reacting immediately to failures or making periodic recoveries, our scheme will postpone substitution of neighbors to the moment when they need to be used, thus trading latency for

[12]The routing scheme and the routing geometry of Bamboo are identical to Pastry. Most of the differences lie in the maintenance mechanisms.

bandwidth utilization. Additionally, we do not create new links (LRCs) for nodes when they enter, but only when messages go through their nodes. Again, this trades latency for bandwidth. The advantage of doing this is that even for sharp increases of the churn rate, maintenance traffic is kept under control (including the creation of links).

3.2 Position-Based DHTs

All the DHTs presented before assume that there is a network with operational routing underneath. Then, most of these DHTs create a logical overlay network where nodes perform routing in a way that is strongly decoupled from the data network. This creates an overhead for every lookup operation of the DHT. Given this problem, we can use position to eliminate the mismatch between the data network and the logical network. Therefore, inheriting from the work of Bose & Morin (1999) and Ratnasamy *et al.* (2002), we can simultaneously solve the routing and the DHT problem with a single network. Hence, like in the routing problem, construction of a DHT can strongly benefit from the use of positional information, whenever there is a strong relation between position and topology. While the most obvious application of this idea is to wireless *ad hoc* networks, wired networks can also benefit from this (Padmanabhan & Subramanian, 2001).

3.2.1 The Case of Wireless *Ad Hoc* Networks

None of the DHTs described before is suitable for wireless *ad hoc* networks. The main reason for this is that they need an underlying data network with operational routing to work. However, routing is probably the most important issue to solve in wireless *ad hoc* networks. Therefore, the problem of creating a DHT for wireless environments has deserved fewer efforts (there are, nevertheless, some examples, e.g., Ratnasamy *et al.*, 2002; Eriksson *et al.*, 2004; Pucha *et al.*, 2004). As a result, if routing *per se* is such a complicated task, there is little sense in coming up with overlay networks that need an independent routing facility. As a consequence, DHTs for wireless *ad hoc* networks tend to solve the routing and the DHT problem in an integrated fashion. This is the case of Geographical Hash Table (Ratnasamy *et al.*, 2002) (GHT). Despite not being localized (according to the definition given in Section 2.3.3), in most circumstances GHT will only need information of nearby nodes to operate correctly.

In GHT there is a space of identifications for nodes and keys. Unlike other DHTs, this space is not virtual, but physical. GHT relies on the GPSR protocol with little modifications to operate. GHT uses two additional related concepts: the "home node" and the "home perimeter". The *home node* of a point is the node which is geographically closest to that point in space. The *home perimeter* of a point is the set of edges that encloses that point[13]. Assume that the network supporting the DHT is routing to some destination D. In general, D will not correspond to any node. However, standard behavior of GPSR ensures that a packet always reaches the home node, say H. Since this home node is not the intended destination, the packet passes from greedy to perimeter mode and starts circulating in the face of H that encloses D (home perimeter). Since there is no other node closer to D than H is, the packet will, once again, reach H. Now, unlike standard GPSR, H will not drop the packet but will receive it, because it may be certain that it is the home node of D. Details on the maintenance of the DHT can be found in the work of Ratnasamy *et al.* (2002).

[13]The reader should keep in mind that GPSR needs a planar graph.

One of the shortcomings of this DHT is that path lengths tend to increase as network density increases (since the underlying graph is planar, edges become shorter when node density increases). This problem is inherent to GPSR and as we referred in Section 2.8, one can use clustering to solve it. Perhaps the simplest way of clustering nodes in a position-based algorithm is to divide the space into cells of predictable size (see Figure 2.10). This clustering can then be used to create a DHT. As a result, the cell becomes the addressing unit of the DHT and the nodes inside each cell must cooperate to manage the keys. Gupta *et al.* (2001) suggest this form of clustering. Li *et al.* (2004b), take this idea a step further. They create a DHT on top of a logical structure of cells, which is part of a more general peer-to-peer information sharing architecture. A similar idea exists in the work of Ye *et al.* (2002), which presents a Two-tier Data Dissemination (TTDD). TTDD creates a grid of dissemination for each type of data. This grid also follows a geometrical arrangement. Then, TTDD uses flooding within the local area, while global dissemination is achieved through the grid, for the sake of efficiency. In Chapter 5 we present "Cell Hash Routing" (CHR), which is a DHT for wireless *ad hoc* networks, based in cell clusters. When compared to these previous approaches, uniqueness of CHR comes from the simplicity of some of its solutions. In the Unit Disk Graph (UDG) model, CHR is capable of routing with a very minimal pre-processing algorithm. As a result, we ensure routing convergence at a small cost. Unlike CHR, in the previous approaches, routing from one cell to the next requires the normal node-to-node routing. As a consequence, voids in the network are much more difficult to handle.

3.3 Summary

In this chapter we surveyed and compared some of the most well-known DHTs in the following aspects: performance and scalability (ability to share the load, path length vs. node degree trade-off and node congestion), tolerance to faults (static resilience) and self-configurability (improving path latency and resistance to churn). With exception of position-based DHTs, all other DHTs presented are implemented as a peer-to-peer overlay network and require operational routing underneath. The position-based DHTs have the advantage of not using a logical overlay network and, therefore, are much lighter and can be used in wireless *ad hoc* networks. In wired networks, the main challenge for this type of DHTs is to achieve a good path length vs. node degree trade-off, specially in networks with an uneven distribution of nodes. In Chapter 6 we propose a mechanism called "Hop Level" to tackle this problem. We also explore position-based DHTs for wireless *ad hoc* networks in Chapters 4 and 5.

4

A Wireless DHT Based on a Delaunay Triangulation

The distributed hash tables (DHTs) that we present in this thesis are inspired on the Geographical Hash Table of Ratnasamy *et al.* (2002) (GHT). As we described in Section 3.2.1, to increase the scalability of the DHT, this solution integrates the routing scheme with the lookup operations of the DHT. Our first approach is a direct evolution of GHT that uses a denser underlying network of nodes. More precisely, we use the $PLDel(V)$ triangulation (Section 2.5.3) instead of a Gabriel Graph (*GG*) (Section 2.5.1), which is used in GHT. To create this $PLDel(V)$ triangulation, we present the "Fast Localized Delaunay Triangulation" (FLDT).

4.1 Overview of the Fast Localized Delaunay Triangulation

GHT uses the GPSR algorithm atop of a Gabriel Graph (Section 2.5.1). However, it is a well-known fact that the Gabriel Graph is a bad spanner of the Unit Disk Graph (UDG). This can be confirmed in previous work (Eppstein, 2000; Bose *et al.*, 2002) and we also give further evidence of this fact in Section 4.4. Therefore, by using a good spanner we can considerably improve routing performance of the DHT, which directly translates to shorter paths. Since the logic of the GHT relies on a planar graph, a triangulation is a good candidate to create a good spanner. For these reasons, the $PLDel(V)$ graph of Li *et al.* (2002) is perhaps the best option, because it is a $(4\sqrt{3}\pi)/9$-spanner of UDG (Section 2.5.3). Unfortunately, despite having a communication cost within a constant of the optimal ($O(n \log n)$), the algorithm of Li *et al.*, has a considerable overhead, because nodes need 4 communication steps to create the triangulation (we define a communication step as the period required for sending and then receiving one or more messages which are not causally related). To reduce the number of communication steps, we present the "Fast Localized Delaunay Triangulation" (FLDT). Unlike previous work, our algorithm builds $PLDel(V)$ in a single communication step, maintaining a communication cost of $O(n \log n)$. This represents a significant practical improvement over previous algorithms with similar theoretical bounds. The small cost of our algorithm makes feasible to use $PLDel(V)$ in real systems, instead of the Gabriel or the Relative Neighborhood graphs.

Therefore, our algorithm is well suited to wireless environments for the following reasons: *i*) it is very efficient as it requires just one communication step; *ii*) it is applicable to dynamic and asynchronous settings (see Section 4.5); *iii*) it is localized, only requiring nodes to receive information broadcast by direct neighbors, thus requiring a communication cost within a small constant of the optimum (assuming that a beacon message of $O(\log n)$ bits in an n-node network is necessary per node); *iv*) it requires nodes to keep track of only a constant number of neighbors in the average; *v*) under the constraint of preserving planarity, it builds a graph with good density (see Section 4.4).

4.2 Description of the FLDT Algorithm

In this section we present the FLDT algorithm that builds a $PLDel(V)$ graph. The FLDT algorithm is decentralized, as it does not rely on any centralized component, and localized, since nodes are only required to gather knowledge about some nodes in their 2-hop neighborhood. The algorithm builds a triangulation that ensures routing between any pair of nodes as long as $UDG(V)$ is connected. The algorithm consists of the following logical steps:

1. The *neighbor discovery* step. The purpose of this step is to allow nodes to discover their neighbors. For the sake of clarity, we first describe and analyze the algorithm in the context of a fixed setting, where all nodes know their neighbors *a priori*. The discussion of the use of our algorithm in the context of dynamic settings (that may require the exchange of BEACON messages) is postponed to Section 4.5.

2. The *triangulation* step. The purpose of this step is to let each node compute and advertise to its neighbors the relevant Delaunay triangulations. Based on the information collected during the neighbor discovery step, each node P locally computes a Delaunay triangulation. For convenience of exposition, we introduce the predicate $Delaunay\triangle_P(Q,R)$ that holds true at P if, according to the triangulation computed by node P, triangle $\triangle PQR$ should exist. $Delaunay\triangle PQR$ will also be used when referring to the predicate at no particular node. When $Delaunay\triangle_P(Q,R)$ holds at P, if $\angle QPR \geq \pi/3$, then P broadcasts a TRIANGULATE $\triangle PQR$ message to all nodes within range.

The purpose of the $\pi/3$ condition is to ensure that no node will issue more than 6 TRIANGULATE messages by its own initiative (as in Li *et al.*, 2002). Since no additional messages are sent in the following steps, total communication cost of FLDT is $O(n \log n)$. In practice, the constant involved in this bound is small, because, as we show in Section 4.4, each node announces less than 6 other nodes in average.

3. The *sanity* step. The purpose of this step is to let neighbor nodes eliminate inconsistent Delaunay triangulations. They do so by comparing triangulations computed locally with the triangulations computed by their neighbors in Step 2, as advertised by TRIANGULATE messages. Note that by processing TRIANGULATE messages, nodes may learn about new nodes that are not their direct neighbors. This addititional information will never create new Delaunay triangulations, as triangulations must be formed with direct neighbors. However, TRIANGULATE messages may invalidate some of the triangulations computed in Step 2. This may happen at P if: *i*) Q or R broadcast a TRIANGULATE message with some node T that invalidates $\triangle PQR$, i.e., $T \in \bigcirc PQR$, or *ii*) some node W sends a TRIANGULATE message with an intersecting triangle WXZ, where either X or Z invalidate $\triangle PQR$, i.e., $X \in \bigcirc PQR$ or $Z \in \bigcirc PQR$. Case *i*) ensures that a node only maintains a predicate if its neighbors are not aware of some node that invalidates it, while case *ii*) avoids the existence of intersections[1].

4. The *Gabriel edges* step. The purpose of this step is to add to the graph all missing Gabriel edges. Otherwise, despite always being correct, a Gabriel edge PQ for which no predicate $Delaunay\triangle_P(Q,R)$ holds at P (e.g., after switching to false in Step 3) would not be included by P. This will increase the density of the graph, while keeping $O(n)$ edges (note that a Gabriel edge always belongs to the Delaunay triangulation and can be determined locally without additional exchange of information).

Optimization. To simplify our algorithm, all TRIANGULATE messages

[1]Note that case *i*) can also prevent some intersections.

should be sent in a single control message. □

When comparing FLDT with previous solutions (Li *et al.*, 2002; Lan & Wen-Jing, 2002) one must notice that the simplicity of our algorithm comes from two insights, that we prove to be correct in Section 4.3. First, proposals sent in TRIANGULATE messages, alone, suffice to confirm or reject triangulations proposed by neighbors in their own TRIANGULATE messages (and vice-versa), i.e., there is no need to dedicated replies. This insight builds on the observation that two Delaunay neighbors do not need to agree on some predicate $Delaunay\triangle PQR$. It can hold at P but not at Q and R if these two latter nodes are out of range of each other. The fundamental issue is, in fact, to ensure that two nodes P and Q always agree on whether edge PQ should exist (Lemma 4.3). Second, if three nodes P, Q and R wrongly assume the existence of $\triangle PQR$, intersected by $\triangle WXZ$, such that one of the nodes of $\triangle WXZ$ is inside $\bigcirc PQR$, then P, Q and R will listen to the same TRIANGULATE message on $\triangle WXZ$, thus commuting the predicate $Delaunay\triangle PQR$ to false simultaneously at P, Q, and R (Lemma 4.5).

4.3 FLDT Creates $PLDel(V)$ in a Single Communication Step

From analysis of the algorithm, we know that nodes running FLDT use a single communication step. Hence, in this section we need to prove that FLDT builds, in fact, the graph $PLDel(V)$, under the UDG model. To prove this, we show in Lemma 4.6 that this algorithm builds a subgraph of $LDel^{(1)}(V)$ from which we remove all the intersections. However, we still need to prove that we do not delete some triangles unnecessarily. This is the main result of Theorem 4.1, which shows that some predicate *Delaunay* $\triangle(A,B,C)$ is switched to false only if $\triangle ABC \notin LDel^{(2)}(V)$ is in fact intersected by some Gabriel edge or by some triangle $XYW \in LDel^{(1)}(V)$. In other words, there are no illegitimate edges or triangles deleting triangles that despite not belonging to $LDel^{(2)}(V)$ could otherwise participate in the triangulation and belong to $PLDel(V)$. Theorem 4.1 stands not only on Lemma 4.6, but also on the other Lemmas that we present next. In all the proofs, we assume that there are no four co-circular nodes. Simple tie-breaking mechanisms can remove co-circularities, if they ever occur in practice.

Lemma 4.1 *In the UDG model, if two edges AB and XY intersect, then at least one of the nodes is within communication range of the other three.*

Proof 4.1 *Assume, without loss of generality that X and Y are outside of* $d(A,B)$ *(otherwise the Lemma would immediately follow). In this case it is not possible for both A and B to be outside* $d(X,Y)$*, because circumferences can intersect at two points at most.*

Lemma 4.2 *If, at the end of the algorithm, non-Gabriel edge AB exists at A and B, there must be some node C, such that* $C \in d(A,B)$ *maximizes* $\angle ACB$ *and Delaunay* $\triangle(A,B,C)$ *holds at A and B.*

Proof 4.2 *Refer to Figure 4.1. Consider the circle* $d(A,B)$ *(not shown), which is divided in two halves by AB. One and only one of the halves has nodes inside, otherwise either AB would be a Gabriel edge or both A and B would know that AB would not belong to the Delaunay triangulation. Assume that C is the node that maximizes* $\angle ACB$*. Therefore,* $\forall C' \in d(A,B), C' \notin \bigcirc ABC$*, while* $C \in \bigcirc ABC'$*. Additionally, if A or B were aware of some node D such that* $D \notin d(A,B)$ *and*

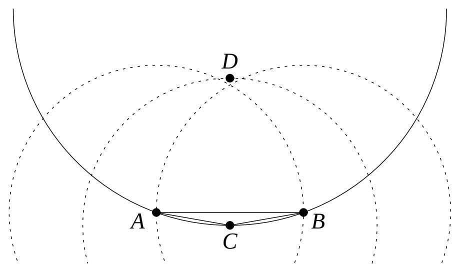


Figure 4.1: *A*, *B* and *C* disagree on $\triangle ABC$

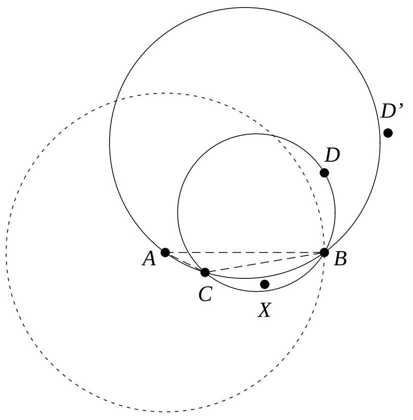


Figure 4.2: *A* and *B* do not agree on $\triangle ABC$

$D \in \bigcirc ABC$, we would have a contradiction, because edge AB could not exist at both nodes A and B. The Lemma follows.

Lemma 4.3 *If after the Delaunay triangulation computed at step 2 of the algorithm, Delaunay $\triangle_A(B,C)$ holds, but edge AB cannot exist at B, B will send a* TRIANGULATE *message with at least one node $D \in \bigcirc ABC$.*

Proof 4.3 *Refer to Figure 4.2. Since non-Gabriel edge AB exists at A, C must be inside $d(A,B)$ (Lemma 4.2). In this case, AB cannot exist at B if Delaunay$\triangle_B(X,D)$ holds at B for some nodes X and D and XD intersects AB (assume without loss of generality that X and C are on the same side of AB, possibly with $X = C$). $D \in \bigcirc ABC$, because otherwise $\bigcirc BXD$, would contain A which would be a contradiction (any such D' in the figure would have to be outside $d(A,B)$ and closer to B than to A: $\bigcirc D'BC$ would intersect $\bigcirc ABC$ at B and C, thus for $X = C$ it would contain A; if $X \neq C$, $\bigcirc D'BX$ would intersect $\bigcirc D'BC$ at B and D', thus containing the part of $\bigcirc D'BC$ that would contain A). Since, $\angle XBD > \angle ABD > \pi/3$, B will send information of D in its* TRIANGULATE *messages.*

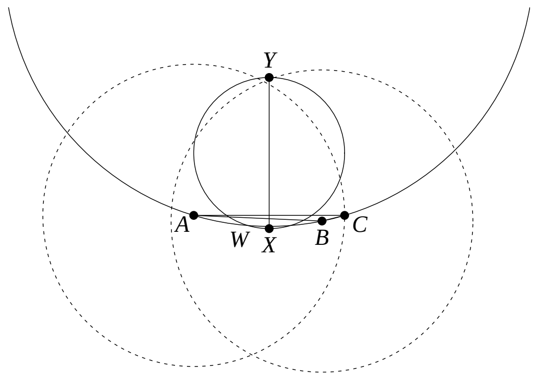


Figure 4.3: Possible intersection

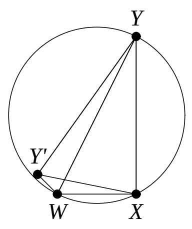


Figure 4.4: Triangle $\triangle XYY'$ may not exist

Corollary 4.1 *If, at the end of the algorithm, non-Gabriel edge AB exists at A, there must be some third node C, such that $C \in d(A,B)$ maximizes $\angle ACB$ and Delaunay $\triangle(A,B,C)$ holds at A, B and C.*

Proof 4.4 *If Delaunay $\triangle_A(B,C)$ holds at A, by Lemma 4.3 it must also hold at B at the end of the algorithm. Assume that C maximizes $\angle ACB$. Since $C \in d(A,B)$, $\angle ACB > \pi/2 > \pi/3$. Now assume that Delaunay $\triangle_C(A,B)$ does not hold, because C is aware of at least one node inside $\bigcirc ABC$. If such node, say D, was announced by another node, say X, such that XD intersects $\triangle ABC$, by Lemma 4.1, A and B would also have listened to X. Hence, we are left with two possibilities: either C includes AC in its triangulation or it does not. If it does not, by Lemma 4.3, Delaunay $\triangle_A(B,C)$ could not hold at the end of the algorithm. Therefore, assume that AC exists. In this case, there is some node $D \neq B$, such that Delaunay $\triangle_C(A,D)$ holds and CD intersects AB. $D \in \bigcirc ABC$, or otherwise, $\bigcirc ACD$ would contain B, which would be a contradiction. Since A is not aware of D, we have that $\angle ACD > \pi/3$. Hence, C would announce this triangle which would make A switch Delaunay $\triangle_A(B,C)$ to false. The same reasoning could be made to B and the Lemma follows.*

Lemma 4.4 *If X is aware of a node $Y \in \bigcirc ABC$, X may only participate in edges, $XY_1, XY_2, \ldots, XY_n$ intersecting AB if $Y_i \in \bigcirc ABC, \forall i \in \{1, \ldots n\}$.*

Proof 4.5 *Refer to Figure 4.3. The first thing that this Lemma assumes is that there is already an intersection between AB and XY, such that $Y \in \bigcirc ABC$. Note that by Lemma 4.1, A and B are aware of X, which implies that $X \notin \bigcirc ABC$. Now assume that a second intersection exists between AB and XY_1, such that $Y_1 \notin \bigcirc ABC$. In this case, XY_1 could not be a Gabriel edge and there could be no node Z such that Delaunay $\triangle_X(Y_1, Z)$ would hold, because any circle going through X and Y_1 would have to contain either A or B. This contradiction proves the Lemma, because we first assumed that XY_1 existed.*

Lemma 4.5 *Assume that after computation of the Delaunay triangulation, edge AB exists at A as part of Delaunay $\triangle_A(B, C)$. Assume that there is some edge XY intersecting AB and some node W at the same side of XY as A is (possibly $W = A$), such that Delaunay $\triangle_X(Y, W)$ holds and $Y \in \bigcirc ABC$. If at the end of the algorithm edge XY still exists, A must have switched Delaunay $\triangle_A(B, C)$ to false.*

Proof 4.6 *Refer to Figure 4.3. Assume that Y minimizes the angle $\angle AXY$ (by Lemma 4.4 such node Y must exist). If $W = A$, the Lemma follows, because $\angle AXY > \pi/3$, which makes X announce the triangle with node Y. Assume now that Delaunay $\triangle_X(Y, A)$ does not hold. By hypothesis, there must be some other node, say W, at the same side of the line going through XY as A is, such that Delaunay $\triangle_X(W, Y)$ holds. If $\angle WXY > \pi/3$ the Lemma follows. Otherwise, if Delaunay $\triangle_W(X, Y)$ also holds, $\angle XYW < \angle AYB < \pi/3$ implies that $\angle XWY > \pi/3$ and the Lemma also follows. Hence, we focus on the case where Delaunay $\triangle_W(X, Y)$ does not hold. W is aware of Y, because $\angle WXY < \pi/3$ and $\angle XYW < \pi/3$.*

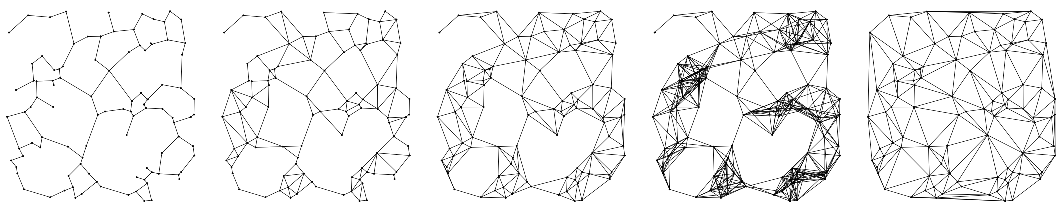
Also, if WX does not exist at W this Lemma follows from Lemma 4.3. For this reason, let us consider the case where there is some node $Y' \in \bigcirc XYW$ such that Delaunay $\triangle_W(X,Y')$ holds. Assume that WY' intersects XY. In this case, $Y' \notin d(X,Y)$ or otherwise X and Y would be aware of Y'. This means that $W \in d(X,Y)$ and by Corollary 4.1 XY could not exist at the end of the algorithm, thus contradicting the initial hypothesis. Hence, we assume that XY' does not intersect XY. We now that $\angle XWY' > \pi/3$. If WY' intersects AB the Lemma follows from Lemma 4.1. Hence, consider that WY' does not intersect AB. Refer to Figure 4.4. This means that YY' intersects AB. In this case, from Lemma 4.1, Y and Y' are within reach of each other. This is a contradiction, because we would need to have $\angle XYY' < \angle AYB < \pi/3$ opposing to the longest side of the triangle $\triangle XYY'$. This proves the Lemma.

Lemma 4.6 *FLDT creates a subgraph of $\mathbf{LDel}^{(1)}(V)$ without intersections.*

Proof 4.7 *The first thing we show is that FLDT creates, in fact, a subgraph of $\mathbf{LDel}^{(1)}(V)$. An edge AB that exists in the final graph must be either a Gabriel edge or an edge for which Delaunay $\triangle(A,B,C)$ holds at A, B and C (Corollary 4.1). This means that the final graph is a subgraph of $\mathbf{LDel}^{(1)}(V)$. Since, by Lemmas 4.5, there can be no intersections, the Lemma follows.*

Theorem 4.1 *FLDT builds $\mathbf{PLDel}(V)$.*

Proof 4.8 *$\mathbf{PLDel}(V)$ is comprised of all triangles of $\mathbf{LDel}^{(1)}(V)$,* except *intersecting triangles that do not belong to $\mathbf{LDel}^{(2)}(V)$. From Lemma 4.6, the final graph is a subgraph of $\mathbf{LDel}^{(1)}(V)$. However, some triangles belonging to $\mathbf{LDel}^{(1)}(V)$ may be deleted if they have intersections. What we need to prove is that we only remove triangles not belonging to $\mathbf{LDel}^{(2)}(V)$ that intersect with other triangles that belong to $\mathbf{LDel}^{(1)}(V)$.*



(a) *RNG* (b) *GG* (c) *PLDel* (d) *UDG* (e) *DT*

Figure 4.5: Example of graphs

Hence, it must not happen that some $\triangle ABC$ is deleted by non-Gabriel edge XY and nor edge XY nor any other edge intersecting $\triangle ABC$ exist in the final graph. Assume without loss of generality that $Y \in \bigcirc ABC$ and that XY intersects AB. Since XY is not a Gabriel edge and $\exists W_1 \in d(X,Y)$, such that W_1X or W_1Y intersects AB (note that $||W_1X|| < ||XY||$ and $||W_1Y|| < ||XY||$). Given that $A,B \notin d(X,Y)$ and $A,B \notin d(W_1,X)$ if intersection is with W_1X ($d(W_1,Y)$ if intersection is with W_1Y), it follows that even if the intersecting edge $\notin LDel^{(1)}(V)$, we can inductively repeat the reasoning until we find one intersecting edge $\in LDel^{(1)}(V)$. Hence, even if AB is deleted due to some edge $XY \notin LDel^{(1)}(V)$, there must be some other edge $\in LDel^{(1)}(V)$ that will legitimately delete AB. Theorem follows.

4.4 Evaluation

In this section, we compare *i*) routing performance in each of the following graphs: *RNG*, *GG*, *PLDel*, *UDG* and *DT* and *ii*) signaling cost of FLDT versus the algorithm of Li *et al.* (2002). Figure 4.5 illustrates the graphs in a network of 100 nodes. We have used the GPSR routing algorithm (Karp & Kung, 2000) in all graphs, except *UDG*, which is not planar. In *UDG* we have used the greedy routing algorithm. Results for the *DT* are depicted only to serve as a reference, because, as we have discussed before,

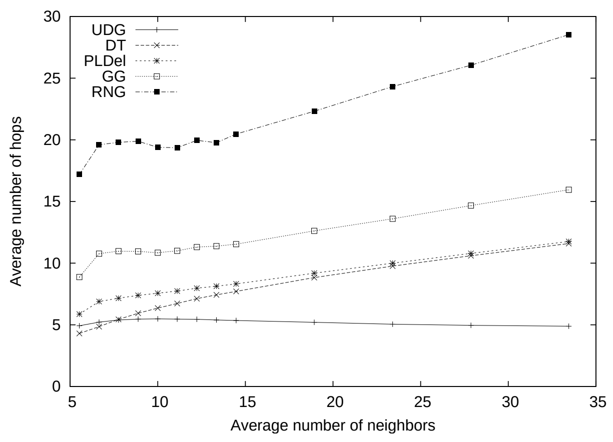


Figure 4.6: Average number of hops

such triangulation is not possible in a wireless environment. Since node density has a crucial impact on the performance of routing algorithms, in our experiments, we have distributed a variable number of nodes (between 140 and 600) inside a square of fixed side (7.5 times the communication range). The reader should notice that density cannot be arbitrarily reduced, because disconnected topologies would result with high probability. On the other hand, increasing node density will benefit *UDG*, because greedy routing will converge with increasingly higher probability and, unlike the remaining graphs, paths will become shorter.

Figure 4.6 shows the average path length in number of hops (for paths where greedy did not fail), while Figure 4.7 depicts the percentage of failures for the greedy routing algorithm in the *UDG* graph. Both curves are functions of the average number of neighbors of a node[2]. From the figures, it is quite evident that when node density is high, no subgraph can do better than *UDG*, unless memory usage is an issue and a node does not want to maintain all its neighbors. In this case, *PLDel* may be a good

[2]For a node whose communication (unit) disk is entirely inside the simulation square.

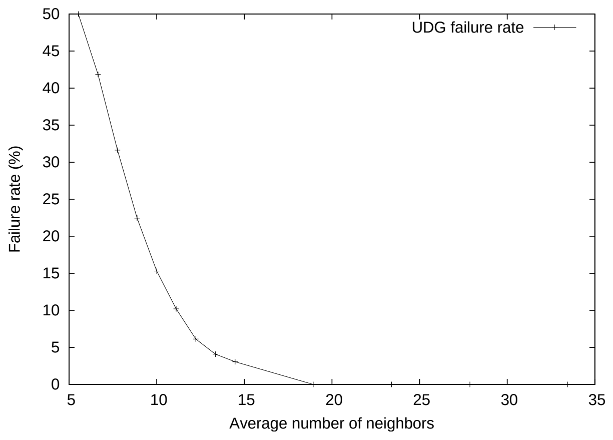


Figure 4.7: Failure rate in the *UDG*

option, because nodes need to maintain only a constant number of neighbors in average. On the other hand, when node density decreases, *PLDel* is definitely the preferable choice, because it achieves the best performance among the algorithms that ensure routing convergence. Since the possibility of a greedy routing failure always exists, no matter how large node density is, it may also be a good idea to maintain two graphs in memory: *UDG* and *PLDel*. The point is to use greedy in *UDG* whenever possible for performance reasons and switch back to a right-hand rule algorithm and to *PLDel* in case greedy fails. Such solution has the advantage of being oblivious to node density. It is also interesting to observe that the number of hops obtained in *PLDel* is typically quite close to that number in a *DT*, for high densities, where all edges are short, but the same is not true when node densities are small, because in these cases, *DT* uses long edges, thus saving many hops.

To complete our evaluation, we depict in Figure 4.8 the average number of neighbors announced by each node, in the algorithm of Li *et al.* and in our own algorithm. Note that whenever a triangle is announced, two

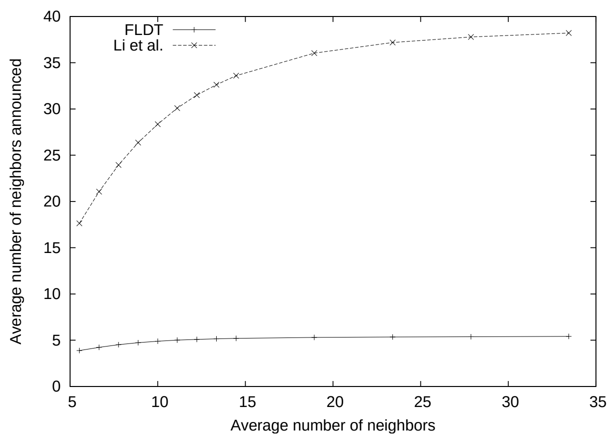


Figure 4.8: Average number of neighbors announced by each node

nodes are counted (the sending node is not counted). The algorithm of Li *et al.* also needs to announce Gabriel edges, which are counted as only one node (again, sending node is not counted). We can see that the number of nodes announced stabilizes in both algorithms as the density increases, and that our algorithm announces approximately between 5.2 and 7 times fewer nodes for the densities of interest. Furthermore, while our algorithm needs a single communication step, the algorithm of Li *et al.* needs 4 steps. Therefore, we believe that these results show that our algorithm builds *PLDel* very efficiently.

4.5 Application in Dynamic Settings

So far, we have described the execution of our algorithm in a static setting, where a node knows *a priori* all its neighbors. We now discuss the application of our algorithm in dynamic settings.

The application of any graph building algorithm in a dynamic setting requires a complementary mechanism to discover new nodes and to detect the departure/failure of existing nodes. In an optimized implementation, the concrete mechanisms to be used may depend on the physical and data link layer technology. However, in the literature (for instance, Li *et al.*, 2002; Lan & Wen-Jing, 2002) it is usually assumed that nodes periodically exchange BEACON messages. We would like to emphasize that our algorithm is particularly well suited for such setting, as TRIANGULATE messages can be easily piggybacked to (or even replace) BEACON messages. Therefore, when BEACON messages are required, our algorithm can be implemented with no additional messages, becoming extremely competitive with regard to the Gabriel or the relative neighborhood graphs, which are not good spanners of *UDG*.

Also, for the sake of simplicity, we have assumed perfect channels in our exposition (i.e., no message losses). However, in a dynamic setting, BEACON messages have to be exchanged periodically. This means that, at no additional cost in terms of number of messages exchanged, our algorithm may retransmit periodically TRIANGULATE and recalculate *PLDel* at the end of each period. Therefore, even if links are lossy, it can be shown that, as long as links are fair (*i.e.*, if a message is sent infinitely often by a process p then it can be received infinitely often by its receiver (Lynch, 1996)), any new node will eventually participate in the triangulation.

4.6 Discussion

Routing protocols for wireless *ad hoc* networks may benefit from using a planar and localized Delaunay triangulation to achieve good routing performance, while, at the same time, guaranteeing convergence. Our proposal is the FLDT algorithm, to build a well-known graph called *PLDel*.

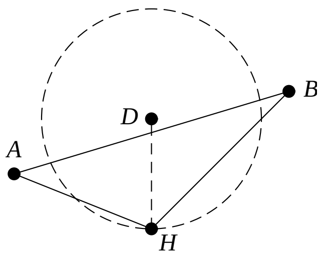


Figure 4.9: A and B must be aware of the home node H

Our experimental results show that *PLDel* can be used either to substitute the *UDG*, when node density is small, or as a complementary graph that ensures routing convergence for all node densities.

FLDT has a communication cost of $O(n \log n)$, which is within a constant of the optimal and requires a single communication step (unlike previous work, that requires 4 communication steps). We have also shown that the signaling cost of FLDT is much smaller than that of previous approaches, due to the small number of control messages. Furthermore, in dynamic settings that require the exchange of beacon messages, our algorithm requires no more messages than the algorithms used to build the very simple but inefficient *GG* or *RNG*. Therefore, due to its efficiency, our algorithm is of practical relevance in location-based wireless *ad hoc* networks.

4.7 Changes to GHT

GHT was designed for use with the *GG*. As is, one may question whether it cannot operate on top of $PLDel(V)$ in situations like the one depicted in Figure 4.9. Node H is the closest node to target D (which *does not* correspond to any node), but, at the same time, it can be in a different face, something that could never happen with a *GG* as we shall see in Lemma 4.7. Fortunately, this does not prevent GHT from working as we prove in Theorem 4.2.

Lemma 4.7 *In* $PLDel(V)$*, if the node* H *closest to destination* D*, is not in the home perimeter, then,* H *is always within reach of at least two nodes* A *and* B *of the home perimeter.*

Proof 4.9 *Refer to Figure 4.9.* A *and* B *must be outside the circle with ray* DH*. Since* AB *crosses this circle below* D*,* $\angle AHB > \pi/2$*. This means that* $H \in d(A,B)$*, which proves the Lemma. Also note that* AB *is not a Gabriel edge.*

Theorem 4.2 *Message delivery of GHT works on* $PLDel(V)$ *(i.e., message will reach node* H*, which is the home node of destination* D*).*

Proof 4.10 *If the home node is in the home perimeter, the proof of this Theorem depends on the correctness of GHT. Assume that the home node is not in the home perimeter. By Lemma 4.7, the home node must be a node of the home perimeter, or some other local minimum node* H' *within reach of at least two nodes of the home perimeter, say* A' *and* B'*. If there is only one such local minimum, GHT works, because the message must circulate at least once around the perimeter, thus reaching* A' *or* B' *and then* H'*. Now, assume that some local minimum* H' *is not the home node. When the message reaches* H' *it switches to greedy mode and it will never return to* H' *in greedy mode again. We have two possible cases: either* H' *forwards the message in greedy mode again (because it knows a neighbor closer to* D*) or it must use perimeter mode. If it delivers in perimeter mode, message will reenter the home perimeter (except edge* $A'B'$*) and will eventually reach some other nodes* A'' *or* B'' *or even* H'' *(possibly* $H = H''$*) closer to* D *than* H' *is. Either in this case or if* H' *delivers the message in greedy mode, message will become closer* *to destination D. This means that there is no other local minimum with exception of the home node H that can receive the message for a second time and still be the closest known local minimum.*

4.8 Summary

In this chapter we presented the algorithm FLDT to create and maintain a localized Delaunay triangulation ($PLDel(V)$) with a single periodical control message and optimal communication cost of $O(n \log n)$. FLDT makes it viable to replace the Gabriel graph in GHT by $PLDel(V)$ to achieve considerably shorter path lengths.

Notes

The algorithm FLDT results from the work of the author to create a Delaunay triangulation for wireless networks. It improves the previous work of Li et al. *(2002). Part of this work has been previously published in the following conference:*

- Filipe Araújo and Luís Rodrigues. Fast localized Delaunay triangulation. In *The 8th International Conference On Principles Of Distributed Systems (OPODIS '04)*, pages 81–93, Grenoble, France, December 2004. Springer-Verlag, LNCS 3544.

5 A Wireless Clustered DHT

Performance of Geographical Hash Table (GHT) tends to degrade when node density (versus communication range) increases, because as edges become shorter, paths tend to become longer. To overcome this problem, we use a clustered distributed hash table (DHT) called "Cell Hash Routing" (CHR). A clustered approach resists better to higher node densities as paths do not need to become longer. Furthermore, it can make a more rational use of the available bandwidth. One interesting outcome of this clustering is that the DHT can take advantage of positional information, but it does not need to be strictly positional, because the hash function may use other output spaces instead. For instance, it can use a deterministic numbering of the clusters. Nevertheless, since position plays a central role, we can still consider CHR to be a position-based DHT.

5.1 Overview of Cell Hash Routing

As we mentioned before, scarceness of resources (notably, bandwidth and energy) and node mobility turn routing into a challenging problem in wireless *ad hoc* networks. Therefore, when compared to a wired network, creating a DHT in a wireless network is more complicated. Like in previous work and similarly to Chapter 4, this motivates us to propose a single solution to the routing problem and to support the DHT. CHR copes with problems like limited available energy, communication range or node mobility. CHR uses positional information to organize a DHT of clusters instead of individual nodes. The purpose of this clustering mechanism is to solve the routing problem, because routing on top of clusters is more efficient than on top of individual nodes. CHR groups nodes into cells, according to their location. This method is inexpensive, because nodes do not need to exchange any dedicated message to determine their own cluster, which is a cell of predefined and globally known shape. Additionally, messages are not addressed to an individual node destination, because routing works at the cell level. Such an approach is well-suited to the world of small and simple wireless devices or embedded systems, where nodes may look for specific contents and not for peers. Consider the case of sensor networks. In these networks, it is often irrelevant to know the output of the individual sensors; it may suffice to compute some function like an average or a maximum temperature at some particular zone. Hence, globally known individual addresses are not necessary as sensor nodes are not individually queried.

The advantage of clustering is twofold. First, it creates a very structured and sparsely populated network of clusters, where we can apply a lightweight routing scheme. Second, the efficiency of the routing scheme is almost not affected by increasing node density. Furthermore, our routing scheme also scales with increasing network sizes, because it is localized. By using a location-based clustered approach, routing in CHR is scalable with respect to both, network size and node density. We believe that this scheme is powerful because it enables us to implement a DHT in a straightforward and efficient way. CHR can be used as a component of more complex architectures, like a publish/subscribe system. When compared to other solutions, uniqueness of CHR comes from the use of routing *on top* of a very regular clustered network, which is simultaneously the basis for the DHT. For these reasons, we believe that CHR is a simple and yet powerful adaptation of the DHT concept for wireless *ad hoc* environments.

5.2 Architecture of CHR

In CHR we divide the space into equally-sized cells that have the shape of squares, as the grid depicted in Figure 2.10. Division of the space into cells allows a simple definition of the network of clusters: all nodes inside the same cell belong to the same cluster. In a sense we will use the notion of cluster as a kind of "supernode", where interactions occur at the level of clusters. The result is a much sparser network, where the routing scheme consumes fewer network and node resources. Since cells are immutable, clusters can receive an identification that does not change with time. Hence, nodes can always determine the identification of their cluster, as well as identify other nodes in the same cluster, even in dynamic scenarios, where they move around and change from cell to cell[1]. A crucial aspect of this division is that there must be a mapping between the identification of a cluster and its physical location, to enable the use of a geographical routing scheme. We adopt the GPSR algorithm because it is simple and can achieve good routing performance in sparse networks. Interestingly, in Section 5.4, we show how to create a non-planar graph

[1]Usually, we use the term "cell" to mean the geographical square and the term "cluster" to mean a group of nodes. However, in some contexts, both concepts apply and therefore, we use the terms interchangeably.

for CHR, where GPSR is still able to converge. This simplifies the preprocessing algorithm and allows GPSR to perform better.

The node that stores a given pair (keyA, valueA) of the DHT depends deterministically on the result of applying the hash function on keyA. One of the fundamental aspects of our architecture is that we use clusters instead of nodes to hold the keys. This means that the relevant network entity is the cluster. Compared with a network of nodes, the simplicity of the network of clusters brings benefits to both, the routing scheme and the DHT operation. For this idea to work, the space of outputs of our hash function is the address space of the clusters. Hence, operation of the DHT becomes simple, when the key hashes to a cluster that really exists, i.e., that is populated by at least one node. Unfortunately, it is impossible to prevent that, in some cases, the key hashes to an empty cluster. In this case, CHR resorts to a variation of GHT. We detail this, as well as other aspects of our architecture, in the following sections.

5.3 Division into cells

The size of the cells is limited by the communication range of the nodes, because we require that a node in a cell can always listen to any other node either in its own cell or in any adjacent cell. This restriction ensures that in most circumstances, the clustered network stays connected, as long as the initial network is also connected. If we assume that nodes have a communication range of R, the resulting square side is at most $R/\sqrt{8}$. This can be seen in Figure 5.1. By adjacent cell, we are always referring to one of the 8 cells that surround the cell of a node. Note that we do not strictly require a Unit Disk Graph (UDG) model. We only require that nodes can

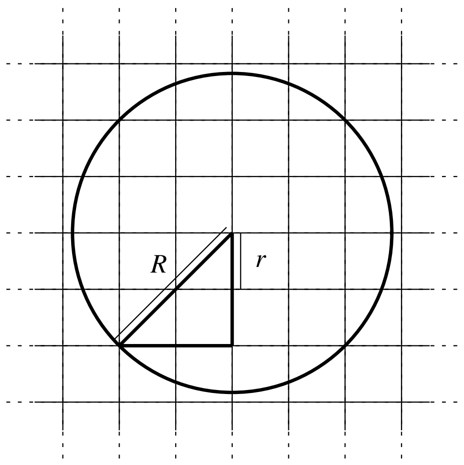
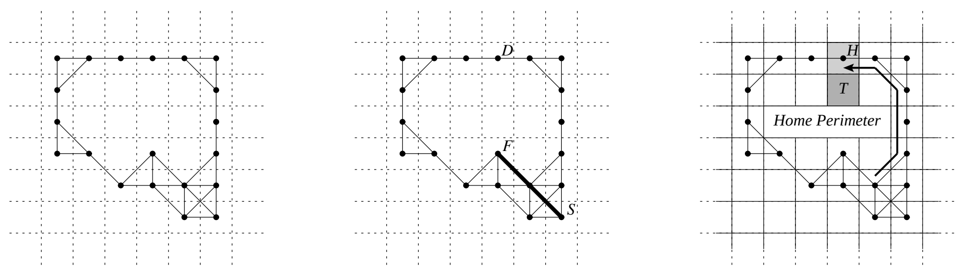


Figure 5.1: Fixed size of the cells

communicate with all the other nodes in their own cell and in the adjacent cells. We leave the issue of extending CHR to more generic models as future work, as we briefly discuss in Section 5.6.

Nodes need to be aware of other nodes in the neighboring cells, mainly for two reasons: *i*) the routing scheme requires nodes to know whether or not the adjacent cells are populated and *ii*) we do not require nodes to perform some kind of leader election algorithm; on the contrary by making nodes know all (or at least part of) their neighbors, we can use randomization to share the routing load among all the nodes. For instance, when a node is routing a message that needs to go through some neighboring cell, it can arbitrarily select any node of that cell as the next hop. Otherwise, the nodes of a given cell would still need to decide which of them would forward the message. More formally, we state in Assumptions 5.1 and 5.2, the conditions that we need to build our architecture.



(a) Definition of the graph of clusters (b) Routing may fail (c) Home cell and home perimeter

Figure 5.2: Network of cells

Assumption 5.1 *All the nodes know and can reach all the other nodes in their own and in any of the adjacent cells.*

Assumption 5.2 *A broadcast message inside a cell is received by all nodes of that cell.*

The purpose of Assumption 5.1 is to ensure that a node has sufficient knowledge to route messages and to support the DHT in the clustered hierarchy. However, as we describe in Sections 5.4 and 5.5 this assumption can be relaxed, for the sake of scalability. For routing to work, a node only needs to know a single neighbor in each of the adjacent cells (nevertheless knowing more than one neighbor will increase robustness). The purpose of Assumption 5.2 is to ensure a simple means of communication among nodes of the same cell, to enable operation of the DHT. But again, partial knowledge of the own cell will be enough in most circumstances. For instance, in the UDG model, these assumptions are easy to ensure, because all nodes of the same and adjacent cells are within range of each other. They only need to adjust their beacons to the density of the network, to reduce collisions of packets to a minimum.

Finally, Definition 5.1 defines a graph comprised of our cell-based clusters. This is represented in Figure 5.2a).

Definition 5.1 *Consider the graph where nodes represent clusters and edges exist between adjacent clusters if and only if their corresponding cells are both populated. To the embedding of the graph where nodes are placed in the center of the cells they represent, we call Geographically Clustered Graph (represented as $\mathcal{G}$).*

5.4 Routing scheme

CHR uses a routing scheme based on GPSR combined with a pre-processing algorithm that creates $\mathcal{G}$. This combination allows to address scenarios such as the one depicted in Figure 5.2b), where empty cells create voids in the cluster network. In this case, node F could be a local minimum in the path from S to D. It is intuitive and there is plenty of evidence (see, for instance, Kuhn *et al.*, 2003, or Chapter 4) that *i*) routing performance of GPSR in planar graphs is better if node density (vs. communication range) is sparse (because longer edges imply fewer hops) and furthermore, *ii*) for a given density of nodes, denser graphs, i.e., with more edges, also allow better performance. The reader should notice that there is no contradiction between *i* and *ii*: *i* states that network should have few nodes, while *ii* states that network should have many edges. Condition *i* is already fulfilled by our clustering approach. Condition *ii* is met because we do not need to planarize our graph, i.e., we do not remove intersections. The rationale for this is that the only intersections that may occur in $\mathcal{G}$ are in the diagonals (e.g., in the right lower corner of Figure 5.2a)). However, as we prove in Theorem 5.1, GPSR always converges in $\mathcal{G}$, despite the existence of these intersections. As a consequence, the pre-processing algorithm of

CHR only requires nodes to beacon the number of their cells (to fulfill Assumption 5.1). Besides this, nodes do not need further communication to define their local view of $\mathcal{G}$.

Theorem 5.1 *GPSR converges in* $\mathcal{G}$.

Proof 5.1 *GPSR converges in planar graphs. The only intersections that exist in* $\mathcal{G}$ *are in the diagonals of* 4 *nodes defining a square, say* H, I, J *and* K, *with edges* HI, IJ, JK *and* KH, *where* HJ *and* KI *intersect. Consider that the packet targeted to* D *is already in perimeter mode and that it entered perimeter mode at node* X. *Additionally, without loss of generality, consider that* XD *intersects the square* $HIJK$ *at edge* HK *(we are not concerned with the remaining intersections with the square) and that the packet reached* H. *Consider that this intersection does not take place at* H *or* K. *Given the very predictable position of the nodes of* $\mathcal{G}$, *the angle* $\angle DHX$ *measured at* H *is greater than* $\pi/2$. *As a consequence,* $|HD| < |XD|$, *the packet will reenter greedy mode and GPSR will converge. This reasoning can be easily extended to the case where intersection occurs at* H *or* K. *If* XD *does not intersect* $HIJK$, *then the packet will never enter a face inside the square, while in perimeter mode. Finally, consider that the packet at forwarding node* H *is in greedy mode.* H *would only send the packet in perimeter mode in a face inside the square for a destination in the direction of the* $\pi/2$ *angle* $\angle IHK$ *defined at* H. *But this is a contradiction, because, in this case* H *would not use perimeter, but greedy mode. Hence, in this case, GPSR also converges, because it never uses intersecting edges while in perimeter mode.*

5.5 DHT implementation

5.5.1 Basic Mechanism

In the most basic setting, the hash function determines the single cluster that will hold the (key, value) pair. In the case of a given pair (keyA, valueA), the cluster whose identification equals hash(keyA) will be responsible for storing valueA. For instance, consider the ("Bob", 32) pair, where the key "Bob" hashes to 144. In this case, the value 32 should be stored at the cell 144. Therefore, if we could ensure that at least one node is kept active in each cell, implementing the DHT would be straightforward. However, some cells may be empty and therefore, we need some mechanism that can also deal with this case.

5.5.2 Addressing of the Cells

Although the routing scheme does not require cells to have specific addresses (position of the destination node would be enough), the DHT requires that cells have globally-known logical addresses. The restriction here is that the space of outputs of the hash function must have a direct correspondence with the address space of the cells. To do this, we can follow a CAN-like approach, where the hash function outputs two-dimensional positions in space. Then, we would have to know the limits of each cell to determine the cell responsible for a given key.

However, in our implementation, we used the following one-dimensional addressing [2]. This assumes a bounded geographical space whose bounds are known by all the nodes. This scheme is equivalent to addressing the elements of a matrix in row-major order. Equation 5.1 shows how

[2]Note that which hash function we use does not have any impact on the routing performance.

to determine the address of a cell in this scheme. D_x and D_y are the size of the space in the two dimensions, d_x and d_y are the sizes of each cell and L_x and L_y are the coordinates of the center point of the cell (it can also be any other point inside the cell). This equation is useful to let a node determine the number of its own cell.

$$A = \lceil D_x/d_x \rceil \times \lfloor L_y/d_y \rfloor + \lfloor L_x/d_x \rfloor \tag{5.1}$$

The reverse correspondence allows nodes to perform geographical routing in $\mathcal{G}$. Equations 5.2 and 5.3 determine the center point (L_x, L_y) of the cell. c represents the number of columns and is computed as $c = \lfloor L_x/d_x \rfloor$, while % is the remainder of the division. To route to a given cell A, nodes need to determine the center point (L_x, L_y) of the destination cell, before they apply the GPSR routing algorithm. We need these equations, because geographical routing takes place using the center points of the cells (graph $\mathcal{G}$), while the DHT addresses of the cells are only logical. Consider again the ("Bob", 32) pair, where the key hashes to 144. To compute the center (L_x, L_y) of the target cell, a given node would have to replace A by 144 in the Equations 5.2 and 5.3.

$$L_y = d_y \left(\lfloor A/c \rfloor + 0.5 \right) \tag{5.2}$$

$$L_x = d_x \left(A\%c + 0.5 \right) \tag{5.3}$$

5.5.3 Division of the Keys in a Cell

The best way of dividing the keys among the nodes inside each cell may depend on the global number of keys to store and on the number of nodes inside a given cell. If the total number of keys to store is fairly small, the best policy may be to store all the keys in all the nodes of the cell. This is simple and tolerant to individual node failures. We believe that, despite simple, this scenario may have wide application. Consider that the average number of nodes per cell is n_c and that each node stores an average of s_n bits in the DHT. If the distribution of the nodes and the keys by the cells is even, the total size of items of the DHT to store in each node is approximately $s_n \times n_c$. This number is reasonable, for moderate node density and if memory of nodes is not too small. It is easy to derive alternative schemes, where the load is balanced among all nodes of a cell. Since communication inside a cell is easy and relatively inexpensive, nodes and items can use a second logical address, internal to the cell, to divide the load. Alternatively, the cluster can be further subdivided into groups such that each group of nodes takes a given group of keys. This second scheme may not require full knowledge of the neighbors that populate a given node's cell. This may be important in densely populated cells, if we want to relax Assumption 5.1.

5.5.4 Resolving Empty Cells

One of the difficulties with our DHT architecture is that it is impossible to ensure that there are no empty cells. The problem with empty cells is that some keys may be left without nodes to store them. Since we use GPSR to route messages, we can follow an approach similar to GHT (Ratnasamy *et al.*, 2002) to tackle this problem. Similarly, we define the concepts of *home cell* and *home perimeter*. Home cell is either the destination cell of a packet, if destination cell is populated, or the cell closest to the destination, in the other case (this requires a tie breaking rule, because many cells may be at the same distance). The home perimeter is the set of edges defining a face that encloses an empty destination cell (more precisely, the destination may be inside a face or outside the exterior face). These concepts are depicted in Figure 5.2c), where T represents the empty destination cell and H the home cell.

Like in GHT, CHR can take advantage of the standard behavior of the GPSR routing algorithm to ensure that a packet always reaches the home cell. This is easy to do if the home cell is the intended destination cell. If, on the contrary, the destination cell is empty, GPSR will also route the packet to the home cell. In the first time the packet reaches the home cell, any node in this cell will recognize that *i*) this cell is not the destination cell and *ii*) there is no edge connecting to the destination cell. Furthermore, the home cell is a local minimum in the path to destination. This forces the packet to enter perimeter mode (it might be circulating a perimeter already). Standard behavior of GPSR forces the packet to loop in the home perimeter if destination does not exist. However, GPSR drops the packet as soon as it discovers a cycle in the path. Like in GHT, we need to change this behavior to ensure that the packet is not discarded at the end of the cycle. At this point, the nodes in the home cell know that the destination cell is empty and assume that their cell will be the destination instead (refer to Figure 5.2c)). In fact, situation in our architecture is even simpler in some cases, because the home cell may already know that the destination cell is empty and that itself is the home cell due to a tie breaking rule. Hence, it can avoid the loop around it (this may happen if the empty destination cell is adjacent).

5.6 Implementation Issues

To implement CHR, there are a number of issues that we need to tackle. For instance, cells do not remain immutable along the lifetime of the network. A cell that is loosing population must get ready to release its keys. Additionally its also convenient to ensure that keys are replicated elsewhere in case the cell becomes unreachable. To overcome these problems, we propose the following solutions:

Dynamic Cell Structure when the number of nodes in a cell drops below l, the cell is considered empty (unless neighboring cells also have few nodes). On the contrary, the cell needs to acquire h nodes before it is considered populated ($h > l$). Note that the value h should be fairly small. As a consequence, knowing h does not require much memory, because, in general, a node will not need to know all the neighbors in its own cell (i.e., Assumption 5.1 can be relaxed). A cell leaving the network delivers its keys to its home cell. An entering cell needs to query its home perimeter to receive its keys. Additionally, it will also receive keys of empty cells for which it becomes the home cell;

Fault-tolerance requirements one of the occurrences that CHR should try to prevent as much as possible is the loss of stored (key,value) pairs. We already suggested the use of thresholds to ensure that keys are sufficiently spread among nodes of the same cell. This mechanism can be complemented with a technique already used in wired DHTs (e.g., Rowstron & Druschel, 2001), that consists of using k hash keys to replicate contents in different cells.

There are also a couple of additional issues to solve that we leave as open problems for future work. We believe that the most interesting of all these issues is the possibility of using CHR to support non-*UDG* models. Closely related to this is the problem of tolerating wrong determination of position by the nodes. Finally, in scenarios where utilization of CHR is more unfavorable, due to a low density of nodes, clustering these nodes may create a partition in an otherwise connected network. This can occur if nodes that are two cells away, and separated by empty adjacent cells, can see each other but are not allowed to create a link. We call this "cluster-induced disconnection". As we point out in Chapter 7, we leave these issues for future work.

5.7 Evaluation

To compare CHR with GHT, we tested the average path lengths in store/lookup operations. To do this, we routed messages from arbitrary existing nodes to arbitrary points in space. Hence, in general, these points did not correspond to any node and both, CHR and GHT had to route to the home cell/home node. We used a square of size 300×300 and a communication range of approximately 106 to have an 8×8 grid. Distribution of nodes in the square was uniform in all our experiments. Since node density is a key aspect to performance, we varied the number of nodes between 80 and 600.

The first thing we evaluate is the probability of having empty cells in CHR for each one of these node densities. This is depicted in Figure 5.3. In our settings, this probability is quite high when node density is low and rapidly decreases, to become nearly 0 as node density approaches an average of 7 nodes per cell. To see the impact of node density on routing performance we evaluate the average path lengths for each node density.

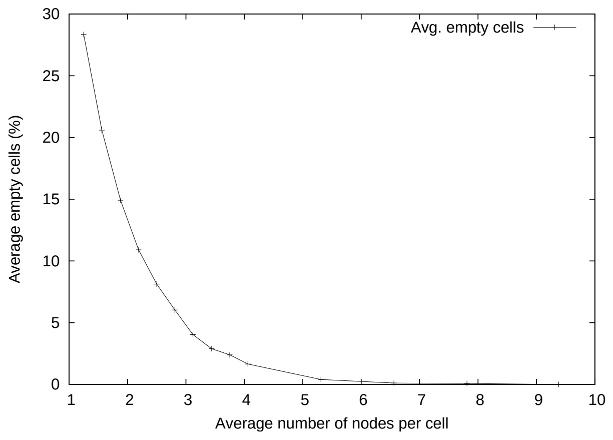


Figure 5.3: Average percentage of empty cells in CHR

In Figure 5.4 we show the average path lengths. The first observation that we can do from inspection of the figure is that, as expected, clustering really benefits CHR. Furthermore as node density increases, path lengths stabilize in CHR, while they grow in a non-clustered approach like GHT. This has to do with the length of the edges of the planar graph needed by GHT. As node density increases, the average edge length decreases and therefore, packets need more hops to reach destination. Finally, we can also observe that the impact of even a large number of empty cells is very small in the path lengths achieved by CHR.

Another advantage of CHR is that a moving node only needs to rebuild its database of keys, when it crosses a cell boundary. Finally, CHR is more robust, because a single key may be stored at many nodes, allowing the DHT to resist better to abrupt departure of nodes. In fact, this is a trade-off, where the down sides are that nodes have to know more neighbors (as required in Assumption 5.1) and need to store more keys (all the keys in their cell). However, as we stated before, both of these problems can be mitigated at the cost of decreasing the robustness of the DHT (because a

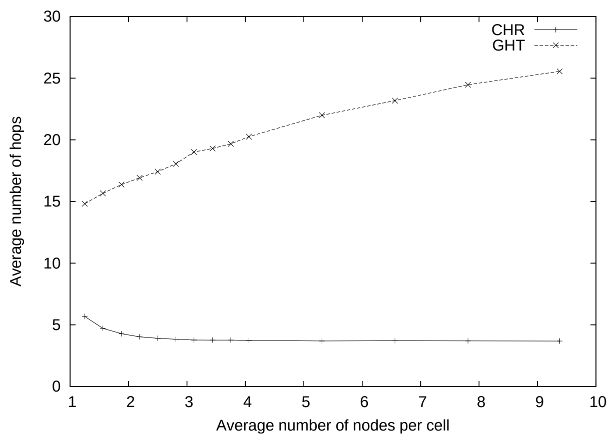


Figure 5.4: Performance of CHR vs. GHT

node can know fewer neighbors and store only a subset of keys in its cell). Nevertheless routing performance is not impaired by these techniques.

We believe that the simplicity of CHR and its favorable performance turn CHR into a valid approach to create a wireless DHT.

5.8 Energy Conservation Issues

Clustering nodes inside equally-sized cells allows us to use simple but efficient mechanisms that reduce consumption of energy without compromising tolerance to faults. We can take advantage of the clustering mechanism that divides nodes according to their cell, to put most nodes to sleep[3]. Such idea was originally proposed in the Geographic Adaptive Fidelity (GAF) scheme (Xu *et al.*, 2001). Besides energy exhaustion, nodes may fail due to several other reasons, including material fatigue, environmental hazards or deliberate attacks. To ensure proper operation of the wireless *ad hoc* network, sleeping nodes should monitor active nodes of their cell frequently.

[3]In the following, we always assume that only one node is awake in each cell.

If crashed nodes are not replaced, messages follow sub-optimal routes, DHTs may loose keys and, furthermore, the network may eventually become partitioned. On the other hand, energy consumption of idle and transmitting or receiving nodes is very high and therefore, nodes should remain sleeping as much as possible. In fact, if the energy consumed with the monitoring process is too high, non-active nodes may exhaust their batteries (and the batteries of active nodes) before they are needed. In this section, we overview the optimal monitoring period in fault-tolerant wireless *ad hoc* networks to ensure that: *i)* the network remains connected (i.e., crashed nodes are detected and substituted fast enough to avoid the network partition) and, *ii)* the lifetime of the network is maximized (i.e., inactive nodes save as much battery as possible).

5.8.1 The $\mathcal{P}$ and $\mathcal{F}$ Metrics

Nodes that are sleeping must wake up periodically to check availability of the active node of their cells. We call "monitoring period" to this periodical verification. We present a more precise definition of this concept in Section 5.8.2. We would like to select a value for the monitoring period that maximizes the system availability. This task can be prohibitively complex due to the multiple combinations of factors that affect the system lifetime such as the initial energy available to nodes, power consumption, network topology, etc. To address this complexity, we propose two new metrics that capture the importance of the *relative* values of different system parameters. Our metrics are motivated by the insight that, in the context of assessing the network availability, time intervals — in particular the monitoring period — should be analyzed in a relative sense: a monitoring period of 1 second has a different impact on a network whose lifetime is just 10 seconds than on a network whose lifetime is 1000 seconds. In a similar manner, the magnitude of values like power needed to transmit or to receive should also be measured in a relative way. Using simulations, we show that our metrics are useful to reason about the impact of faults in the network lifetime.

We start by defining the notions of "lifetime" and "ideal lifetime". Lifetime is the time to first network partition (about this issue, see Blough & Santi, 2002). Ideal lifetime, LT_I, is the network lifetime in a scenario where *i*) there are no faults, *ii*) switching nodes on and off has no cost and *iii*) nodes in the cells are omnisciently replaced at once (if replacement is available). LT_I is determined by simulation. Using LT_I we propose the following two metrics to assess the network behavior:

- the "power on-off consumption factor", $\mathcal{P}$, measures the impact of the energy spent powering nodes on and off. We define it as the ratio between the energy needed for one power on-off operation versus remaining energy spent in 1 time unit. This is determined as $\mathcal{P} = POE/(TE_0/LT_I)$, where POE is the power on-off energy and TE_0 is the total energy available in the beginning of the network life (if we assume that all N nodes have the same energy, E_0, in the beginning, $TE_0 = N \times E_0$). This makes $\mathcal{P}$ a function of all remaining energies (mainly idle, transmission and reception) of the system *but not* of node failure rate;

- the "failure weight factor", $\mathcal{F}$, measures the impact of faults in the network. We define it as the lifetime of the ideal network, LT_I, relative to mean time between failures ($MTBF$), i.e., $\mathcal{F} = LT_I/MTBF$. This makes $\mathcal{F}$ a function of failures of nodes and of all energies *except* power on-off energy. Large $\mathcal{F}$ means many node failures (pos-

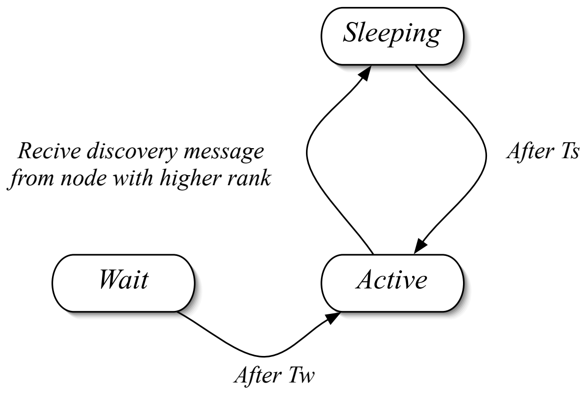


Figure 5.5: SQA algorithm

sibly due to a long network lifetime), while large $\mathcal{P}$ means a lot of energy needed to power a node on and off (at least compared with remaining energies, like idle and traffic energies).

5.8.2 The Monitoring Algorithm

To the algorithm that controls the monitoring process of nodes inside each cell we call "Sleep-Query-Active" algorithm (SQA). SQA is designed mainly to networks with low or no mobility, like sensor networks. The states of SQA are depicted in Figure 5.5. In steady state, nodes can only be in one of two states: either sleeping or active. Sleeping nodes periodically wake to monitor active nodes. Decision of which nodes go to sleep or stay active is based on the "rank" of nodes. The rank reflects the available energy of the node and it is larger for nodes with more energy. Active nodes only go to sleep when they listen about other nodes in the same cell with a higher rank. The purpose of the *wait* state is to desynchronize nodes that start at the same time. In our experiments, T_w was randomly set between 0 and 1 seconds with uniform probability.

Nodes running SQA synchronize with each other sending “discovery” messages in the following situations: *i*) when they enter active state, *ii*) periodically when they are in the active state (to overcome the loss of messages) and *iii*) in the active state when they receive a *discovery* message from a node with lower rank. Despite not providing any additional protection against node failures, nodes with larger supplies of energy (i.e., higher rank) will give an additional degree of protection against unexpected energy consumption caused by some peak of traffic. The single parameter to tune in SQA is the sleeping timeout, T_s, which we deem as the “monitoring period”. T_s is randomly chosen from an interval that is fixed beforehand. When we say that $T_s = c$, we really mean that T_s is selected from the interval $[0.5 \times c, 1.5 \times c]$. Then, each time a node goes to sleep, it picks the value for T_s from that interval with uniform probability. Our experimental evaluation shows that this choice is appropriate, because more often that not, the wireless *ad hoc* network will tend to behave in a very predictable way and fixing an optimal value for T_s will yield longer lifetimes than a dynamic approach like GAF. Selecting the most appropriate T_s is a challenging task that we resume in the next sections.

5.8.3 Determination of the Monitoring Period

Methodology of Analysis

Following a theoretical approach to determine T_s is a task of great difficulty (e.g., see Blough & Santi, 2002). Hence, we have opted to use simulations to evaluate the effect of different parameters on T_s. Unfortunately, without a correct methodology, the process of determining the effect of T_s using simulations is also a daunting task. As we have referred before, there are many factors that can influence network lifetime and consequently, T_s. Furthermore, these factors can be combined in multiple ways and often cannot be completely isolated in order to analyze their impact on network lifetime. Finally, but not the least, a single ns-2[4] simulation of a given configuration (i.e., for a single monitoring period), even when in executed on a Pentium IV 2.8 GHz with 2Gb of RAM, takes more than 100 seconds to complete.

To handle this complexity, we propose a methodology of analysis that allows to reason about the impact of these metrics before assessing the impact of network topology in the final system availability. Instead of always running simulation on a complete network, we first perform a careful study of the behavior of each network cell. Then, by estimating how many cells are required to maintain the connectivity of a given topology, we extrapolate the impact of the parameters in the entire network. We illustrate this methodology in Figure 5.6. The approach has both conceptual and practical advantages. From the conceptual point of view, it allows to separate the analysis of the influence of topology from other factors. From the practical point of view, cell level simulations *i*) allow to isolate factors that influence network lifetime and *ii*) run much faster. Therefore, cell simulation allows a much richer analysis of different combinations of factors in practical time. An additional advantage of the cell simulations is that its results can be used to assess other system properties, like the coverage of a sensor network in the presence of faults, for instance.

[4]"The ns Manual", http://www.isi.edu/nsnam/ns/ns-documentation.

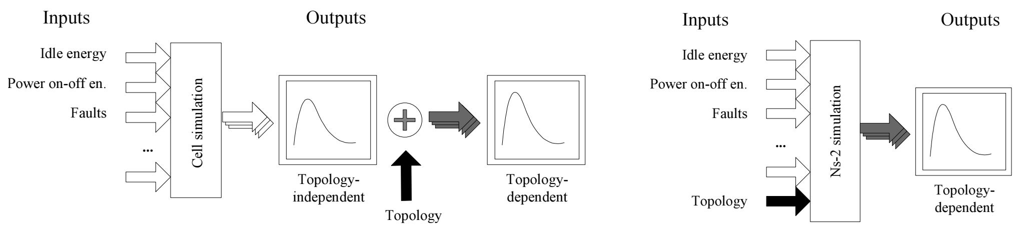


Figure 5.6: Cell Based Methodology *vs* Network Simulation

Table 5.1: Consumption of energy for the nodes tested

Node	Rx (W)	Tx (W)	Idle (W)	Sleep (W)	Initial Energy (J)
IEEE 802.11	0.974	1.341	0.843	0.066	15
MEDUSA-II	0.01248	0.01565	0.01234	0.00002	1
Rockwell's WINS	0.751	1.081	0.728	0.064	20

Simulation Settings

In our experiments we have used three different types of nodes: a node equipped with a Lucent IEEE 802.11 2 Mbps WaveLAN PC Card, a Rockwell WINS node and a MEDUSA-II node. Table 5.1 resumes the consumption of the three different nodes in the situations considered in our simulations. Figures for the first node were taken from the work of Feeney & Nilsson (2001), while values for the other two types of nodes were inferred from the work of Raghunathan *et al.* (2002). We assume that failures of nodes follow an exponential distribution. For simulation purposes, we have modeled this as a geometric distribution. After constant time intervals P, all nodes may fail with a given random probability p (we set $P = 0.5$ seconds in our simulator). Hence parameter r of the exponential distribution is $r \approxeq -\frac{1}{P}\ln(1-p)$, while $MTBF = 1/r$.

To plot a graphic that represents lifetime relative to LT_I against the monitoring period relative to LT_I (e.g., Figure 5.9), we select a number of monitoring periods, T_s, not exceeding the ideal lifetime. Then, we fix all the parameters, like power on-off consumption, idle power, initial energy, etc. and we experimentally analyze the lifetime achieved for each T_s. We used a square size of 800×800 meters with 256 nodes, which we divided into 8×8 cells (giving an average of 4 nodes per cell). Communication range was 250 meters. We performed simulations at the cell level and using the ns-2 simulator, using the settings that we describe next.

Cell Level Simulation Settings To determine the lifetime for a given monitoring period, we fix this monitoring period and use time as the independent variable. Then, as time goes by we assume a constant consumption of energy and observe whether the cell is awake or sleeping (it is awake if there is any node awake, otherwise it is sleeping). We used an average of 100 of these trials to approximate a continuous random variable, function of time t, that represents the probability that the cell is awake. An example of a random variable like this is depicted in Figure 5.7, for a specific value of T_s. To infer network behavior from this, we need to know the topology of the network. If disconnection occurs when an average number of D out of N cells are sleeping, we use a rough approximation and assume that when the awake probability of a cell drops below $(N-D)/N$, the network gets disconnected. Taking our grid for example, we used a simple simulation to derive the probability density function of the number of sleeping cells that cause network disconnection. This looks like a Gaussian curve centered at 40 and truncated at the 64 cells. Therefore, in such a topology, the threshold $(64-40)/64 = 0.375$ corresponds to a point where, more often that not, network will be disconnected[5]. Figure 5.8 shows the relative

[5]In this case, disconnection occurs when a significant proportion of the network is, in fact, unusable. We also observed this for other grid configurations.

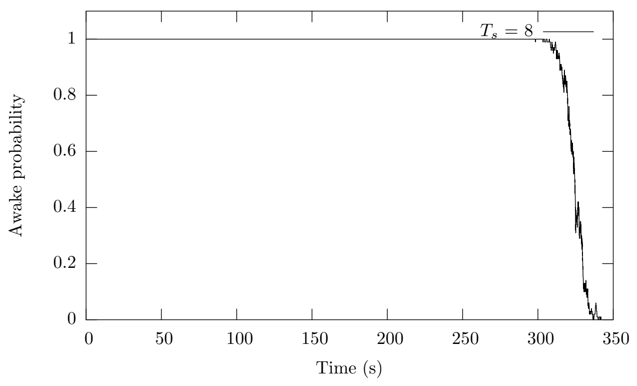


Figure 5.7: Probability of a cell being awake

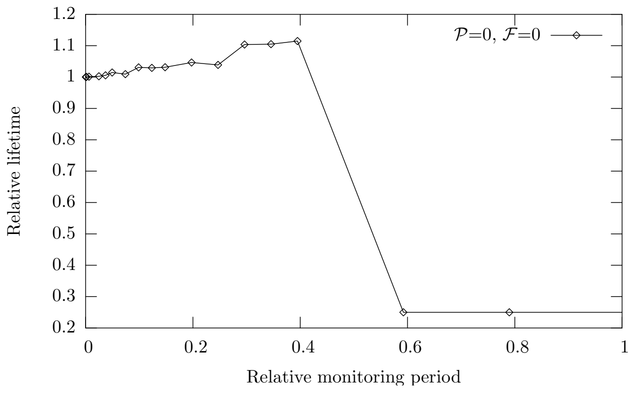


Figure 5.8: Relative network lifetime

lifetime graph as a function of the monitoring period for these settings. Lifetime and monitoring periods represented in this plot are relative to the ideal lifetime LT_I, to abstract away the absolute magnitudes that govern the network behavior.

Note that an entire data series needed to create a graphic like the one represented in Figure 5.7 produces a single point in Figure 5.8. In this case, this point should occur around $t = 327$ seconds (where the line $y = 0.375$ intersects the probability curve). In the cell simulations LT_I is estimated as the number of nodes of the cell $\times$ the time it takes to consume all the energy of a node. For the settings of these figures, this is around 324. Since $T_s = 8$ and $LT = 327$, this gives a relative monitoring period of $8/324 \approxeq 0.025$ and a relative lifetime of $327/324 \approxeq 1.009$. It is not really counterintuitive to have a lifetime greater than the ideal, due to the large idle power. In fact, this makes it advantageous to let some cells sleeping from time to time, to prolong their lives. On the contrary the ideal lifetime assumes that all the cells should be constantly awoken, which is not always the best strategy.

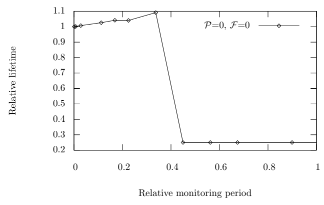
Network Level Simulation Settings We used the ns-2 simulator, version 2.27, to perform the network level simulations presented here. This required us to implement the SQA algorithm as well as port the GPRS routing algorithm to the same version of ns-2. We used a simulation environment similar to the one described by Xu *et al.* (2001). Nodes were divided in traffic and transit nodes. Traffic nodes serve as sources and sinks of traffic, while transit nodes are only used as intermediate hops for that traffic. Only transit nodes run the SQA algorithm with GPSR working as the underlying routing scheme. Traffic was generated by constant bit rate (CBR) traffic sources. In all our experiments we fixed the number of traffic nodes to 10. To prevent traffic nodes from stop generating traffic, their supply of energy was infinite.

Evaluation of Results

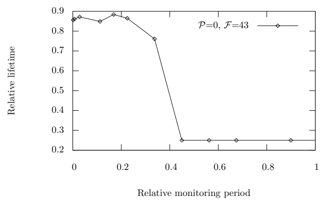
Figure 5.9 shows extreme as well as typical values for $\mathcal{P}$ and $\mathcal{F}$ for the cell level simulations. We can see that large values of $\mathcal{F}$ tend to require smaller monitoring periods to respond faster to faults (thus shrinking the curve at the right and making the peak start slightly earlier). On the other hand, larger values of $\mathcal{P}$ will penalize small monitoring periods, due to the cost of powering the nodes on and off (thus shrinking the curve at the left). Hence, as these two metrics grow, the curve tends to become thinner. Moreover, the growth of these metrics also makes the curve shorter as they impact network lifetime. By observing these and other simulations that we have done, we are able to conclude that very different operational conditions have similar behaviors, as long as the metrics $\mathcal{P}$ and $\mathcal{F}$ are similar.

In our simulations, longest lifetimes are almost always achieved when monitoring period is in the range of 10 to 20% of the ideal lifetime, for most values of $\mathcal{P}$ and $\mathcal{F}$. This stability has to do with the fact that a perfect monitoring algorithm should ensure that network has as few active nodes as possible (fewer than the number of cells, in practice), but preserving the minimum required to prevent disconnection from occurring. Hence, substitution of nodes depends on the death rate of nodes, which, on its turn, will determine lifetime. This explains why better strategies for (potentially) longer lifetimes, should use longer monitoring periods. Nevertheless, if this period goes over some threshold (30 to 50%), the relative lifetime sharply decreases, because nodes that die are not replaced and many cells become empty. This reveals a thin line between optimal and disastrous configuration.

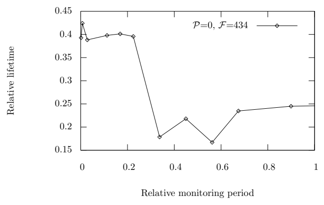
Table 5.2, which summarizes the results obtained, offers a qualitative analysis of $\mathcal{P}$ and $\mathcal{F}$. Outside the parenthesis we describe the system parameter that dominates network lifetime (other energies refers to idle and traffic energies), while inside we describe the shape of the peak that exists in the monitoring period (earlier, normal or later, respectively means that peak starts closer, in the normal place or farther away from the y-axis).



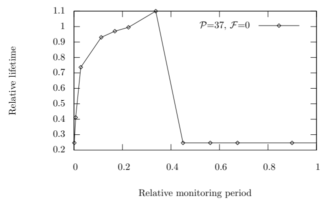
(a) Small $\mathcal{P}$/ Small $\mathcal{F}$



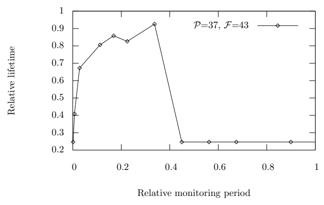
(b) Small $\mathcal{P}$/ Intermediate $\mathcal{F}$



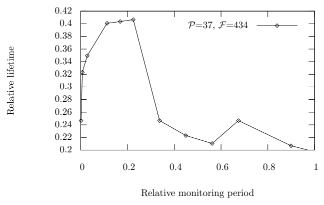
(c) Small $\mathcal{P}$/ Large $\mathcal{F}$



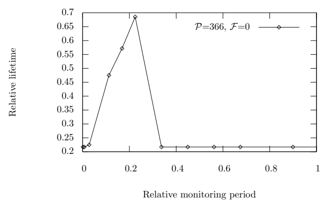
(d) Intermediate $\mathcal{P}$/ Small $\mathcal{F}$



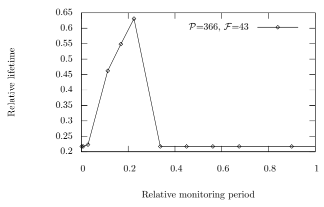
(e) Intermediate $\mathcal{P}$ and $\mathcal{F}$



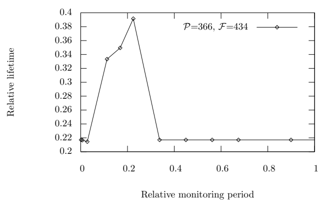
(f) Intermediate $\mathcal{P}$/ Large $\mathcal{F}$



(g) Large $\mathcal{P}$/ Small $\mathcal{F}$



(h) Large $\mathcal{P}$/ Intermediate $\mathcal{F}$



(i) Large $\mathcal{P}$/ Large $\mathcal{F}$

Figure 5.9: Impact of $\mathcal{P}$ and $\mathcal{F}$

Table 5.2: Dominating parameter (and peak shape) for variations of $\mathcal{P}$ and $\mathcal{F}$

	Small $\mathcal{F}$	Intermediate $\mathcal{F}$	Large $\mathcal{F}$
Small $\mathcal{P}$	Other en. (earlier)	Other en. & Failures (earlier)	Failures (earlier)
Intermediate $\mathcal{P}$	All en. (normal)	None (normal)	Failures (slightly earlier)
Large $\mathcal{P}$	On-off (later)	On-off (later)	Depends rel. magnitude (later)

With today's technology, awake nodes in idle mode spend a lot of power. Hence, we expect that they will operate in the first line of the table ("Small $\mathcal{P}$"). If with technological improvements idle energy decreases, $\mathcal{P}$ will depend mainly on data traffic generated in the network. In this case, the network will operate in a zone captured by the bottom line of the table ("Large $\mathcal{P}$"), whenever average traffic becomes low. In such scenarios, the appropriate choice of T_s will make an even more significant impact on the network lifetime.

5.8.4 Discussion

These experimental results demonstrate the appropriateness of using $\mathcal{P}$ and $\mathcal{F}$ to assess network behavior, by showing that, often, these metrics strongly determine network operation. Furthermore, results show that it is possible to achieve a lifetime close to the ideal (LT_I) by selecting the monitoring period adequately and according to $\mathcal{P}$ and $\mathcal{F}$. To conclude, we can say that the use of an algorithm like GAF (Xu *et al.*, 2001) or some variation of it, together with the experimental analysis of the $\mathcal{P}$ and $\mathcal{F}$ metrics can considerably extend the lifetime of a wireless DHT like CHR.

5.9 Summary

This chapter takes our previous work on wireless DHTs one step further, for densely populated networks. CHR divides the space into equally-sized cells to create a clustered architecture. This clustered architecture defines a virtual graph ($\mathcal{G}$) that works with a modified version of GPSR even without making the graph $\mathcal{G}$ planar. Our experimental results show that CHR achieves much better results than standard GHT, specially for higher node densities. Furthermore, since battery exhaustion critically determines the lifetime of an *ad hoc* network, we have also focused on techniques to conserve energy. To this matter we have considered a networks with clusters similar to CHR and we have created two metrics, "power on-off consumption factor", $\mathcal{P}$, and "failure weight factor", $\mathcal{F}$, that can help us to create wireless *ad hoc* networks more resistant to failures.

Notes

CHR is a joint work of the University of Lisbon and the University of Ulm in Germany supported by the European Science Foundation program MiNEMA (Middleware for Network Eccentric and Mobile Applications). Besides the author and Professor Luís Rodrigues, from the University of Lisbon, this collaboration involved Professor Jörg Kaiser, Changling Liu, and Carlos Mitidieri. The section on energy conservation is based on work developed by the author and by Professor Luís Rodrigues. Parts of the work of this chapter have been published as follows:

- Filipe Araújo, Luís Rodrigues, Jörg Kaiser, Changling Liu, and Carlos Mitidieri. CHR: a distributed hash table for wireless ad hoc networks. In *The 25th IEEE International Conference on Distributed Computing Systems Workshops (DEBS '05)*, Columbus, Ohio, USA, June 2005.
- Filipe Araújo and Luís Rodrigues. On the monitoring period for fault-tolerant sensor networks. In *Second Latin-American Symposium on Dependable Computing (LADC '05)*, October 2005 (to appear).

6
Position-Based DHTs for Wired Overlay Networks

In this chapter, we present a position-based distributed hash table (DHT), called “GeoPeer”. We can look at GeoPeer as the wired counterpart of the wireless DHT created by the Fast Localized Delaunay Triangulation (FLDT), presented in Chapter 4, because GeoPeer is also based on a Delaunay triangulation. Unlike FLDT, in GeoPeer we can use non-localized algorithms to create a complete Delaunay triangulation. Unfortunately, such a simple Delaunay triangulation suffers from long path lengths, which grow to $O(\sqrt{n})$, where n is the number of nodes. While this figure would not be a problem in a wireless network, there are many (non-position-based) overlay networks (Chapter 3) that can achieve much better than this. Therefore, we created a complementary mechanism, called “Hop Level”, that augments the Delaunay triangulation and allows the complete overlay network to achieve a nearly optimal path length/node degree trade-off. Hence, in this chapter, we present the two halves that make GeoPeer. We start in Section 6.1 with the GeoPeer architecture and in Section 6.2 we present the Hop Level mechanism. Although we can use this mechanism to complement GeoPeer, it is independent of the specific underlying network which does not need to be a a Delaunay triangulation.

6.1 GeoPeer

In this section we present a position-based DHT called "GeoPeer". This DHT is based on a Delaunay triangulation that con be augmented with the long range contacts (LRCs) that we will present in Section 6.2. Unlike the DHT of Chapter 4, GeoPeer can take advantage of an underlying network with routing capabilities, like IP, to create a complete Delaunay triangulation.

We can use GeoPeer as an ordinary DHT enhanced with positional information to support position-aware services on top of IP. Other peer-to-peer systems that we presented in Chapter 3, such as Pastry (Rowstron & Druschel, 2001), Tapestry (Zhao *et al.*, 2001), Chord (Stoica *et al.*, 2001), D2B (Fraigniaud & Gauron, 2003a), Koorde (Kaashoek & Karger, 2003) or Viceroy (Malkhi *et al.*, 2002), do not own the characteristics required to support position-aware services. Systems such as CAN (Ratnasamy *et al.*, 2001), TOPLUS (Garcés-Erice *et al.*, 2003), eCAN (Xu & Zhang, 2002), and the Delaunay triangulation proposed by Liebeherr *et al.* (2001), are closer in spirit to GeoPeer but they also lack features which are essential to support position-aware services efficiently.

Unlike most other systems, the use of geographical position is inherent to GeoPeer. Therefore, GeoPeer owns a number of interesting properties: it is capable of providing position-awareness in fundamental operations performed by applications, such as reads, writes or queries. In this section we focus on the scalability of the network of stationary nodes that provides support to very large-scale position-aware services (possibly, in cooperation with mobile nodes and wireless sensors). To the best of our knowledge, the scalability, decentralization, and dynamic aspects of the stationary infrastructure supporting position-aware computing have been overlooked in the literature.

6.1.1 Overview of GeoPeer

In GeoPeer, the identification of nodes corresponds to their geographical location. Using their identification, nodes self-organize into a planar Delaunay triangulation augmented with carefully selected LRCs to significantly reduce path lengths. A graph based on a Delaunay triangulation has the following desirable characteristics:

1. expected $O(1)$ node degree;
2. good nearby routing performance; and
3. simple distributed construction.

The combination of these features results in a peer-to-peer system with the following unique advantages:

- by creating a mesh of nodes identified by their physical location, support for applications that execute position-aware operations, such as queries or broadcasts, can be provided by very simple mechanisms;
- when compared with a two-dimensional CAN-like network, the node degree in a Delaunay triangulation should be greater, but still $O(1)$ in expectation (near 6 instead of 4 in perfectly balanced cases) and, therefore, nearby routing should be improved;
- due to the LRCs that augment the Delaunay triangulation, GeoPeer has logarithmic path lengths for the network sizes we tested (Section 6.2.8).

In the GeoPeer DHT, keys correspond to positions in space. Hashing some value representing an object yields the GeoPeer identification of that object. Since identification and position are equivalent, the hash function returns a pseudo-arbitrary position in space. Therefore, positional information attributes may be used to carefully position resources in some application dependent way, e.g., by enforcing the use of a hash function that returns some node inside a restricted zone, near the clients of a service.

6.1.2 Main Components

In the following, the main components of the GeoPeer architecture are described in detail. These components are:

1. an algorithm that creates and maintains the Delaunay triangulations;
2. an algorithm that ensures that any possible key is held by exactly one existing node;
3. an algorithm that performs routing of messages in the overlay network; and
4. a mechanism to establish LRCs.

We describe the first three mechanimsms, while the LRC mechanism is postponed to Section 6.2.

6.1.3 Creation and Maintenance of Delaunay Triangulations

To create and maintain the Delaunay triangulation, GeoPeer uses a scheme similar to the one of Liebeherr *et al.* (2001) (however, unlike GeoPeer, Liebeherr *et al.* do not use LRCs). Note that many constructions proposed for wireless *ad hoc* networks, such as the ones of Li *et al.* (2002); Gao *et al.* (2001), are not applicable in this context, because these algorithms assume static settings for triangulation and, more importantly, they assume that nodes are provided with broadcast-capable radios.

Messages

To create and maintain the Delaunay triangulations, nodes periodically exchange messages with their neighbors. The five message types exchanged by the algorithm are:

- the BEACON message, used by a node to inform its neighbors that it is still actively participating in the overlay network;
- the JOIN message, used to add new nodes to the network;
- the FAILURE message, used to disseminate information about the failure or departure of a node;
- the TRIANGULATE message, used by a node to propose the setup of a Delaunay triangle with its neighbors;
- the BREAKLINKS message, used to reconfigure the network in response to new joins and leaves.

The purpose and function of each of these message types will be detailed in the following.

Steps

The algorithm is decentralized, as it does not rely on any single point of control. It consists of three logical steps:

1. the *neighbor discovery* step. Node N initiates this step to enter the network. To join, node N must use some out-of-band mean to discover one node already participating in the network, say P. P will then forward a special JOIN message on behalf of N destined to N. Since N does not yet belong to the network, the JOIN message will be received by some node X. This node X will forward the JOIN message to all the Delaunay neighbors of N that X knows about (to inform them of the existence of N) and will also reply with another JOIN message to N with the list of those Delaunay neighbors (note that this step does not establish the triangulations);

2. the *neighbor maintenance* step. In this step, nodes that belong to the same triangle periodically exchange BEACON messages to inform their neighbors that they are alive and actively participating in the network;

3. the Delaunay *triangulation* step. This is naturally, the most complex step of the algorithm.

Based on the information collected in the previous steps, each node P computes a Delaunay triangulation using its own local knowledge. As a result, P may find out that there should exist a Delaunay triangle $\triangle PN_1N_2$, between P, N_1 and N_2. In this case, for convenience of exposition, we say that the predicate $Delaunay\triangle_P(N_1,N_2)$ is true at P.

When $Delaunay\triangle_P(N_1,N_2)$ holds at P, P sends a TRIANGULATE $\triangle PN_1N_2$ message to both N_1 and N_2. When P receives a TRIANGULATE $\triangle PN_1N_2$ from N_1, if $Delaunay\triangle_P(N_1,N_2)$ holds then P replies to N_1 with another TRIANGULATE message, otherwise, P replies with a BREAKLINKS message including all nodes that it believes should triangulate with N_1.

Therefore, if all neighbors agree on the triangulation, they will exchange a consistent set of TRIANGULATE messages and the corresponding Delaunay triangles are set-up. Otherwise, they update their local information using the contents of the BREAKLINKS message and re-execute the local computation. Note that if there is some node inside $\bigcirc PN_1N_2$, the predicate $Delaunay\triangle_P(N_1,N_2)$ is immediately switched to false. A very simple way of checking this condition was presented by Sibson (1977). Again, as we did in Chapter 4, we assume that no four nodes are co-circular (co-circularities can be easily addressed by slightly perturbing the position of involved nodes).

Dynamic Aspects of the Algorithm

As noted before, to cope with a dynamic topology, the algorithm must take into account the following aspects:

1. the failure of nodes;
2. the emergence of new nodes and, as a consequence, the possibility of nodes having a different view of the network topology.

Node failures and departures are detected through the absence of BEACON messages from that node (to simplify, we do not distinguish these two events, however departures allow a more gracious way to redistribute the keys). When some neighbor of F detects that node F failed, it recomputes the Delaunay triangulation and sends a FAILURE message to all its Delaunay neighbors. All nodes that are neighbors of F should resend the FAILURE messages of F. This ensures that all Delaunay neighbors of F become aware of its failure. Since network is asynchronous, nodes must store information about the failure of F. Therefore, FAILURE and BREAKLINKS messages include a list of nodes that are known to be failed (possibly empty in the case of a BREAKLINKS message). If after a TRIANGULATE message from P, N replies with a BREAKLINKS message, with indication of some node F that P knows to be failed, P sends a FAILURE message and later retries the triangulation.

It is also possible for nodes to enter the graph at any instant. Assume that P becomes aware of the presence of some new node Q and, as a result of recalculating the Delaunay triangulation, some triangles, $Delaunay\triangle_P(N_1,N_2)$ commute from true to false. In such case P sends a BREAKLINKS message to the vertices of those triangles. If $Delaunay\triangle_P(Q,N_1)$ is true for some node N_1, P will again send TRIANGULATE messages as described before.

Optimizations

For clarity of exposition, we deferred discussion of the following issue: BREAKLINKS and FAILURE messages should not carry indefinitely information about all nodes that failed in the past, as this could become a considerable overhead. Therefore, N only resends information that F failed to some peer node A until A acknowledges. Furthermore, to avoid storing information of some failed node F forever, nodes discard information of F after the expiration of a timeout.

6.1.4 Division of Space

For each point in space there is one and only one responsible GeoPeer node. The node responsible for a point P inside triangle $\triangle ABC$ is always the node of $\triangle ABC$ closest to P. This definition accounts for the case depicted in Figure 6.1a), where the Voronoi cells may cross triangle borders. In "well behaved" triangles where the center of the circumcircle, O, lies inside the triangle, such as the one depicted in Figure 6.1b), division of the space is straightforward and is done according to the figure. Areas A_A,

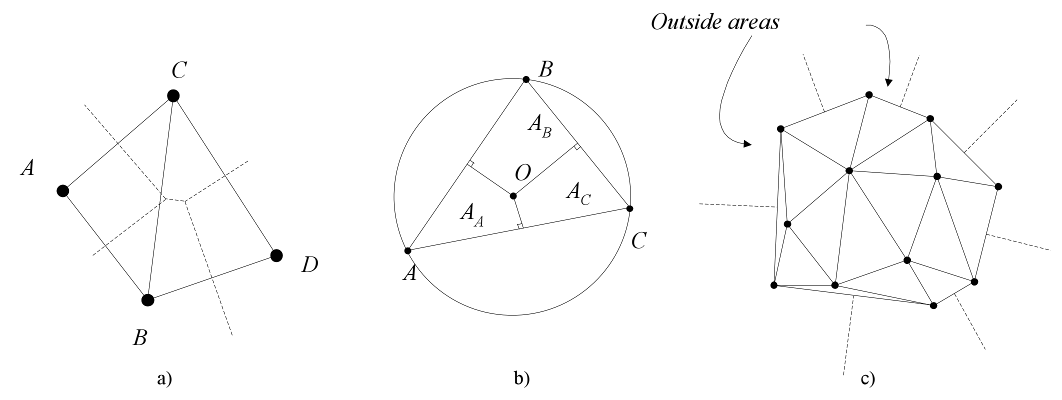


Figure 6.1: a) Voronoi cells (dashed lines) cross triangle boundaries, b) Circumcircle, c) Outside areas

A_B and A_C cover the entire triangle and define the set of points that are, respectively, closer to A, B and C. If the point O lies outside the triangle, some of the points of the Voronoi cell of some node A may end up in the region of responsibility of other node (say node B or C in Figure 6.1a)). In the borders of the plane, where no further triangulations are possible, proximity criterion is used to determine the areas of responsibility of the nodes, as depicted in Figure 6.1c).

6.1.5 Basic Routing

To route messages we use the greedy routing algorithm for the following reasons: *i*) it ensures convergence in a Delaunay triangulation (Bose & Morin, 1999), *ii*) it is efficient in most circumstances (Li *et al.*, 2002) (although Bose & Morin showed some cases where performance is arbitrarily bad, these examples should be pathological); and *iii*) greedy routing algorithm copes with LRCs without any modification. Still, nodes need to resource to a different algorithm when they find a situation like the one depicted in Figure 6.1a). In this case, message may progress from triangle to triangle until it reaches the node responsible for the destination point. In general, this will involve a very small number of hops. For instance, in the case depicted, *A* could immediately send the message to either *B* or *C* (whichever is closest to destination) and this node would be the final destination of the message.

6.1.6 Applications of GeoPeer

GeoPeer may, as any other decentralized peer-to-peer system, be used to support any sort of application that benefits from a scalable implementation of a DHT, such as, for instance, decentralized storage services (Douceur & Wattenhofer, 2001; Druschel & Rowstron, 2001; Rhea *et al.*, 2001). However, some of the characteristics of GeoPeer, like location-awareness and uneven distribution of nodes, make it specially fit for the support of location-aware services. We now illustrate the benefits of the architecture by giving some examples of context-aware services that can be trivially implemented on top of GeoPeer and that can benefit from the reduced diameter of a GeoPeer network.

Geographically-scoped multicast this service consists of disseminating a notification to all nodes located inside a given geographic region (e.g., an alarm about some natural disaster). The service can be easily implemented by routing a notification to the GeoPeer nodes responsible for the center of that area which will, in turn, initiate a scoped-broadcast of the notification, using the technique proposed by Liebeherr *et al.* (2001). It should be noted that, with the exception of a two-dimensional CAN (and variations like eCAN) no other peer-to-peer system referred before would directly support this service. Furthermore, the use of Delaunay triangulations makes GeoPeer more efficient than CAN or eCAN;

Geographically-scoped queries this service is used to collect information from nodes located inside a given geographic region (e.g., environmental or security monitoring of geographical areas by connection of the relevant sensors to the GeoPeer nodes). It can also be used to collect more mundane information, such as the location of cinemas or bars around a given location. The service works by having the node responsible for the center of the region of interest acting as an ambassador of the client. This node can query all nodes in a given diameter, collect all the replies, and send the consolidated information back to the client in a single message (this may involve computation of averages, selection of the lowest or highest values, etc.);

Other location-aware services GeoPeer also opens new less obvious possibilities for applications that need to determine location of critical resources, like a rendezvous point in a core-based multicast tree (Ballardie, 1997) or in publish-subscribe applications (Pietzuch & Bacon, 2002; Castro *et al.*, 2002). Exploration of this possibility is outside the scope of this thesis and we leave this as an open possibility for future work (see Chapter 7).

6.2 The Hop Level Mechanism

6.2.1 Overview of the Hop Level Mechanism

In this section, we present and evaluate a mechanism called "Hop Level" that creates and maintains LRCs in (wired) overlay networks. The Hop

Level mechanism owns the following characteristics:

i) support for unbalanced node distribution;

ii) support for multidimensional spaces;

iii) near-optimal path lenght/node degree trade-off; and

iv) lazy creation of the LRCs.

Although Hop Level can considerably reduce the path lengths of planar position-based overlay networks, it reaches farther than that. Hop Level is also well suited for overlay networks that support range data queries (as opposed to distributed hash tables that only support exact queries) with one or more dimensions.

It is a well-known fact that one of the major problems of DHTs is their lack of support for range data queries (Chawathe *et al.*, 2003). Unlike DHTs that only perform exact queries, Distributed Storage systems (Aspnes *et al.*, 2004; Bharambe *et al.*, 2004; Karger & Ruhl, 2004) (DSSs) allow efficient range queries. This makes the design of a DSS more complex, because we can no longer assume that data is uniformly distributed in space[1]. Additionally, we cannot assume that entrance and departure patterns of data items will also favor balancing. On the contrary, DHTs were based on the assumption that consistent hashing would result in a good balance of node identifications and data items.

Often, in overlay networks, including DHTs and DSSs, it is possible to distinguish between two different types of contacts: "nearby" contacts, forming a kind of connected lattice between nodes that have close virtual identifications, and "long range contacts" (LRCs) between nodes that have "distant" virtual node identifications. While the former type of contacts

[1]This assumption does also not hold in a position-based DHT.

may be important in certain overlays to ensure connectedness and routing convergence, short path lengths actually depend on the latter type of contacts. In fact, it is the capability to “jump” over many closer nodes in a single hop that makes it possible to achieve short path lengths. Therefore, the goal of the Hop Level mechanism is to create and maintains long range contacts in overlay networks, including position-based DHTs and DSSs. It is particularly well suited to these two types of networks, because it can cope with unbalanced distribution of nodes and it supports single as well as multidimensional range queries on the data. We believe that this is one of the most innovative aspects of Hop Level, because most overlay networks are tied to one-dimensional address spaces, where nodes must be numerically or alphabetically ordered (e.g., SkipNets of Harvey *et al.*, 2003).

An additional characteristic of Hop Level is the nearly optimal path length/node degree trade-off that it can achieve. Furthermore, unlike existing overlay networks that we are aware of, in the Hop Level mechanism, when a node enters the network, it postpones creation of the LRCs. Later, it progressively creates the LRCs as they are needed to route real messages. In fact, lazy creation of the LRCs is one of the most significant aspects of Hop Level, as this reduces control traffic with only a minor effect on routing performance. In this way, behavior of Hop Level under churn is very good.

6.2.2 Comparison of Hop Level with Previous Work

To achieve short path lengths, most DHTs assume a homogeneous distribution of nodes. Some, like Chord, might resist to a disadvantageous distribution, but at the cost of trying to populate the entire node identification space with LRCs. Unlike most other DHTs, LAND (Abraham *et al.*, 2004) copes with an unbalanced distribution of nodes, but it hashes identifiers of objects, thus making it unsuitable to support range queries. SkipNet (Harvey *et al.*, 2003) was also designed from scratch to cope with the unbalanced use of identification space. In fact, SkipNet is more appropriate to support a DSS, because it supports queries. However, the identification space of a SkipNet is one-dimensional and generalization to higher-dimensional spaces does not seem trivial. Unlike SkipNet, works proposed by Aspnes *et al.* (2004); Bharambe *et al.* (2004); Karger & Ruhl (2004) have explicit support for complex load balancing mechanisms without impairing efficient range queries. Of these, only Mercury (Bharambe *et al.*, 2004) supports multidimensional range queries. However, in Mercury this requires a different data structure (a ring of nodes) for each queriable attribute (including a copy of the data). When compared to these systems, support of multidimensional range queries is inherent to the Hop Level mechanism and does not need to be mapped to multiple one-dimensional queries.

Perhaps the works that are closer in spirit to Hop Level are those that try to add LRCs to a previously existing lattice to create a "small-world". This the case of the work of Kleinberg (2000), Barrièrere *et al.* (2001) and of Symphony (Manku *et al.*, 2003). A small-world is characterized by a constant node degree and poly-logarithmic diameter. The work of Kleinberg (2000) models the small-world phenomenon as a lattice of squares with $O(1)$ contacts, where nodes select their single LRC according to a random process based on the distance to their peers raised to the power $-r$. The interesting conclusion of this work is that power $r = 2$ represents the correct balance between the geographical information implicit in the LRCs and their ability to forward messages to long distances. Other overlay networks, with $O(\log n)$ contacts, implicitly follow a similar principle and keep a nearly constant number of LRCs for exponentially larger disjoint areas of the virtual space around the identification of each node. Perhaps the most evident case where there is a distinction between short and long range contacts is in the "expressways CAN" of Xu & Zhang (2002) (Section 3.1.4). Hop Level inherits the idea of distributing a nearly constant number of LRCs between disjoint groups of nodes of exponentially increasing sizes. Crucially, the fundamental difference we introduce, is the decoupling between the number of surrounding nodes and the size of the surrounding area, as these may not coincide. In the next section, we precisely state the problem that we are addressing.

6.2.3 Problem Statement

In this section, we will consider that routing convergence is ensured by nearby contacts already existing in the overlay network (e.g., as in Liebeherr *et al.*, 2001; Kleinberg, 2000). Although these are examples of two-dimensional networks (of which we tested a Delaunay triangulation like the one we presented in Section 6.1 or the one of Liebeherr *et al.*), there is however no restriction on the number of dimensions of the overlay network. A crucial point here is that the distribution of nodes does not need to follow any specific pattern.

Hence, we will consider the following conditions:

i) nodes are organized into a multidimensional underlying overlay network having only nearby contacts;

ii) identification of nodes is arbitrary (as a consequence distribution of

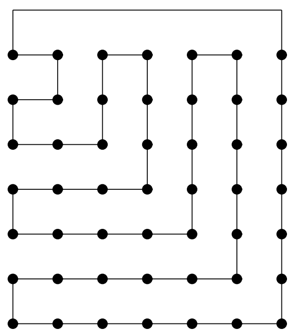


Figure 6.2: Mapping a 2-D space into a ring

nodes in space may be unbalanced).

The goal of condition *ii* is to maintain locality, by preventing arbitrary conversion of node addresses from one space of identifications to another, e.g., by a hash function. There are many practical examples where this restriction holds. In some DSSs (Aspnes *et al.*, 2004), the overlay structure directly reflects the contents of the data, which is organized in a sequential order. In this way, it is possible to make range queries efficiently. On the contrary, hashing data to obtain some balance in a different identification space would defeat this goal. Another example where condition *ii* holds occurs in systems where identification of a node bears some relation with its physical location, like in the GeoPeer DHT (see Section 6.1) or when using landmark ordering (Ratnasamy *et al.*, 2001). The reader should notice that, without condition *ii*, there is a trivial solution to the problem we are addressing, since any n-dimensional discrete space can be mapped into a ring, as depicted in Figure 6.2 for a two-dimensional case.

Furthermore, we will consider the use of a routing scheme where *i*) the pre-processing algorithm can only collect information of $O(1)$ nearby peers and $O(\log n)$ distant peers per node and *ii*) the routing algorithm will select, among the forwarding node's contacts (either short or long range), the one which is closest to destination in terms of Euclidean distances[2]. Given these conditions, our goal is to design a mechanism that creates and maintains a set of LRCs at each node such that routing convergence is guaranteed with $O(\log n)$ expected path lengths *despite* non-uniform node distribution. Moreover, the number of LRCs stored by each node ($O(\log n)$, where n is the number of nodes in the system) must not depend on the size of the virtual identification space, but only on the number of nodes effectively existing in the system. Balancing the workload among the peers in the DSS is an issue orthogonal to our work and it has been already tackled in previous work (e.g. Aspnes *et al.*, 2004).

6.2.4 Description of Hop Level

We now describe our proposal to create and maintain LRCs in unbalanced and sparse overlays. The goal of the *Hop Level* mechanism is to prevent messages from doing more than a predefined number of hops of the same length, say b hops. To achieve this goal, LRCs are established automatically whenever a message makes b consecutive hops. Consider, for instance, that some node F is forwarding to node N_1 a message m originated at node S and destined to node D. Assume that F realizes that this will be the b-th hop of the message. In this case, F triggers the creation of a LRC from S to N_1, denoted $S \xrightarrow{1} N_1$, by sending a control message to S. The process is repeated from N_1 onwards: if after b hops, message m reaches N_2, N_1 will create a LRC to N_2, $N_1 \xrightarrow{1} N_2$, and so on. Let us call these LRCs, "level-1

[2]There is no loss of generality in assuming Euclidean distances, as other metrics could also be used if more appropriate to the structure of the lattice, e.g., Manhattan distance or one-dimensional virtual identification distance.

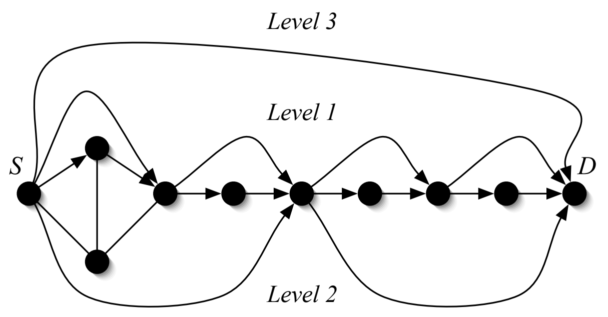


Figure 6.3: Hop Level LRCs

LRCs". If the message path is long enough, it may happen that a sequence of *b level-1 LRCs* occurs, for instance: $S \xrightarrow{1} N_1$, $N_1 \xrightarrow{1} N_2$, …, $N_{b-1} \xrightarrow{1} N_b$. In this case, a new LRC from S directly to N_b should be created. This new LRC, $S \xrightarrow{2} N_b$, is one level above of the previous ones. This mechanism should be applied recursively for all levels. Hence, a LRC of level-l jumps over b^l hops. Figure 6.3 illustrates our mechanism.

Note that only *b consecutive* hops of the same level should trigger a LRC of the next level. We illustrate this restriction with some concrete examples in a network where b is set to 2. Consider the following message path ($\rightarrow$ represents a hop where no LRC is used): $N_0 \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4 \rightarrow N_5 \rightarrow N_6 \rightarrow N_7 \rightarrow N_8$. In this case, the following LRCs would be created: $N_0 \xrightarrow{1} N_2$, $N_2 \xrightarrow{1} N_4$, $N_4 \xrightarrow{1} N_6$, $N_6 \xrightarrow{1} N_8$, $N_0 \xrightarrow{2} N_4$, $N_4 \xrightarrow{2} N_8$, and $N_0 \xrightarrow{3} N_8$, for a grand total of seven LRCs. Consider now the following message path: $N_0 \xrightarrow{3} N_8 \xrightarrow{1} N_{10} \xrightarrow{1} N_{12} \xrightarrow{2} N_{16}$. In this case, the following additional LRCs would be created: $N_8 \xrightarrow{2} N_{12}$, $N_8 \xrightarrow{3} N_{16}$, and $N_0 \xrightarrow{4} N_{16}$. On the other hand, a message path such as $N_0 \xrightarrow{1} N_2 \xrightarrow{4} N_{18} \xrightarrow{1} N_{20}$ would not trigger the creation of any additional LRC.

Although Hop Level does not need any *a priori* limit to the number of LRCs, such limit may be imposed to ensure different trade-offs between path lengths and node degrees. The limit should not impair the ability of Hop Level to adapt to network conditions without needing human assistance (self-configurability). Therefore, we do not impose any limit to the total number of LRCs per node or to the number of levels of each node, but only to the number of LRCs that exist in each level. As network size increases, the number of levels will also increase accordingly and thus will fix the number of LRCs existing in the node. The shape of this growth is evaluated in Section 6.2.8.

6.2.5 Algorithm

Description

Our implementation of the Hop Level mechanism requires a minimum of three variables per level l to be carried in each message m: the number of hops, $nh_m[l]$, the node that may receive a new LRC of that level, $s_m[l]$, and whether this node has space for an additional LRC, $a_m[l]$. Whenever level counter $nh_m[l-1]$ reaches the limit b, a new LRC, starting at $s_m[l]$ should be created.

When a forwarding node uses a LRC of level-l to send a message, it must check the LRC used by the previous hop node, say level-p. If $l > p$, neither one of the LRCs that preceded this hop can be used to create new LRCs (e.g., if a level-3 LRC is being taken after a previous level-2 LRC). Now, consider that message m is going to be sent along its b-th consecutive hop of level-l to node N. In this case, forwarding node F sends a control message to the node that initiated the sequence of level-l, prompting it to create a LRC of level-$(l+1)$ to node N. Then, node F sets the number of hops of level-l to 0 and increments the number of hops of level-$(l+1)$ by 1.

Should this substituting hop become the b-th hop of level-$(l+1)$, the same process is repeated for level-$(l+1)$, and so on, until a level with fewer than b hops is reached.

To implement this algorithm, messages must carry the level p_m of the LRC used by the previous hop to reach F, and an indication of the highest level of the array that contains valid information, max_m. Each node F, when forwarding the message m to N, executes Algorithm 6.1. The variable l captures the level of the LRC used by F to reach the next hop N. $a_F[k]$ is a boolean variable that indicates whether F has slots available at level k to store additional LRCs. If F is the source of the message, $F = S$, it is necessary to set previous level $p_m \leftarrow \perp$. In this case, the execution of the algorithm will initialize $max_m \leftarrow l+1$, $s_m[max_m] \leftarrow S$, $a_m[max_m] \leftarrow a_S[max_m]$ and $nh_m[max_m - 1] \leftarrow 0$.

Maintenance of Routing Tables

As the membership of the overlay changes, some LRCs become obsolete and new LRCs need to be created. To maintain the LRCs evenly distributed in face of membership changes, we periodically delete the least recently used LRC of some randomly selected levels. In our experiments, path lengths did exhibit low sensitivity to variations of the deletion period. Nodes should also purge hanging LRCs that point to neighbors that left. To do this, nodes can send periodic beacons to their neighbors. Alternatively, we can trade this beacon traffic by latency, by using, again, a lazy approach. In this latter solution, nodes only detect that a LRC is hanging when they try to use it. We evaluate this lazy approach in Section 6.2.8.

Algorithm 6.1 Hop Level algorithm

```
{Executed at node $F$ when forwarding $m$ to node $N$}
{Control information carried in message $m$:}
    {$max_m$ — highest valid level; $p_m$ — level of LRC used to reach $F$;}
    {$\forall k \in [0, max_m] : nh_m[k], s_m[k], a_m[k]$ — resp., number of hops, first node and whether there are available slots in the first node for level-$k$;}
 1: $l \leftarrow$ level of LRC from $F$ to $N$ ($F \xrightarrow{l} N$)
 2: if $p_m = \perp$ or $p_m < l$ then
 3:     $max_m \leftarrow l+1$; $lim \leftarrow max_m$
 4: else
 5:     $lim \leftarrow p_m$
 6: end if
 7: for all $k \in \{l, \dots, lim-1\}$ do
 8:     $s_m[k+1] \leftarrow F$; $a_m[k+1] \leftarrow a_F[k+1]$; $nh_m[k] \leftarrow 0$
 9: end for
10: $nh_m[l] \leftarrow nh_m[l]+1$
11: while $nh_m[l] \geq b$ do
12:     $nh_m[l] = 0$
13:     if $a_m[l+1] > 0$ then
14:         instruct $s_m[l+1]$ to create LRC $s_m[l+1] \xrightarrow{l+1} N$
15:     end if
16:     $l \leftarrow l+1$;
17:     if $max_m == l$ then
18:         $max_m \leftarrow max_m+1$; $nh_m[max_m-1] \leftarrow 0$
19:         $s_m[max_m] \leftarrow s_m[max_m-1]$; $a_m[max_m] \leftarrow a_m[max_m-1]$
20:     end if
21:     $nh_m[l] \leftarrow nh_m[l]+1$
22: end while
```

6.2.6 Signaling Cost

The Hop Level mechanism presented in Algorithm 6.1 requires $O(\log n \times (\log b + \log N))$ bits in each message to store the arrays $nh[k]$ and $s[k]$ (if no slot is available $s[k]$ may be left empty), where n is the effective number of nodes and N is the size of the virtual identification space. This can be reduced by making nodes store back pointers to previous hops, instead of using the array $s_m[k]$. Back pointers will only require a limited amount of memory at nodes, as they can be cleaned periodically. This reduces the size needed to store the arrays to $O(\log n \times \log b)$ by message. Since addresses carried in the messages need $O(\log N)$ bits, this is an acceptable cost for practical uses. In this way, total cost of messages to create LRCs may be reduced to $O(\log N)$ by LRC by hop, which is similar to other overlay networks.

6.2.7 Hop Level in a Ring

In rings, the behavior of Hop Level is particularly favorable. In fact, in a steady state ring, we expect nodes to eventually end up with levels corresponding perfectly to the number of hops of the LRC. Even if many membership changes had occurred in the past, the deletion mechanism that we use, will allow the Hop Level mechanism to rebuild the LRCs. Hence, starting from the lower levels, LRCs will eventually perfectly reflect distances (in hops) according to their levels. Then, each hop will reduce the distance by at least a factor of b. For instance, with $b = 2$ each hop reduces the distance to destination by a factor of, at least, 2. We believe that Hop Level can be used with only a few modifications to *determinstically* ensure logarithmic path lengths in steady state rings.

6.2.8 Evaluation

Experiment Settings

In this section we experimentally evaluate Hop Level with $b = 2$. Most experiments, including the comparison with eCAN-like mechanism (Chapter 3) use a Delaunay triangulation as the underlying lattice. However, for benchmarking purposes, we have also used a mapping of a two-dimensional space into a one-dimensional ring. In our experiments we evaluate the following aspects:

i) the behavior of Hop Level, when different limits for LRCs by level are used; this includes knowing the distribution of the LRCs by the levels;

ii) the behavior of Hop Level, when compared with a mechanism derived from the "expressway CAN" (Xu & Zhang, 2002), which we call the "eCAN-like" mechanism, both in balanced and extremely unbalanced scenarios;

iii) the behavior of Hop Level in a ring;

iv) the cost of the bootstrap mechanism of Hop Level; and, finally,

v) the behavior of Hop Level in dynamic settings, including settings with strong membership variation, i.e., under churn.

In the tests, arbitrary pairs of nodes exchange a large number of messages in networks with sizes ranging from 100 to $50,000$ nodes. To route the messages we have used the greedy routing algorithm, because it has good performance and it works both in the underlying lattice and with LRCs, without requiring any extensions. Furthermore, it agrees to the conditions of Section 6.2.3. Hence, next hop is always the neighbor (connected by a short or long range contact) closest to destination. To let Hop Level LRC scheme converge, and depending on the network size, we routed up to $1,000,000,000$ different messages and only used the final 3000 paths in the evaluation of path lengths. Nevertheless, we also show that our mechanism achieves good routing performance much earlier than that. To test unbalanced distributions of nodes we used a truncated Gaussian bivariate distribution with standard deviations of 0.01 in a $[0, 1] \times [0, 1]$ square.

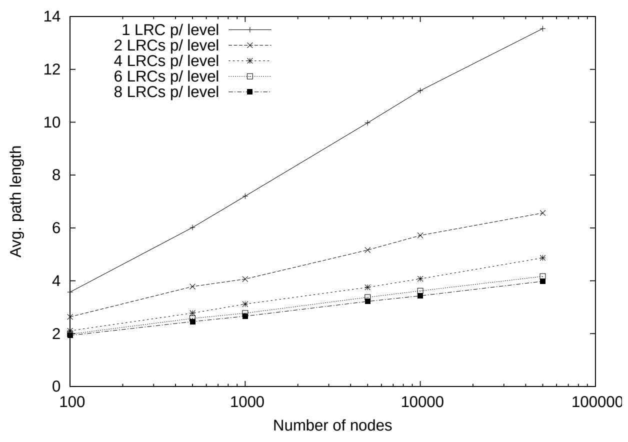


Figure 6.4: Average path lengths for Hop Level

Number of LRCs per Level

The first aspect that we evaluate is the performance achieved by different configurations of the Hop Level mechanism. The goal is to determine the limit for the number of LRCs per level that ensures the most reasonable compromise between path lengths and node degrees. Figures 6.4 and 6.5 respectively show the average path lengths (in number of hops) and the average number of LRCs used by each node for different network sizes and for different configurations of the Hop Level mechanism: with 1, 2, 4, 6 and 8 LRCs per level.

We can see that all configurations achieve an approximately logarithmic/logarithmic trade-off (a logarithmic growth is represented by a straight line). We believe that this is quite an interesting aspect, because it minimizes the need for manual configuration of parameters. Naturally, the largest the number of LRCs per node, the more Hop Level trades node degree for path lengths (to be more precise, in our experiments we observed that 8 is near the limit worthwhile using in a Delaunay triangulation). Knowing the trade-offs achieved by each configuration of the mechanism

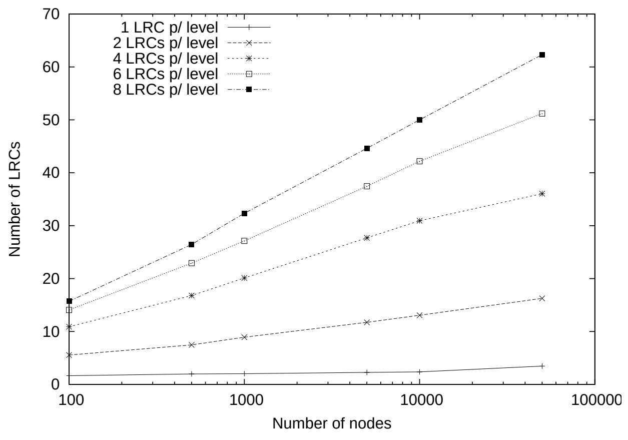


Figure 6.5: Average LRCs per node for Hop Level

has the obvious advantage of letting the user choose the configuration that best fits his/her needs. In the case of a Delaunay triangulation, we can observe that 4 or 6 LRCs per level seem to be very reasonable choices, because path lengths are close to those achieved with 8 LRCs, but at the smaller cost of using fewer LRCs. Hence, in the rest of our experiments, we have fixed the limit of LRCs per level to 6.

Figure 6.6 shows how many LRCs exist on the entire network and the average length of those LRCs for each hop level. To do this evaluation, we have used a $50,000$ node network with a balanced distribution of nodes, because a balanced distribution allows to reason in terms of distance. The effect of truncating the number of LRCs per level to 6 is quite evident in the figure. Therefore, knowing the growth of the number of levels suffices to determine the growth of the number of LRCs per node. To do this analysis, we will look at the distances of the LRCs. As the level increases, LRCs become farther away and eventually borders of the square will start to limit their distance. This effect totally dominates the growth of the distance in the last levels. Before this becomes evident, growth ratio of the distances is nearly constant from level to level, experiencing only a

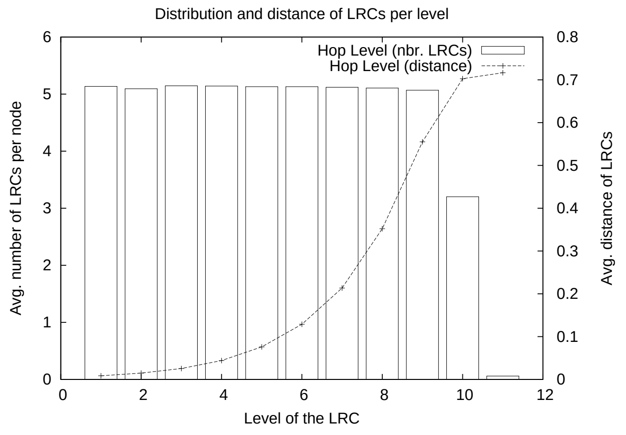


Figure 6.6: Distribution of LRCs per level (50, 000 node balanced network)

small decay in the higher levels (from 1.75 to 1.58, for the levels with all the LRCs). Hence, this reasoning points to the conclusion that the number of levels is approximately logarithmic, because distance growth from one level to the next is approximately exponential.

Comparison with "eCAN-like"

To offer some comparative measurement, we ran our scheme against a benchmark mechanism called "eCAN-like" (Section 3.1.4). This benchmark results from an adaptation of the eCAN (Xu & Zhang, 2002) logarithmic/logarithmic node degree/path length mechanism (whose applications most closely resemble those of our own algorithm). We must emphasize that the resulting mechanism, "eCAN-like", is a simplified version of the complete eCAN solution, that only captures the fundamental impact of the expressways in routing. It does not attempt to reproduce other features of eCAN (such as the mechanisms that provide support for complex interaction schemes like publish/subscribe). In spite of these simplifications, we believe that our implementation of expressways mimics the eCAN LRC mechanism with enough accuracy to allow a fair comparison.

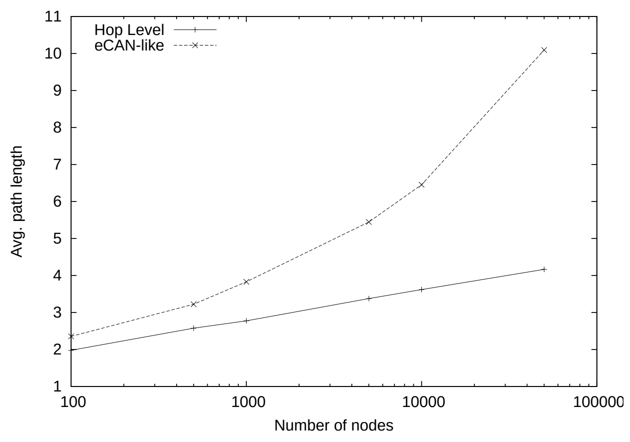


Figure 6.7: Average path lengths (Hop Level vs. eCAN-like)

Results for the most unbalanced network are depicted in Figure 6.7. The number of LRCs is not depicted because it is constant in eCAN-like. When the network is balanced (cases not shown here), eCAN-like network behaves perfectly well and achieves logarithmic path lengths, given that enough LRCs are provided. The problem in fact is to know how many LRCs should be used and from the figures it is clear that eCAN-like is no longer logarithmic when distribution of nodes is very unbalanced. The reason for the bad behavior of the eCAN-like mechanism is easily explainable: density of LRCs is no longer enough near the center and routing to nearby nodes will tend to become linear with the number of hops in the lattice, instead of logarithmic. Increasing the levels of the LRCs would solve the problem, unless, of course, density in some places were also increased. A solution to this limitation could be to use a brute-force approach with a number of LRCs logarithmic to the space granularity (i.e., to the size of the virtual identification space). On the contrary, Hop Level is strongly decoupled from the identification space and, not only node distribution has little impact on its performance, but no configuration is needed for different distributions of nodes in the space of identifications.

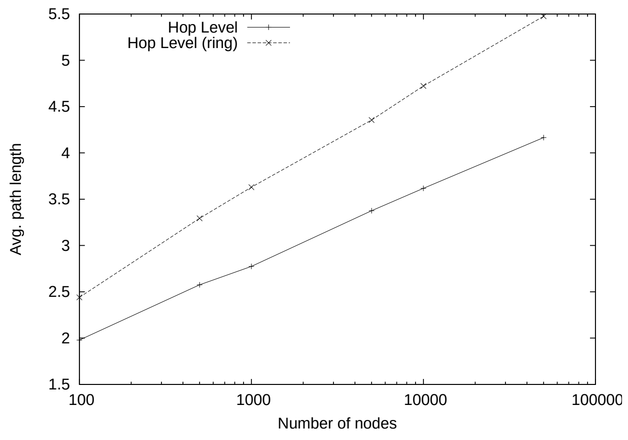


Figure 6.8: Average path lengths (Hop Level vs. Ring)

Evaluation of Hop Level in a Ring

Now, we determine the behavior of Hop Level in the scenario described in Section 6.2.3, where the two-dimensional space is mapped into a ring. Experimental results for the most unbalanced network are depicted in Figure 6.8 and 6.9. As expected, behavior of Hop Level in a ring is nearly perfect with the additional advantage of not requiring some previously established limit for the number of LRCs per level, because this number is at most two in the ring. On the other hand, the smaller connectivity of a ring, when compared to a Delaunay triangulation with the corresponding fewer LRCs created, has some cost in terms of achievable performance, as paths are typically longer.

Network Convergence

Perhaps one of the central aspects in the evaluation of Hop Level is to know how fast do the path lengths converge to their optimal value and, reciprocally, how fast do the nodes create their LRCs. In both cases, we can observe that for all network sizes under test, a short number of mes-

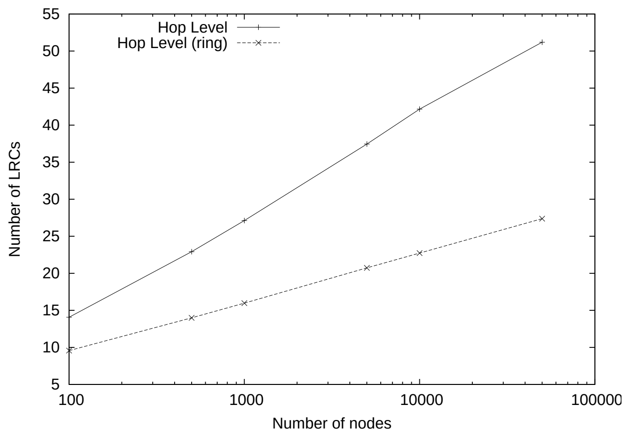


Figure 6.9: Average number of LRCs per node (Hop Level vs. Ring)

sages suffices to let the network reach a state similar to a steady state. Figures 6.10 and 6.11 show for networks with, respectively, 100 and $10,000$ nodes, the growth in the number of LRCs of the entire network and the reduction in the path lengths (note that the x-axis is logarithmic). As we can see, the slope of the curves is very high when network is recent (remember that until this point we are considering a static setting) and, as network grows older, it suddenly becomes very low. For all network sizes we tested, path lengths within 3 times the optimal can be achieved well before 5 messages have been generated by each node.

Although these results show that network converges to a steady state very fast, such fact would not be of much relevance in a static setting where nodes enter the network at once and never leave, because sooner or later a final state would be reached. Hence, what we still need to determine is the behavior of our mechanism in more realistic environments where nodes can enter and leave the network. In a dynamic environment there is an inevitable trade-off between the amount of routing information given to a node (to allow the node to take good routing decisions) and the capacity of the network to cope with membership changes. The larger the

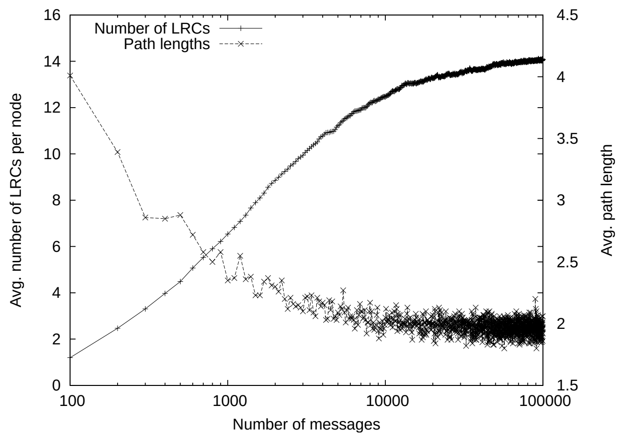


Figure 6.10: Average path lengths and number of LRCs for a network with 100 nodes

former the smaller the latter and vice-versa. Therefore, in the next section we analyze the behavior of Hop Level in dynamic environments.

Dynamic Settings

In this section we will use settings similar to the ones described in Araneola (Melamed & Keidar, 2004), which are based in real measurements (Almeroth & Ammar, 1996; Saroiu *et al.*, 2002). Hence, we assume that around 7% of the nodes are permanent, i.e., they boot up with the network and never go away for the rest of the life of the network. The remaining 93% of the nodes are non-permanent and can enter or leave network at any instant and repeatedly do so. When a node enters the network it becomes active, when it leaves it goes to sleep state. When network starts, non-permanent nodes are neither active nor sleeping, but in a fourth state that we can call as *out*. This means that the network starts with 7% of the permanent nodes. Then, the bootstrap process brings 50 new nodes from *out* to active or sleep states with equal probabilities at each time

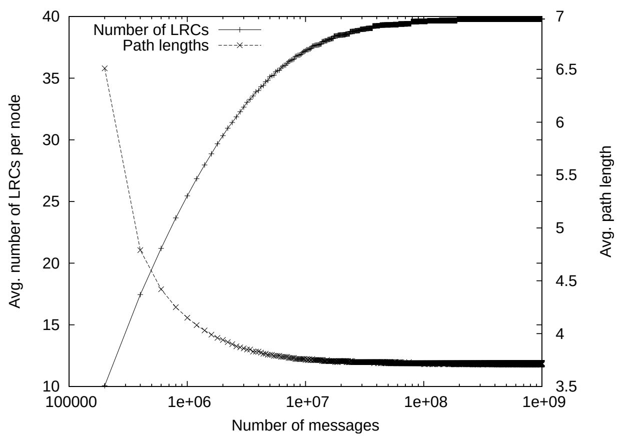


Figure 6.11: Average path lengths and number of LRCs for a network with 10000 nodes

step[3]. After each time step, any non-permanent node that is either active or sleeping can switch from one state to the other with a given fixed probability — this simulates the churn (note that nodes reenter the network in a fresh state, i.e., without any LRCs originating or pointing to it). A node can never return to the *out* state. Therefore, joins and leaves are modeled by an exponential distribution. In summary, network starts with few participants, then the number of participants starts to grow steadily until some limit. Furthermore, during all over the network life, there are some (non-permanent) nodes that are constantly entering and departing. The reader should notice that, since a node can be in active or sleep state with equal probabilities, in a test with $2,000$ nodes, average network size will be slightly above $1,000$ (due to the permanent nodes), after the bootstrap process. The main parameter to vary in this experiment is the rate at which nodes enter and leave the network or, in other words, the average lifetime of non-persistent nodes. The probability of switching state after a time step is varied from 0.00005 to 0.0025. In the Hop Level mechanism,

[3]A time step is counted after 50 messages.

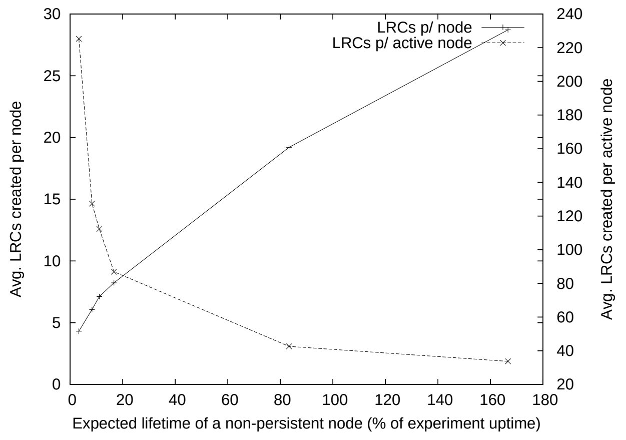


Figure 6.12: Number of LRCs created under churn

churn is associated with two types of costs: the signaling cost of changing network topology and the cost of worse routing performance. In fact, since Hop Level uses a kind of lazy reconstruction of the network of LRCs, routing performance necessarily degrades if fewer LRCs are available as a consequence of young nodes.

Figure 6.12 shows the number of LRCs created in the network under churn. This corresponds to the signaling cost of the Hop Level mechanism. Analysis of this cost may be done according to two perspectives. From the perspective of active non-persistent nodes, there is no problem with their short existence in the network. On the contrary, the shorter the lifetime, the fewer LRCs such a node will create (as opposed to most DHTs, where this number is independent of the churn rate). This corresponds to the line deemed "LRCs p/ node". On the other hand, the load for the network and for the persistent nodes increases with churn. This is represented in the line deemed "LRCs p/ active node", which shows the total number of LRCs created in the network, divided by the average number of active nodes. We can see that even with very small lifetimes

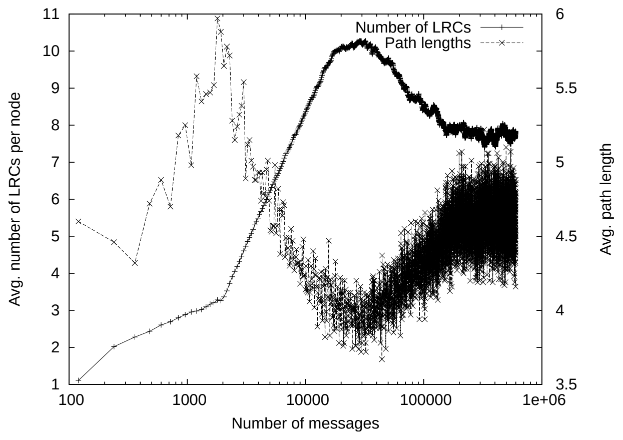


Figure 6.13: Path lengths and active number of LRCs under churn

(in the order of 20,000 generated messages in a 1,000 node network, i.e., around 20 messages per node), the growth in the number of LRCs created per active node is moderate, when compared to scenarios where average lifetime is 50 times longer.

The second cost to be paid by churn is in the routing performance degradation. This is illustrated in Figure 6.13 for a non-persistent node's lifetime of 6.7% of experiment up time (66,642 messages, i.e, a node sends around 66 messages before it leaves). Pattern seen in this graphic is similar for other average lifetimes, with the following difference: shorter lifetimes (more churn) converge faster to a smaller stabilized number of active LRCs, which means that smaller lifetimes will result in longer paths (slightly above 4.5 in the case shown, around 5 hops for the shortest lifetime tested). The graphics of this figure show an overshoot that is due to the bootstrap process. As network density keeps growing, the LRCs of a node get more distributed around the network (even without changing). Hence, this leaves space for the creation of more LRCs. Some time after the number of nodes stabilizes, this process ends and the number of LRCs per (new entering) node starts to decay until it stabilizes to a value that depends on the churn rate.

We have also checked the cost imposed by hanging LRCs that point to nodes that are gone. If no action is taken to remove these LRCs, nodes may try to use them to route. We have observed that, in this case, the number of messages that follow hanging LRCs, ranges from 1.2% to 13.3%, for the two extreme churn rates. Pro-actively looking for hanging LRCs, to trade-off message latency for network bandwidth utilization, is also possible, but this should follow the conclusions of Rhea *et al.* (2004). For instance, this search should be made periodically and never in reaction to some event that may lead to a positive feedback capable of congesting the network. Another aspect of interest is to know the fraction of LRCs that are kept by the permanent nodes. As the churn grows, this fraction grows somewhat moderately between approximately 17% and 23%. In fact, the limitation imposed on the number of LRCs per level, together with the limited diameter of the network, ensures that nodes do not get a disproportionately large number of LRCs.

These results show that performance of Hop Level is nearly optimal and independent of node distribution in space. Furthermore, its lazy way of creating LRCs enables Hop Level to resist churn very well without compromising performance in fresh networks (i.e., with respect to churn, Hop Level is highly self-configurable). For these reasons, we believe that the Hop Level mechanism is very well suited to support efficient multidimensional range queries in Distributed Storage Systems as well as position-based DHTs.

6.3 Summary

In this chapter we presented a position-based DHT for wired networks, called "GeoPeer" and a complementary mechanism called "Hop Level". GeoPeer uses a Delaunay triangulation to route and to distribute the keys. It can take advantage of position to support a number of services like geographically-scoped multicasts or queries. Since GeoPeer suffers from long path lengths, Hop Level creates and maintains LRCs that augment the Delaunay triangulation. The positive aspects of Hop Level are the resistance to churn and the possibility of using this mechanism to support more complex distributed storage systems.

Notes

The work that we described in this chapter is part of an architecture that intends to bring quality of service (QoS) to publish/subscribe systems, using a DHT. The work on such system has followed two main threads: one focused on the DHTs and the other on the publish/subscribe system itself. GeoPeer and Hop Level are part of the DHT thread and result mainly from the work of the author and of Professor Luís Rodrigues. Part of this work has been previously published in the following conferences:

- Filipe Araújo and Luís Rodrigues. GeoPeer: A location-aware peer-to-peer system. In *The 3rd IEEE International Conference on Network Computing and Applications (NCA '04)*, pages 39–46, Cambridge, MA, USA, August 2004.
- Filipe Araújo and Luís Rodrigues. Long range contacts in overlay networks. In *Euro-par 2005*, pages 1153–1162, Lisbon, Portugal, August 2005. Springer-Verlag, LNCS 3648.

7 Conclusions and Future Work

7.1 Conclusions

In this thesis we have addressed the problem of deploying scalable, fault-tolerant and self-configuring dictionaries in highly dynamic networks, like peer-to-peer overlays and wireless *ad hoc* networks. Our approach to this problem consisted of combining two existing methods: distributed hash tables (DHTs) and position-based routing. While modern DHTs already own a number of attractive features to support large-scale dictionaries, they are not suitable for all kinds of networks. For instance, it could be difficult to implement such an overlay on top of a wireless *ad hoc* network. Also, in wired networks, churn raises a very difficult problem, as state information often changes too fast for the available bandwidth. To overcome these problems, we argued that we can use positional information to create efficient DHTs. Positional information allows to simultaneously solve the routing problem and to support operation of the DHT. The simplicity of position-based routing, allows us to strongly reduce the amount of control information, thus enabling implementation of scalable DHTs in several environments.

To demonstrate the efficiency of position-based DHTs, we developed and evaluated the following mechanisms and algorithms: the Fast Localized Delaunay Triangulation, which is an algorithm for wireless *ad hoc* networks that creates a triangulation in a single communication step with the optimal cost of $O(n \log n)$; the cluster-based Cell Hash Routing (CHR) DHT; the GeoPeer peer-to-peer network, which we can regard as the wired counterpart of FLDT, because it creates and maintains a complete Delaunay triangulation for wired networks; and the Hop Level mechanism, to augment the Delaunay triangulation with a set of long range contacts (LRCs).

As a result of this work, we can outline the following conclusions:

- utilization of the Unit Disk Graph (UDG) model raises some important questions. On one side we can take advantage of it, to create simple and provably correct algorithms like FLDT. On the other hand, this proves do not hold in real-life models. Additionally, inaccuracies in position may also cause inconsistencies in the final graph that may ultimately lead to the loss of some packets;

- in arbitrary non-UDG connectivity models, there is no way to prevent intersections using a localized algorithm. This is a serious drawback for position-based routing schemes, not to mention the difficulties in determining position of nodes and the irregularities in the connectivity model. While evidence seems to demonstrate that shortest path algorithms like AODV (Perkins, 1997) or DSR (Johnson & Maltz, 1996) are not suitable to large scale wireless *ad hoc* networks, it is unclear whether the position-based routing can hold to its promises;

- in this thesis, we have shown that it is possible to use clustering as a very effective way to create and maintain routing and DHT operation, especially when the density of nodes increases;

- position-based clustering may bring two additional advantages: higher tolerance for more inexact localization schemes and lower dependency of the *UDG* model. The clustered scheme of CHR could use local exchange of routing information to overcome the inaccuracies of the connectivity model and still take advantage of the localized approach of position-based routing;
- one of the problems of using real positions, instead of virtual identifications to create the DHTs, is that distribution of the items of the DHT among the nodes may become uneven. At the present we are not aware of any simple and elegant way of tackling this issue;
- churn raises some of the most difficult problems to the self-configurability in wired DHTs. Earlier work showed that there is a trade-off between bandwidth wasted by the routing scheme and the observed latency. Our contribution to this problem was to show that, using a lazy creation of LRCs, we can resist to very high churn rates and still have only limited increases in bandwidth utilization, with a small impact on latency.

7.2 Future Work

The work of this thesis leaves some open problems that we consider of interest for future research, including the following:

Relaxing the *UDG* model we believe that it is possible to rely on some connectivity models more relaxed than *UDG*. We believe that either FLDT, or variations of FLDT can easily cope with more relaxed and, hence, more realistic models. We conjecture that these algorithms can work even if nodes only have a partial view of their unit disk graph, possibly in a way that is related to the position of their neighbors;

Eliminating the UDG model constraint unlike the previous case, where the idea is to relax the UDG model, here we try to remove any assumption about the connectivity. We believe that the characteristics of CHR turn it into a promising platform to create a routing scheme (without necessarily supporting the DHT) that goes beyond the UDG model. As we referred before, the idea is to use position-based routing in the large scale and topology information exchange in the small scale. Such solution could use routing information to reach only the invisible nodes inside the cell or adjacent cells. This would combine the scalability of topology control algorithms with the resistance of shortest path algorithms to connectivity models that are more general than UDG. Additionally, this technique could easily tolerate an incorrect determination of position. Also, we believe that this approach will also be the best way to solve the problem of cluster-induced disconnection that we mentioned in Chapter 5;

Adaptability of Hop Level to churn we also believe that it would be interesting to have LRCs schemes capable of achieving different trade-offs between latency and bandwidth. For instance, instead of following a purely lazy approach, it could be interesting to create some LRCs when a node entered the network, but still following a lazy approach to replace the LRCs. Or, one could periodically check availability of LRCs and use a lazy creation. This could be extended to a scenario where a DHT could adapt its routing scheme to the churn rate and to the available bandwidth.

Support of Quality of Service

A very interesting aspect that we left outside the scope of this thesis was to introduce quality of service (QoS) information in a DHT. We took early steps to create a publish/subscribe system with support for the QoS parameters bandwidth and latency (see Araújo & Rodrigues, 2002; Carvalho *et al.*, 2003), but to conserve space we did not include description of this work in the thesis. Such publish/subscribe system works on top of a "classic" (non-position-based) DHT. The nodes of the DHT do not propagate the QoS information, which stays local to their links. Each reservation of resources must go through a small number of predetermined paths (using different rendezvous nodes, like the rendezvous nodes of Scribe (Rowstron *et al.*, 2001) or Hermes (Pietzuch & Bacon, 2002)) to find the best alternative. We describe this idea in the work of Carvalho *et al.* (2005).

A promising future step to support QoS publish/subscribe systems would be to place the resources, namely the rendezvous nodes, nearer the clients. If clients share a geographical relation (for instance they could all belong to the same university, city or country) it makes sense to use position to place the rendezvous node(s) in the zone of the clients.

Towards Global GeoPeer

One of the concerns of this thesis was to reduce the gap between the routing scheme and the DHT, to increase the scalability of the DHT in highly dynamic networks with stringent bandwidth limitations. We have shown

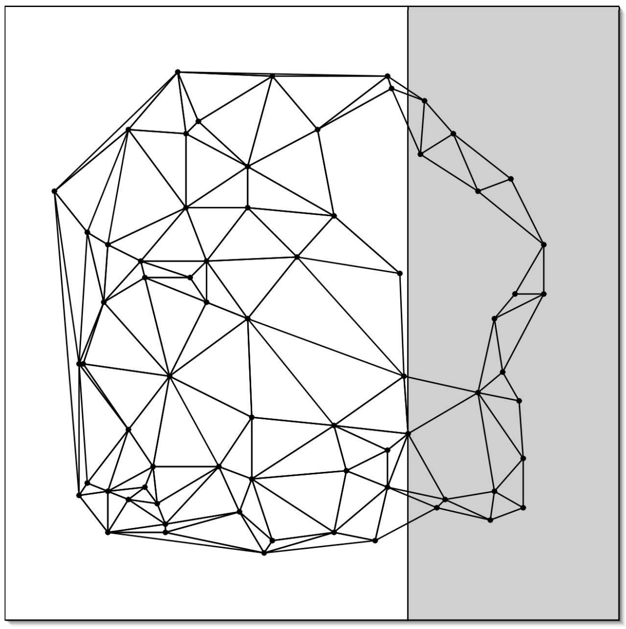


Figure 7.1: Global wired and wireless network based on Delaunay triangulations

that using positional information was a viable approach for wireless as well as wired networks, as long as the topology matched the position of nodes. The use of position enables the creation of a global architecture that integrates both types of networks in a seamless way. We outline the possibility of using a position-based DHT that spans across the wired and the wireless. Wired as well as wireless nodes would participate in the DHT and share the storage of keys, according to their geographical position. In Figure 7.1, we depict a possible global DHT like this, based on Delaunay triangulations. In the gray area, at the right side of the figure, nodes are purely wireless, while in the white area, at the left, nodes are wired. In this figure we assumed that some of the wired nodes also have wireless interfaces, to ensure the connectedness of the network. The implementation of such a Global GeoPeer is a very interesting follow up of our work.

As a final remark to this work, we can say that the use of position is a promising approach to create efficient DHTs. Positional information makes it possible to use localized routing schemes, which is specially useful in wireless *ad hoc* networks and other settings where bandwidth is very scarce to the pace of topological change. Despite these promises, work on position-based DHTs (and position-based routing schemes) is still ongoing, due to some open problems that persist. Most notably, future work should focus on making position-based routing more robust in real life settings where the UDG model does not hold.

Bibliography

ABRAHAM, ITTAI, MALKHI, DAHLIA, & DOBZINSKI, OREN. 2004. LAND: stretch $(1+\varepsilon)$ locality-aware networks for DHTs. *Pages 550–559 of: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics.

ALMEROTH, KEVIN C., & AMMAR, MOSTAFA H. 1996 (August). Collecting and Modeling the Join/Leave Behavior of Multicast Group Members in the MBone. *Pages 209–216 of: High Performance Distributed Computing (HPDC '96)*.

ARAÚJO, FILIPE, & RODRIGUES, LUÍS. 2002 (July). On QoS-Aware Publish-Subscribe. *Pages 511–515 of: The 22nd IEEE International Conference on Distributed Computing Systems Workshops (DEBS '02)*.

ASPNES, JAMES, KIRSCH, JONATHAN, & KRISHNAMURTHY, ARVIND. 2004 (July). Load Balancing and Locality in Range-Queriable Data Structures. *In: Twenty-Third Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2004)*.

BALLARDIE, A. 1997 (September). *Core Based Trees (CBT version 2) Multicast Routing*. Request for Comments 2189.

BARRIÈRERE, LALI, FRAIGNIAUD, PIERRE, KRANAKIS, EVANGELOS, & KRIZANC, DANNY. 2001. Efficient Routing in Networks with Long Range Contacts (Extended Abstract). *In:* WELCH, JENNIFER (ed), *15th Internation Conference on Distributed Computing*. Lecture Notes in Computer Science, no. LNCS 2180. Lisbon, Portugal: Springer.

BARRIÈRE, LALI, FRAIGNIAUD, PIERRE, NARAYANAN, LATA, & OPATRNY, JAROSLAV. 2002. Robust Position-Based Routing in Wireless Ad Hoc Networks with Irregular Transmission Ranges. *Wireless Communications And Mobile Computing journal*.

BELLMAN, R.E. 1957. *Dynamic Programming*. Princeton, N.J.: Princeton University Press.

BHAGWAN, RANJITA, SAVAGE, STEFAN, & VOELKER, GEOFFREY M. 2003. Understanding Availability. *In:* (Kaashoek & Stoica, 2003).

BHARAMBE, ASHWIN R., AGRAWAL, MUKESH, & SESHAN, SRINIVASAN. 2004. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev.*, **34**(4), 353–366.

BLOUGH, D., & SANTI, P. 2002. Investigating Upper Bounds on Network Lifetime Extension for Cell-Based Energy Conservation Techniques in Stationary Ad Hoc Networks. *In: ACM Mobicom*.

BONDY, J. A., & MURTY, U. S. R. 1976. *Graph Teory with Applications*. Elsevier North-Holland.

BOSE, PROSENJIT, & MORIN, PAT. 1999. Online Routing in Triangulations. *In: 10th Annual Internation Symposium on Algorithms and Computation (ISAAC)*.

BOSE, PROSENJIT, & MORIN, PAT. 2001. Competitive online routing in geometric graphs. *Pages 35–44 of: 8th Colloquium on Structural Information & Communication Complexity.* Carleton University Press.

BOSE, PROSENJIT, MORIN, PAT, STOJMENOVIC, IVAN, & URRUTIA, JORGE. 1999. Routing with Guaranteed Delivery in *ad hoc* Wireless Networks. *Pages 48–55 of: International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM).*

BOSE, PROSENJIT K., DEVROYE, LUC, EVANS, W., & KIRKPATRICK, DAVID. 2002. On the spanning ratio of Gabriel graphs and beta-skeletons. *Pages 479–493 of:* RAJSBAUM, SERGIO (ed), *Proc. 5th Latin American Symp. Theoretical Informatics (LATIN 2002).* Lecture Notes in Computer Science, no. 2286. Springer-Verlag.

CALINESCU, G. 2003. *Computing 2-Hop Neighborhoods in Ad Hoc Wireless Networks.* Adhoc-Now '03.

CALINESCU, G., MANDOIU, I., WAN, P.J., & ZELIKOVSKY, A. 2001. Selecting Forwarding Neighbors in Wireless Ad Hoc Networks. *In: Fifth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIALM).*

CARVALHO, NUNO, ARAÚJO, FILIPE, & RODRIGUES, LUÍS. 2003 (Setembro). IndiQoS: um Sistema Publicação-Subscrição com Qualidade de Serviço. *In: 6a Conferência sobre Redes de Computadores (CRC 2003).*

CARVALHO, NUNO, ARAÚJO, FILIPE, & RODRIGUES, LUÍS. 2005 (July). Scalable QoS-Based Event Routing in Publish-Subscribe Systems. *In: The 4th IEEE International Conference on Network Computing and Applications (NCA '05).*

CASTRO, MIGUEL, DRUSCHEL, PETER, KERMARREC, ANNE-MARIE, & ROWSTRON, ANTONY. 2002. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC).*

CHAWATHE, YATIN, RATNASAMY, SYLVIA, BRESLAU, LEE, LANHAM, NICK, & SHENKER, SCOTT. 2003. Making gnutella-like P2P systems scalable. *Pages 407–418 of: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications.* ACM Press.

CHEN, G., & STOJMENOVIC, I. 1999 (June). *Clustering and Routing in Wireless Ad Hoc Networks.* Tech. rept. TR-99-05. Department of Computer Science, SITE, University of Ottawa, Ottawa, Ontario K1N 6N5, Canada.

C.R.LIN, & GERLA, M. 1997. Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal Selected Areas in Communication*, **15**(7), 1265–1275.

DAS, BEVAN, & BHARGHAVAN, VADUVUR. 1997. Routing in Ad-Hoc Networks Using Minimum Connected Dominating Sets. *Pages 376–380 of: ICC (1).*

DIJKSTRA. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 269–271.

DOBKIN, D., FRIEDMAN, S. J., & SUPOWIT, K. J. 1990. Delaunay Graphs are Almost as Good as Complete Graphs. *Discrete Computational Geometry*, July.

DOUCEUR, J., & WATTENHOFER, R. 2001. Optimizing File Availability in a Secure Serverless Distributed File System. *Pages 4–13 of: Proceedings of 20th IEEE SRDS.*

DRUSCHEL, P., & ROWSTRON, A. 2001 (May). PAST: A large-scale, persistent peer-to-peer storage utility. *In: HotOS VIII.*

DUCHON, PHILIPPE, HANUSSE, NICOLAS, LEBHAR, EMMANUELLE, & SCHABANEL, NICOLAS. 2005. Could any graph be turned into a small-world? *Special issue of the international journal Theoretical Computer Science on Complex Networks.*

EPPSTEIN, D. 2000. Spanning Trees and Spanners. *Pages 425–461 of: Handbook of Computational Geometry.* Amsterdam: Elsevier North-Holland.

ERIKSSON, JAKOB, FALOUTSOS, MICHALIS, & KRISHNAMURTHY, SRIKANTH. 2004 (February). Scalable Ad Hoc Routing: The Case for Dynamic Addressing. *In: IEEE Infocom 2004.*

FEENEY, LAURA MARIE, & NILSSON, MARTIN. 2001. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. *In: IEEE INFOCOM.*

FINN, GREOGRY G. 1987 (March). *Routing and Addressing Problems in Large Metropolitan-Scale Internetworks.* Tech. rept. ISU/RR-87-180. Institute for Scientific Information.

FRAIGNIAUD, P., & GAURON, P. 2003a (January). *The Content-Addressable Network D2B.* Tech. rept. 1349. LRI, Univ. Paris-Sud, France.

FRAIGNIAUD, P., & GAURON, P. 2003b (July). *An Overview of the Content-Addressable Network D2B.* Brief Announcement at 22nd ACM Symp. on Principles of Distributed Computing (PODC).

FRAIGNIAUD, PIERRE, & GAVOILLE, CYRIL. 2002. A Space Lower Bound for Routing in Trees. *In:* 19^{th} *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, vol. Lecture Notes in Computer Science. Springer.

GAO, JIE, GUIBAS, LEONIDAS J., HERSHBERGER, JOHN, ZHANG, LI, & ZHU, AN. 2001. Geometric Spanners for Routing in Mobile Networks. *In: 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 01).*

GARCÉS-ERICE, L., ROSS, K.W., BIERSACK, E.W., FELBER, P.A., & URVOY-KELLER, G. 2003. Topology-Centric Look-Up Service. *In: COST264/ACM Fifth International Workshop on Networked Group Communications (NGC).*

GUMMADI, K., GUMMADI, R., GRIBBLE, S., RATNASAMY, S., SHENKER, S., & STOICA, I. 2003a. The impact of DHT routing geometry on resilience and proximity. *Pages 381–394 of: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications.* New York, NY, USA: ACM Press.

GUMMADI, KRISHNA P., DUNN, RICHARD J., SAROIU, STEFAN, GRIBBLE, STEVEN D., LEVY, HENRY M., & ZAHORJAN, JOHN. 2003b. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. *Pages 314–329 of: SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles.* New York, NY, USA: ACM Press.

GUPTA, INDRANIL, VAN RENESSE, ROBBERT, & BIRMAN, KENNETH P. 2001. Scalable Fault-Tolerant Aggregation in Large Process Groups. *Pages 433–442 of: DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS).* IEEE Computer Society.

GUPTA, INDRANIL, BIRMAN, KENNETH P., LINGA, PRAKASH, DEMERS, ALAN J., & VAN RENESSE, ROBBERT. 2003. Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead. *In:* (Kaashoek & Stoica, 2003).

HAEBERLEN, ANDREAS, FLANNERY, ELIOT, LADD, ANDREW M., RUDYS, ALGIS, WALLACH, DAN S., & KAVRAKI, LYDIA E. 2004. Practical robust localization over large-scale 802.11 wireless networks. *Pages 70–84 of: MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking.* ACM Press.

HARVEY, NICHOLAS J. A., JONES, MICHAEL B., SAROIU, STEFAN, THEIMER, MARVIN, & WOLMAN, ALEC. 2003 (March). SkipNet: A Scalable Overlay Network with Practical Locality Properties. *In: Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03).*

HU, LINGXUAN, & EVANS, DAVID. 2004. Localization for mobile sensor networks. *Pages 45–57 of: MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking.* New York, NY, USA: ACM Press.

ITSIRADIOEQUIPMENT. 1996 (June). *Radio Equipment and Systems: High Performance Radio Local Area Network Type 1, Functional Specifications.* ITSI STC-RES10 Committee.

J. CHU, K. LABONTE, B. N. LEVINE. 2002 (July). Availability and locality measurements of peer-to-peer file systems. *In: Scalability and Traffic Control in IP Networks II.* Proceedings of SPIE, vol. 4868.

JAIN, R., PURI, A., & SENGUPTA, R. 1999. *Geographical routing using partial information for wireless ad hoc networks.*

JOHNSON, DAVID B, & MALTZ, DAVID A. 1996. Dynamic Source Routing in Ad Hoc Wireless Networks. *In:* IMIELINSKI, & KORTH (eds), *Mobile Computing*, vol. 353. Kluwer Academic Publishers.

JR., L.R. FORD, & FULKERSON, D.R. 1962. *Flows in Networks.* Princeton, N.J.: Princeton University Press.

KAASHOEK, M. FRANS, & KARGER, DAVID R. 2003. Koorde: A Simple Degree-Optimal Distributed Hash Table. *In:* (Kaashoek & Stoica, 2003).

KAASHOEK, M. FRANS, & STOICA, ION (eds). 2003. *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*. Lecture Notes in Computer Science, vol. 2735. Springer.

KAPLAN, ELLIOTT D. (ed). 1996. *Understanding GPS: Principles and Applications.* Artech House.

KARGER, DAVID, LEHMAN, ERIC, LEIGHTON, TOM, PANIGRAHY, RINA, LEVINE, MATTHEW, & LEWIN, DANIEL. 1997. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. *Pages 654–663 of: STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press.

KARGER, DAVID R., & RUHL, MATTHIAS. 2004. Simple efficient load balancing algorithms for peer-to-peer systems. *Pages 36–43 of: SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM Press.

KARP, BRAD, & KUNG, H. T. 2000. GPRS: Greedy Perimeter Stateless Routing for Wireless Networks. *In: ACM/IEEE International Conference on Mobile Computing and Networking.*

KLEINBERG, JON. 2000. The Small-World Phenomenon: An Algorithmic Perspective. *In: Proceedings of the 32nd ACM Symposium on Theory of Computing.*

KRANAKIS, E., SINGH, H., & URRUTIA, J. 1999. Compass Routing on Geometric Networks. *In: 11th Canadian Conference on Computation Geometry (CCCG 99).*

KUHN, FABIAN, WATTENHOFER, ROGER, & ZOLLINGER, AARON. 2002. Asymptotically optimal geometric mobile ad-hoc routing. *In: 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'02).*

KUHN, FABIAN, WATTENHOFER, ROGER, ZHANG, YAN, & ZOLLINGER, AARON. 2003 (July). Geometric Ad-Hoc Routing: Of Theory and Practice. *In: 22nd ACM Symposium on the Principles of Distributed Computing (PODC 2003).*

LAN, LUAN, & WEN-JING, HSU. 2002. Localized Delaunay Triangulation for Topological Construction and Routing on MANETs. *In: 2nd ACM Workshop on Principles of Mobile Computing (POMC'02).*

LI, J., JANNOTTI, J., DE COUTO, D., KARGER, D., & MORRIS, R. 2000 (August). A scalable location service for geographic ad-hoc routing. *Pages 120–130 of: Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00).*

LI, JINYANG, STRIBLING, JEREMY, GIL, THOMER M., MORRIS, ROBERT, & KAASHOEK, M. FRANS. 2004a (February). Comparing the performance of distributed hash tables under churn. *In: Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04).*

LI, MEI, LEE, WANG-CHIEN, & SIVASUBRAMANIAM, ANAND. 2004b (May). Efficient Peer-to-Peer Information Sharing over Mobile Ad Hoc Networks. *In: Second Workshop on Emerging Applications for Wireless and Mobile Access (MobEA II), in conjunction with the World Wide Web Conference (WWW).*

LI, XIANG-YANG, CALINESCU, GRUIA, & WAN, PENG-JUN. 2002. Distributed Construction of a Planar Spanner and Routing for Ad Hoc Wireless Networks. *In: The 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM).*

LIBEN-NOWELL, DAVID, BALAKRISHNAN, HARI, & KARGER, DAVID. 2002. Analysis of the evolution of peer-to-peer systems. *Pages 233–242 of: PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*. New York, NY, USA: ACM Press.

LIEBEHERR, J., NAHAS, M., & SI, W. 2001 (May). *Application-Layer Multicasting with Delaunay Triangulation Overlays*. Tech. rept. CS-2001-26. University of Virginia, Department of Computer Science, Charlottesville, VA 22904.

LYNCH, N. 1996. Distributed Algorithms. *Chap. 16, pages 691–732 of: Data Link Protocols*. Morgan-Kaufmann.

LYNCH, N., MALKHI, D., & RATAJCZAK, D. 2002. Atomic Data Access in Content Addressable Networks: A Position Paper. *In: 1st. International Workshop on Peer-to-Peer Systems (IPTPS'02)*.

MALKHI, DAHLIA, NAOR, MONI, & RATAJCZAK, DAVID. 2002 (July). Viceroy: A Scalable and Dynamic Emulation of the Butterfly. *In: Twenty-First ACM Symposium on Principles of Distributed Computing (PODC 2002)*.

MANKU, GURMEET SINGH, BAWA, MAYANK, & RAGHAVAN, PRABHAKAR. 2003. Symphony: Distributed Hashing in a Small World. *In: 4th Usenix Symposium on Internet Technologies and Systems*. http://www.usenix.org/events/usits03/.

MAYMOUNKOV, P., & MAZIÉRES, D. 2002 (March). Kademlia: A peer-to-peer information system based on the XOR metric. *In: 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*. http://www.cs.rice.edu/Conferences/IPTPS02/.

MELAMED, ROIE, & KEIDAR, IDIT. 2004 (August). Araneola: A Scalable Multicast System for Dynamic Environments. *Pages 5–14 of: The 3rd IEEE International Conference on Network Computing and Applications (NCA '04)*.

NI, S.Y., TSENG, Y.C., CHEN, Y.S., & SHEU, J.P. 1999 (August). The Broadcast Storm Problem in a Mobile Ad Hoc Network. *Pages 151–162 of: Conference on Mobile Computing, MOBICOM*.

NI, S.Y., TSENG, Y.C., & SHEU, J.P. 2001 (April). Efficient Broadcasting in a Mobile Ad Hoc Network. *Pages 16–19 of: International Conference on Distributed Computing and Systems (ICDCS'01).*

NICULESCU, DRAGOŞ, & NATH, BADRI. 2004. VOR base stations for indoor 802.11 positioning. *Pages 58–69 of: MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*. ACM Press.

PADMANABHAN, VENKATA N., & SUBRAMANIAN, LAKSHMINARAYANAN. 2001. An investigation of geographic mapping techniques for internet hosts. *SIGCOMM Comput. Commun. Rev.*, **31**(4), 173–185.

PERKINS, C. 1997. *Ad-hoc on-demand distance vector routing.*

PERKINS, CHARLES, & BHAGWAT, PRAVIN. 1994. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *Pages 234–244 of: ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications.*

PIETZUCH, P., & BACON, J. 2002. Hermes: A Distributed Event-Based Middleware Architecture. *In: 22nd IEEE International Conference on Distributed Computing Systems Workshops (DEBS '02).*

PLAXTON, C. GREG, RAJARAMAN, RAJMOHAN, & RICHA, ANDREA W. 1997. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. *Pages 311–320 of: ACM Symposium on Parallel Algorithms and Architectures.*

PUCHA, HIMABINDU, DAS, SAUMITRA M., & HU, Y. CHARLIE. 2004 (September-October). *How to Implement DHT in Mobile Ad Hoc Networks?*

Student poster, the 10th ACM International Conference on Mobile Computing and Network (MobiCom 2004).

QAYYUM, A., VIENNOT, L., & LAOUITI, A. 2000 (March). *Multipoint Relaying: An Efficient Technique for Flooding in Mobile Wireless Networks*. Tech. rept. Research Report RR-3898. INRIA.

RAGHUNATHAN, VIJAY, SCHURGERS, CURT, PARK, SUNG, & SRIVASTAVA, MANI B. 2002. Energy-Aware Wireless Microsensor Networks. *IEEE Signal Processing Magazine*, March, 40–50.

RAO, ANANTH, PAPADIMITRIOU, CHRISTOS, SHENKER, SCOTT, & STOICA, ION. 2003. Geographic routing without location information. *Pages 96–108 of: MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press.

RATNASAMY, S., KARP, B., YIN, L., YU, F., ESTRIN, D., GOVINDAN, R., & SHENKER, S. 2002 (September). GHT: A Geographic Hash Table for Data-Centric Storage in SensorNets. *In: First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*.

RATNASAMY, SYLVIA, FRANCIS, PAUL, HANDLEY, MARK, KARP, RICHARD, & SCHENKER, SCOTT. 2001. A scalable content-addressable network. *Pages 161–172 of: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press.

RHEA, S., WELLS, C., EATON, P., GEELS, D., ZHAO, B., WEATHERSPOON, H., & KUBIATOWICZ, J. 2001. Maintenance-Free Global Data Storage. *IEEE Internet Computing*, **5**(5), 40–49.

RHEA, S., GEELS, D., ROSCOE, T., & KUBIATOWICZ, J. 2003 (December). *Handling Churn in a DHT*. Tech. rept. University of California at Berkeley.

RHEA, SEAN, GEELS, DENNIS, ROSCOE, TIMOTHY, & KUBIATOWICZ, JOHN. 2004 (June). Handling Churn in a DHT. *Pages 127–140 of: USENIX 2004 Annual Technical Conference*. http://www.usenix.org/events/usenix04/.

ROWSTRON, ANTONY, & DRUSCHEL, PETER. 2001. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, **2218**, 329–350.

ROWSTRON, ANTONY I. T., KERMARREC, ANNE-MARIE, CASTRO, MIGUEL, & DRUSCHEL, PETER. 2001. SCRIBE: The Design of a Large-Scale Event Notification Infrastructure. *Pages 30–43 of: Networked Group Communication*.

SAROIU, STEFAN, GUMMADI, P. KRISHNA, & GRIBBLE, STEVEN D. 2002 (January). A Measurement Study of Peer-to-Peer File Sharing Systems. *In: Multimedia Computing and Networking (MMCN)*.

SEN, SUBHABRATA, & WANG, JIA. 2002. Analyzing peer-to-peer traffic across large networks. *Pages 137–150 of: IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. New York, NY, USA: ACM Press.

SIBSON, R. 1977. Locally equiangular triangulations. *The Computer Journal*, **21**(3), 243–245.

STOICA, ION, MORRIS, ROBERT, KARGER, DAVID, KAASHOEK, FRANS, & BALAKRISHNAN, HARI. 2001 (August). Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. *In: ACM SIGCOMM*.

STOJMENOVIC, IVAN. 2002. Position-Based Routing in Ad Hoc Networks. *IEEE Communications Magazine*, July.

STOJMENOVIC, IVAN, & LIN, XU. 2001. Loop-Free Hybrid Single-Path/Flooding Routing Algorithms with Guaranteed Delivery for Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems*, **12**(10).

STOJMENOVIC, IVAN, SEDDIGH, MAHTAB, & ZUNIC, JOVISA. 2002. Dominating Sets and Neighbor Elimination-Based Broadcasting Algorithms in Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems*, **13**(1), 14–25.

TAKAGI, H., & KLEINROCK, L. 1984. Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals. *IEEE Transactions on Communications*, **32**(3), 246–257.

VAN LEEUWEN, J., & TAN, R.B. 1995. *Compact Routing Methods: A Survey*. Tech. rept. UU-CS-1995-05. Universiteit Utrecht.

WANG, YU, & LI, XIANG-YANG. 2002. Geometric Spanners for Wireless Ad Hoc Networks. *In: The 22nd IEEE International Conference on Distributed Computing Systems*.

WANG, YU, & LI, XIANG-YANG. 2003. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. *Pages 59–68 of: DIALM-POMC '03: Proceedings of the 2003 joint workshop on Foundations of mobile computing*. New York, NY, USA: ACM Press.

WATTENHOFER, MIRJAM, WATTENHOFER, ROGER, & WIDMAYER, PETER. 2005. Geometric Routing without Geometry. *In: 12th Colloquium on Structural Information and Communication Complexity*. Mont-St-Michel, France: Springer-Verlag, LNCS 3499.

WU, J., & LI, H. 1999 (August). A Dominating Set Based Routing Scheme in Ad Hoc Wireless Networks. *Pages 7–14 of: Third International Workshop Discrete Algorithms and Methods for Mobile Computing and Communication (DIALM)*.

XU, YA, HEIDEMANN, JOHN S., & ESTRIN, DEBORAH. 2001. Geography-informed energy conservation for Ad Hoc routing. *Pages 70–84 of: Mobile Computing and Networking*.

XU, ZHICHEN, & ZHANG, ZHENG. 2002. *Building Low-maintenance Expressways for P2P Systems*. Tech. rept. HPL-2002-41. HP.

YE, FAN, LUO, HAIYUN, CHENG, JERRY, LU, SONGWU, & ZHANG, LIXIA. 2002. A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks. *In: Proceedings of ACM MOBICOM*.

ZHAO, B. Y., KUBIATOWICZ, J. D., & JOSEPH, A. D. 2001 (April). *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*. Tech. rept. UCB/CSD-01-1141. UC Berkeley.