Helsinki University of Technology Department of Communications and Networking
Teknillinen korkeakoulu Tietoliikenne- ja tietoverkkotekniikan laitos
Espoo 2010 Report 8/2010

# DISTRIBUTED RESOURCE DISCOVERY: ARCHITECTURES AND APPLICATIONS IN MOBILE NETWORKS

Nicklas Beijar

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Faculty of Electronics, Telecommunications and Automation for public examination and debate in Auditorium S4 at Aalto University School of Science and Technology (Espoo, Finland) on the 10th of December, 2010, at 12 noon.

Aalto University
School of Science and Technology
Faculty of Electronics, Telecommunications and Automation
Department of Communications and Networking

Aalto-yliopisto
Teknillinen korkeakoulu
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Tietoliikenne- ja tietoverkkotekniikan laitos

Distributor:
Aalto University
School of Science and Technology
Department of Communications and Networking
P.O. Box 13000
FI-00076 Aalto
Tel. +358-9-470 25300
Fax +358-9-470 22474



ISBN 978-952-60-3416-4
ISBN 978-952-60-3417-1 (pdf)
ISSN 1797-478X
ISSN 1797-4798 (pdf)

Multiprint Oy
Espoo 2010

Aalto University

| ABSTRACT OF DOCTORAL DISSERTATION | AALTO UNIVERSITY<br>SCHOOL OF SCIENCE AND TECHNOLOGY<br>P.O. BOX 11000, FI-00076 AALTO<br>http://www.aalto.fi/ |
|---|---|

Author Nicklas Beijar

Name of the dissertation
Distributed Resource Discovery: Architectures and Applications in Mobile Networks


| | | | |
|---|---|---|---|
| Manuscript submitted | 10.3.2010 | Manuscript revised | 27.9.2010 |
| Date of the defense | 10.12.2010 | | |
| ☒ Monograph | | ☐ Article dissertation (summary + original articles) | |

| | |
|---|---|
| Faculty | Faculty of Electronics, Communications and Automation |
| Department | Department of Communications and Networking |
| Field of Research | Networking |
| Opponent(s) | Professor Frank Fitzek (Aalborg University, Denmark) |
| Supervisor | Professor Raimo Kantola (Aalto University) |
| Instructor | |



Abstract

As the amount of digital information and services increases, it becomes increasingly important to be able to locate the desired content. The purpose of a resource discovery system is to allow available resources (information or services) to be located using a user-defined search criterion. This work studies distributed resource discovery systems that guarantee all existing resources to be found and allow a wide range of complex queries. Our goal is to allocate the load uniformly between the participating nodes, or alternatively to concentrate the load in the nodes with the highest available capacity.

The first part of the work examines the performance of various existing unstructured architectures and proposes new architectures that provide features especially valuable in mobile networks. To reduce the network traffic, we use indexing, which is particularly useful in scenarios, where searches are frequent compared to resource modifications. The ratio between the search and update frequencies determines the optimal level of indexing. Based on this observation, we develop an architecture that adjusts itself to changing network conditions and search behavior while maintaining optimal indexing. We also propose an architecture based on large-scale indexing that we later apply to resource sharing within a user group. Furthermore, we propose an architecture that relieves the topology constraints of the Parallel Index Clustering architecture. The performance of the architectures is evaluated using simulation.

In the second part of the work we apply the architectures to two types of mobile networks: cellular networks and ad hoc networks. In the cellular network, we first consider scenarios where multiple commercial operators provide a resource sharing service, and then a scenario where the users share resources without operator support. We evaluate the feasibility of the mobile peer-to-peer concept using user opinion surveys and technical performance studies. Based on user input we develop access control and group management algorithms for peer-to-peer networks. The technical evaluation is performed using prototype implementations. In particular, we examine whether the Session Initiation Protocol can be used for signaling in peer-to-peer networks. Finally, we study resource discovery in an ad hoc network. We observe that in an ad hoc network consisting of consumer devices, the capacity and mobility among nodes vary widely. We utilize this property in order to allocate the load to the high-capacity nodes, which serve lower-capacity nodes. We propose two methods for constructing a virtual backbone connecting the nodes.



| | | | |
|---|---|---|---|
| Keywords | Resource discovery, peer-to-peer, ad hoc network, distributed search algorithm | | |
| ISBN (printed) | 978-952-60-3416-4 | ISSN (printed) | 1797-478X |
| ISBN (pdf) | 978-952-60-3417-1 | ISSN (pdf) | 1797-4798 |
| Language | English | Number of pages | 220 p. |
| Publisher | Department of Communications and Networking / Aalto University | | |
| Print distribution | | | |
| ☒ The dissertation can be read at | http://lib.tkk.fi/Diss/2010/isbn9789526034171/ | | |

Aalto-yliopisto

| VÄITÖSKIRJAN TIIVISTELMÄ | AALTO-YLIOPISTO<br>TEKNILLINEN KORKEAKOULU<br>PL 11000, 00076 AALTO<br>http://www.aalto.fi/ |
|---|---|

Tekijä Nicklas Beijar

Väitöskirjan nimi
Hajautettu resurssien paikantaminen: arkkitehtuureja ja sovelluksia mobiiliverkoissa

| | |
|---|---|
| Käsikirjoituksen päivämäärä 10.3.2010 | Korjatun käsikirjoituksen päivämäärä 27.9.2010 |
| Väitöstilaisuuden ajankohta 10.12.2010 | |
| ☒ Monografia | ☐ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit) |

| | |
|---|---|
| Tiedekunta | Elektroniikan, tietoliikenteen ja automaation tiedekunta |
| Laitos | Tietoliikenne- ja tietoverkkotekniikan laitos |
| Tutkimusala | Tietoverkkotekniikka |
| Vastaväittäjä(t) | Prof. Frank Fitzek (Aalborgin yliopisto, Tanska) |
| Työn valvoja | Prof. Raimo Kantola (Aalto-yliopisto) |
| Työn ohjaaja | |

Tiivistelmä

Kun digitaalisen tiedon ja verkkopalvelujen määrät kasvavat, halutun sisällön paikantaminen muuttuu yhä tärkeämmäksi. Resurssien paikantamisjärjestelmän tehtävänä on mahdollistaa resurssien (tiedon tai palvelujen) paikantaminen käyttäjän määrittämän hakuehdon avulla. Tämä työ tutkii hajautettuja resurssien paikantamisjärjestelmiä, jotka takaavat kaikkien olemassa olevien resurssien löytämisen sekä mahdollistavat laajan joukon monimutkaisia hakuja. Tavoitteena on, että järjestelmän kuorma jakautuisi tasaisesti osallistuvien solmujen kesken tai vaihtoehtoisesti kohdistuisi suurimman kapasiteetin omaaville solmuille.

Työn ensimmäisessä osassa esitetään malleja rakenteettomien arkkitehtuurien suorituskyvyn analysointiin sekä esitetään uusia arkkitehtuureja, jotka tarjoavat erityisesti mobiiliverkkojen kannalta tärkeitä ominaisuuksia. Verkkoliikenteen pienentämiseen käytetään indeksointia, jonka hyöty näkyy erityisesti skenaarioissa, joissa haut ovat suhteellisen yleisiä resurssien muutoksiin nähden. Hakujen ja päivitysten esiintymistaajuuksien suhde määrää optimaalisen indeksoinnin. Tämän havainnon perusteella kehitetään arkkitehtuuri, joka sopeutuu hakukäyttäytymisen sekä verkon muutoksiin niin, että indeksoinnin taso pysyy optimaalisena. Sitten esitetään kattavaan indeksointiin perustuva arkkitehtuuri, jota myöhemmin sovelletaan käyttäjäryhmien sisäiseen resurssien jakoon. Lisäksi esitetään arkkitehtuuri, joka pienentää Parallel Index Clustering -arkkitehtuurin vaatimuksia topologian suhteen. Arkkitehtuurien suorituskyky arvioidaan simulointien avulla.

Työn toisessa osassa sovelletaan resurssien paikantamisarkkitehtuureja kahteen mobiiliverkkoon: matkapuhelinverkkoon sekä ad hoc -verkkoon. Matkapuhelinverkossa tarkastellaan ensin skenaarioita, joissa kaupalliset operaattorit yhdessä tarjoavat resurssien jakopalvelun, ja sitten skenaariota, jossa käyttäjät keskenään jakavat resursseja ilman operaattorin tukea. Arvioidaan mobiilivertaisverkkokonseptin toteutuskelpoisuutta mielipidekyselyjen sekä teknisten suoritusarviointien avulla. Käyttäjien palautteen perusteella kehitetään pääsynvalvonta- ja ryhmänhallinta-algoritmeja vertaisverkkoympäristöön. Tekniset arvioinnit tehdään prototyyppitoteutusten avulla. Erityisesti tutkitaan voidaanko Session Initiation Protocol -protokollaa käyttää vertaisverkkojen merkinantoon. Lopuksi tutkitaan resurssien paikantamista ad hoc -verkossa. Huomataan, että kuluttajalaitteista koostuvassa ad hoc -verkossa eri laitteiden kapasiteetit ja liikkuvuudet vaihtelevat laajasti. Tilannetta hyödynnetään siten, että kuorma kohdennetaan korkean kapasiteetin omaaville solmuille, jotka osaltaan palvelevat tehottomampia solmuja. Ehdotetaan kaksi menetelmää, joiden avulla solmuja yhdistävä virtuaalirunkoverkko rakennetaan.

| | | | |
|---|---|---|---|
| Asiasanat | resurssien paikantaminen, vertaisverkko, ad hoc verkko, hajautettu hakualgoritmi | | |
| ISBN (painettu) | 978-952-60-3416-4 | ISSN (painettu) | 1797-478X |
| ISBN (pdf) | 978-952-60-3417-1 | ISSN (pdf) | 1797-4798 |
| Kieli | Englanti | Sivumäärä | 220 s. |
| Julkaisija | Tietoliikenne- ja tietoverkkotekniikan laitos / Aalto-yliopisto | | |

Painetun väitöskirjan jakelu

☒ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/2010/isbn9789526034171/

Aalto-universitetet

| SAMMANFATTNING (ABSTRAKT)<br>AV DOKTORSAVHANDLING | AALTO-UNIVERSITETET<br>TEKNISKA HÖGSKOLAN<br>PB 11000, FI-00076 AALTO<br>http://www.aalto.fi/ |
|---|---|
| Författare Nicklas Beijar | |
| Titel<br>Distribuerad resursupptäckt: arkitekturer och tillämpningar i mobilnät | |
| Inlämningsdatum för manuskript 10.3.2010 | Datum för det korrigerade manuskriptet 27.9.2010 |
| Datum för disputation 10.12.2010 | |
| ☒ Monografi | ☐ Sammanläggningsavhandling (sammandrag + separata publikationer) |
| Fakultet | Fakulteten för elektronik, kommunikation och automation |
| Institution | Institutionen för kommunikations- och nätverksteknik |
| Forskningsområde | Nätverksteknik |
| Opponent(er) | Prof. Frank Fitzek (Aalborg Universitet, Danmark) |
| Övervakare | Prof. Raimo Kantola (Aalto-universitetet) |
| Handledare | |


Sammanfattning (Abstrakt)

Vartefter som mängden nättjänster och digital information ökar blir det alltmer viktigare att kunna hitta det önskade innehållet i nätet. Avsikten med ett resursupptäcktssystem är att låta resurser (information eller tjänster) hittas med hjälp av användardefinierade sökargument. Detta arbete studerar resursupptäcktssystem som garanterar att alla existerande resurser upphittas och tillåter ett flertal typer av komplexa förfrågan. Vårt mål är att allokera belastningen jämt mellan de deltagande noderna eller alternativt att koncentrera belastningen till noderna med störst kapacitet.

Den första delen av detta arbete undersöker prestandan av existerande ostrukturerade arkitekturer och presenterar nya arkitekturer, vilka erbjuder egenskaper som är speciellt lämpliga i mobilnät. För att reducera nätverkstrafiken använder vi indexering, som är fördelaktigt framför allt i scenarion där sökning sker oftare än modifiering av resurser. Förhållandet mellan sök- och uppdateringsfrekvenserna avgör den optimala nivån av indexering. Baserad på denna observation utvecklar vi en arkitektur som anpassar sig till varierande nätverksförhållanden och sökbeteende samtidigt som en optimal indexering upprätthålls. Vi presenterar också en arkitektur baserad på omfattande indexering som vi senare tillämpar på resursdelning inom användargrupper. Dessutom presenterar vi en arkitektur som befriar Parallel Index Clustering arkitekturen från sina krav på topologin. Arkitekturernas prestanda undersöks genom simulering.

I den andra delen av arbetet tillämpar vi arkitekturerna i två typer av mobilnät: mobiltelefonnät och ad hoc nätverk. I mobiltelefonnätet betraktar vi först scenarion där flera kommersiella operatörer erbjuder en resursdelningstjänst och sedan ett scenario där användarna sinsemellan delar resurser utan hjälp av operatören. Vi undersöker genomförbarheten av det mobila peer-to-peer konceptet genom användarundersökningar och tekniska utvärderingar. Baserat på användarnas åsikter utvecklar vi algoritmer för behörighetskontroll och grupphantering för peer-to-peer nätverk. Den tekniska utvärderingen genomförs med prototyper. Dessutom undersöker vi om Session Initiation Protocol kan användas för signalering i ett peer-to-peer nät. Till slut studerar vi resursupptäcktssystem i ad hoc nätverk. Vi observerar att nodernas kapacitet och mobilitet varierar brett i ett ad hoc nätverk bestående av konsumentapparater. Vi utnyttjar denna egenskap för att allokera belastningen till noder med hög kapacitet, vilka stöder de noder som har lägre kapacitet. Vi presenterar två metoder för konstruerande av ett virtuellt stamnät som ansluter noderna.



| | | | |
|---|---|---|---|
| Ämnesord (Nyckelord) | resursupptäckt, peer-to-peer nätverk, ad hoc nätverk, distribuerad sökalgoritm | | |
| ISBN (tryckt) | 978-952-60-3416-4 | ISSN (tryckt) | 1797-478X |
| ISBN (pdf) | 978-952-60-3417-1 | ISSN (pdf) | 1797-4798 |
| Språk | Engelska | Sidantal | 220 s. |
| Utgivare | Institutionen för kommunikations- och nätverksteknik / Aalto-universitetet | | |
| Distribution av tryckt avhandling | | | |
| ☒ Avhandlingen är tillgänglig på nätet http://lib.tkk.fi/Diss/2010/isbn9789526034171/ | | | |

# Preface

This dissertation is the result of research within two areas: ad hoc networks and peer-to-peer networks. Common to both areas is the need to locate resources – information and services – in the network. While all architectures presented in this work originally have been designed for mobile networks, they can also be used in fixed networks, and therefore the generic architectures and the network-specific functions are separately described in this dissertation.

The work was started in 2003 while I was finishing my Licentiate's thesis. Although the dissertation work represented a new research topic, the M.Sc. and Lic.Sc. topics on IP telephony routing turned out to provide a useful background for the dissertation work. The decision to do a dissertation on resource discovery was fortified when I was granted the honor of being admitted to the Graduate School of Electronics, Telecommunications and Automation (GETA), which provided funding for a period of four years starting in 2003. In addition, I am grateful for the generous financial support from the Finnish Foundation for Technology Promotion (TES) and the TeliaSonera Foundation.

The work on ad hoc networks has been carried out in the MobileMAN project funded by the European Commission under the Information Society Technologies programme. The work on peer-to-peer networks started in 2005 in a department funded project, MobileP2P, where many of the original concepts forming the work were created. In 2007 the work on peer-to-peer networking continued in the three-year Decicom project funded by Tekes, Nokia, Ericsson, and Nethawk. This project enabled the development of the ideas into the form they are presented here.

During the research work, the additional duties of teaching and instructing student work have provided me with new perspectives and an opportunity to widen my knowledge. Towards the end of the dissertation work, working in the ICT SHOK Future Internet research programme has allowed me to work on the foundations for enabling the peer-to-peer type of communication in future networks.

# Acknowledgements

First of all, I wish to express my gratitude to my supervisor, Professor Raimo Kantola, whose input and guidance have made this work possible. The proposals and challenges that he provided made the work an interesting and rewarding journey.

I wish to thank my colleagues and co-authors for their valuable participation in this work. The most influential persons include Dr. José Costa Requena and Marcin Matuszewski, with whom I have been privileged to work with in close co-operation. I also wish to thank Juuso Lehtinen, Tuomo Soinio (née Hyyryläinen), Victor Morales Reyes, Jarrod Creado and Veikko Pankakoski for their excellent work and inspiring discussions. I thank Tapio Levä, Aura Palmgren, Susanne Lagerström, and Emma Nordbäck for their contributions to the user studies. Thanks to Mikko Heikkinen for valuable input and participation in running the Decicom project. I thank William Martin for checking the language. I am also indebted to the scientific reviewers of the thesis, Prof. Jussi Kangasharju from University of Helsinki and Prof. Qin Lv from University of Colorado at Boulder. My gratitude goes to Prof. Frank Fitzek from Aalborg University for devoting his time and expertise to act as my opponent.

I am thankful for the productive collaboration with the research teams of Oulu University, Nokia, Ericsson and Jyväskylä University. In particular, acknowledgements go to Erkki Harjula, Prof. Mika Ylianttila, Jani Hautakorpi, Miguel García-Martín, and Mikko Vapa. I wish to thank my international colleagues, especially the MobileMAN team, for making traveling and conferences rewarding experiences.

I am grateful for the opportunity to work with the skillful and creative people in the Department of Communications and Networking, who create an innovative and motivating atmosphere. The good administrative and technical support from Arja Hänninen, Sanna Patana, Marja Leppäharju, Kimmo Pitkäniemi, Markus Peuhkuri, to name but a few, has been crucial in resolving all practical issues.

Many other persons deserve thanks – you know who you are.

I wish to thank my parents for their support and guidance. Last but not least, thanks to my wife Minna for all the love, encouragement and understanding. Victor, you are my greatest gift – the time spent with you gives me all the energy.

# Contents

# List of figures

# List of tables

# Abbreviations

| | |
|---|---|
| 3G | 3rd Generation |
| 3GPP | 3rd Generation Partnership Project |
| AMC | Adaptive Multi-hop Clustering |
| AODV | Ad hoc On-demand Distance Vector |
| AS | Application Server |
| CDF | Cumulative Distribution Function |
| CDS | Connected Dominating Set |
| DDCA | Distributed Dynamic Clustering Algorithm |
| DHT | Distributed Hash Table |
| DLBC | Degree-Load-Balancing Clustering |
| DNS | Domain Name System |
| DOM | Document Object Model |
| DSDV | Destination Sequence Distance Vector |
| DSR | Dynamic Source Routing |
| DVMRP | Distance Vector Multicast Routing Protocol |
| FTP | File Transfer Protocol |
| GGSN | Gateway GPRS Support Node |
| GPS | Global Positioning System |
| GSM | Global System for Mobile communications |
| GUI | Graphical User Interface |
| HSS | Home Subscriber Server |
| HTTP | Hypertext Transfer Protocol |
| ID | Identity |
| IETF | Internet Engineering Task Force |
| IMS | IP Multimedia Subsystem |
| ICE | Interactive Connectivity Establishment |
| IP | Internet Protocol |
| IPIC | Indirect Parallel Index Clusters |
| IPR | Intellectual Property Rights |
| J2ME | Java 2 Micro Edition |
| MAC | Medium Access Control |
| MANET | Mobile Ad hoc Network |
| MCDS | Minimum Connected Dominating Set |
| MIDP | Mobile Information Device Profile |
| MP3 | MPEG-1 Audio Layer 3 |
| MPEG | Moving Picture Expert Group |
| MSRP | Message Session Relay Protocol |
| MST | Minimum Spanning Tree |
| N/A | Not Available |
| NAT | Network Address Translator |
| OLSR | Optimized Link State Routing |
| OSPF | Open Shortest Path First |
| P2P | Peer-to-Peer |
| PDA | Personal Data Assistant |

| | |
|---|---|
| PIC | Parallel Index Clusters |
| PID | Proportional-Integral-Derivative |
| PIM-DM | Protocol Independent Multicast Dense Mode |
| PSC | Parallel Search Clusters |
| QIL | Query/Index Link |
| RIP | Routing Information Protocol |
| RSBRW | Replicating Stack-Based Random Walk |
| RSTP | Rapid Spanning Tree Protocol |
| SBRW | Stack-Based Random Walk |
| SDP | Session Description Protocol |
| SIL | Search/Index Link |
| SIP | Session Initiation Protocol |
| SLP | Service Location Protocol |
| SSDP | Simple Service Discovery Protocol |
| STP | Spanning Tree Protocol |
| TCP | Transmission Control Protocol |
| TTL | Time-To-Live |
| TURN | Traversal Using Relay NAT |
| UDP | User Datagram Protocol |
| UMTS | Universal Mobile Telecommunications System |
| UPnP | Universal Plug and Play |
| URI | Uniform Resource Identifier |
| VoIP | Voice over IP |
| WCDMA | Wideband Code Division Multiple Access |
| WLAN | Wireless Local Area Network |
| WWW | Word Wide Web |
| XML | eXtensible Markup Language |
| ZRP | Zone Routing Protocol |

# Chapter 1

# Introduction

## 1.1 Background

The Internet is built on the philosophy that the control and intelligence should be at the network edge instead of at the core [SRC84]. Currently, a similar shift toward the edge is seen in the information and media market. This shift concerns both the production of media and the distribution technology [KK08]. Whereas media was earlier generated by large organizations and published through a few distribution channels, the popularity of user-generated content has increased rapidly during recent years. The main enablers have been the improved technical possibilities for ordinary users to create and publish digital media [VW07]. Users have a large choice of distribution services, including YouTube [YouTube] for publishing video material, Flickr [Flickr] for publishing photos, myHeimat [myHeimat] for citizen journalism and justin.tv [JustinTV] for live broadcasts. In several popular services, such as Wikipedia [Wikipedia], the ordinary user is responsible for creating and maintaining the information and the service. Personal opinions and experience can be published through various blogging systems, bulletin boards and chat systems. Distribution of files and media has traditionally been implemented using servers in the network, using for example the File Transfer Protocol (FTP) and the World Wide Web (WWW). Distributed systems, such as peer-to-peer (P2P) file sharing systems and personal web-servers are currently rising as an interesting alternative to server-based systems. Moving the distribution system to the network edge allows saving costs by avoiding large centralized servers.

The trend of decentralization fits well with the currently popular Web 2.0 ideology. Web 2.0 [ORe05] refers to web services emphasizing user-created content, social networks, communities and collaboration. Facebook [Facebook], MySpace [MySpace], and LinkedIn [LinkedIn] are examples of popular services allowing users to network and keep in touch with other users. These online communities also support the distribution of user-generated media by controlling the access and focusing the attention on topics relevant to a given user.

Electronic devices are being equipped with an increasing amount of processing power, memory and communication capabilities at the same time as they become more lightweight and mobile. Today's mobile phones, Personal Data Assistants (PDAs), cameras, and music players have capacities comparable to those of computers a few years ago. In particular, mobile phones have been at the center of development, with increasing memory and processor capacity and a wide range of integrated applications and peripherals. Modern mobile phones are equipped with cameras, media players, radio, television and Global Positioning System (GPS) receivers. This broadens the use of the phone from simple voice communications to video communications, media sharing, collaboration, entertainment, and gaming. The phone is becoming a device for creating and consuming all types of digital media. The powerful capabilities and open software development platforms of mobile phones, and their wide range of communication networks, make these devices capable of participating in the current trend of user-generated media and social networking. As the phone is light and small, the user carries it along almost always. In addition to the applications known in the fixed network, mobility gives room for new innovative applications and services that were not meaningful in a fixed computer.

A variety of network technologies allow mobile devices to interconnect with other devices and with external services. Cellular networks, such as GSM (Global System for Mobile communications) and UMTS (Universal Mobile Telecommunications System) are by far the most popular for voice communications. Driven by the Internet, the share of packet data traffic on these networks increases. Cellular networks provide good mobility with wide coverage. They are based on a fixed infrastructure controlled by the operator. For local data communications, Wireless Local Area Networks (WLANs) are common. They typically connect to the public Internet through the private network of an organization. Complementary to the above technologies, recent research efforts have produced data networks operating between the devices without any fixed infrastructure. These so called wireless mesh networks, or *mobile ad hoc networks* (MANETs), are formed of mobile devices co-operating to forward packets where the radio range of a single sender is not sufficiently large to cover the destination. They are typically built on top of IEEE 802.11 [IEEE2007] or Bluetooth [Bluetooth] technology. Their lack of centralized elements leads to using decentralized algorithms for routing, resource discovery and other network management functions, making them inherently self-organizing.

As the amount of information increases, the importance of being able to locate the desired resources is emphasized. For web pages, search engines like Google [Google] provide a fast and convenient method for locating a page containing a specified keyword. Search engines, however, are not able to locate recently created pages, dynamically generated pages and pages with access limitations. Furthermore, searching is limited to keywords without support for more expressive semantics. Since the search is central to the operation of many web services, these often provide a separate search system to bypass the restrictions of search

engines. These, however, operate only locally. As the information moves to the network edge, there is a growing need for scalable methods to locate information in the terminals, in user communities and in social networks. In addition to information, there is a need to locate services, users, communities, events, and hardware resources.

Vanthournout et al. [VDB05] define a *resource* as "any source of supply, support, or aid that a component in a networked environment can readily draw upon when needed. Examples are: files, measurements, CPU cycles, memory, printing, control devices, forums, online shops, etc." Vanthournout et al. define *resource discovery* as the "ability to locate resources that comply to a set of requirements given in a query." They further define a *resource discovery service* as "the service that returns the location of matching resources in response to a query with requirements." The resource discovery service thus takes a query as input and returns a location. In this work we also use the terms *resource discovery system* as a synonym for resource discovery service.

Schwarz et al. [SEK+92] define *searching* as "an automated process, where the user provides some information about the resources being sought, and the system locates some appropriate matches." Thus, the search does not identify any specific resource and does not initiate the access to the resources. Schwarz et al. further define *browsing* as "the user-guided activity of exploring the contents of a resource space." The resource space may include all resources in the system but typically the resources are limited to the ones residing at a given location. In addition to searching and browsing, we can identify a third type of information retrieval. In a *lookup*, the identity of a resource is known and the information, including the location, is desired. As a result of searching, browsing or a lookup, a given resource may be selected for *access* by the user. Each type of resource may be accessed in a different way. For example, a file is downloaded using an indicated file transfer protocol.

Resource discovery is a central component in *peer-to-peer* (P2P) systems. Schollmeier [Sch01] defines peer-to-peer as "a distributed network architecture, where the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, etc.). These shared resources are necessary to provide the service and content offered by the network (e.g. file-sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (service and content) providers as well as resource (service and content) requesters (servent-concept)." Androutsellis-Theotokis and Spinellis [AS04] define peer-to-peer systems as "distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority." Roussopoulos et al. [RBR+04] characterize peer-to-peer systems as self-organizing systems with symmetric communication and decentralized control.

Hitherto peer-to-peer has been nearly synonymous with file sharing, with popular systems for sharing music and video material (e.g. Napster [Napster], Gnutella [Gnutella]) or documents (e.g. Freenet [Freenet]). It has been estimated that up to 70% [KBB+04] and 80% [FLM+03] of the Internet traffic is related to peer-to-peer file sharing, depending on the measurement point and how peer-to-peer traffic is identified. During night-time, the peer-to-peer traffic stands for nearly all of the total traffic [FLM+03]. Currently most of the activity in file sharing systems is illegal distribution of copyrighted files, such as music and movies, but legal applications are emerging. For example, the BitTorrent [BitTorrent] peer-to-peer protocol is used for the distribution of the Linux operating system [How03]. Commercial content providers are considering peer-to-peer systems as an efficient distribution method allowing cost savings and reducing the need for massive server farms as the users themselves contribute with the storage, bandwidth and processing power. Examples include Spotify [Spotify] for delivering streamed music and the P2P-Next [P2PNext] project aiming to provide large-scale delivery of video streams.

Peer-to-peer systems can be used in applications beyond file sharing. The peer-to-peer approach has been proposed as the underlying model for a wide variety of applications, from storage systems and cooperative content distribution to Web caching and communication infrastructures [QB06]. Seti@home [SETI] utilizes a peer-to-peer approach to combine the unused computer capacity of several users to analyze telescope data. Skype [Skype] utilizes peer-to-peer technology in order to locate users and bypass firewalls in the distributed implementation of Internet telephony. Whereas conventional Internet telephony, represented by the Session Initiation Protocol (SIP) [RSC+02] and H.323 [ITU98], has been based on servers, a distributed version of the SIP protocol, P2P-SIP [P2PSIP], is being developed within the Internet Engineering Task Force (IETF). Peer-to-peer technology can also be utilized to implement applications similar to the currently popular Web 2.0 services, represented by LinkedIn [LinkedIn], Facebook [Facebook], Flickr [Flickr], blogs, and chat and discussion groups. The inherent peer-to-peer type of interaction between users and the need for large servers in these applications make them good candidates for distribution. Roussopoulos et al. [RBR+04] categorize applications into the following general problem areas that are suggested as suitable for peer-to-peer: routing problems, backup, distributed monitoring, data sharing, data dissemination and auditing.

The popularity and interest in peer-to-peer systems can be explained by a few advantages. First, the load is distributed between the participants and individuals can distribute files practically without any costs. The storage space and bandwidth consumption is distributed between users eliminating the need for a server with massive capacity. This lowers the costs and risks in the introduction of a new service. For commercial providers, distribution offers lower requirements on server capacity and better scalability, which above all reduces the capital expenditure but also operation expenditure. For new providers, peer-to-peer systems lower the

entrance costs to the market. Moreover, the responsibility is distributed. No single service provider is responsible for hosting the material. It is difficult to track the participants, and there is no single point that can be shut down and held responsible, which has been an important driver especially in illegal file sharing. The system is also fault tolerant and provides a high degree of self-organization and load-balancing [YG02]. Traditional client-server solutions show their restrictions especially under peak usage [T08].

From the individual user's perspective the drawback is the required contribution and effort. Each peer-to-peer system uses some of the processing power, storage and bandwidth of the user. Participating in several peer-to-peer systems simultaneously multiplies these requirements. Each peer-to-peer system also requires an application to be installed, which is more cumbersome than using an application residing on a web server. These problems can be alleviated by implementing the peer-to-peer functionality as middleware, allowing a common resource discovery system to be shared between several applications.

A peer-to-peer file sharing application represents a typical application of resource discovery. The desired file is specified in a query that defines the attributes of the file. The most common attribute is the file name, but other important attributes include the file type, the description, the artist and the bitrate of a MP3 song, the codec of a video file, etc. Using peer-to-peer in a mobile environment gives new possible attributes, for example the location where a picture is taken. A search mechanism can be used to search for several other types of resources, such as a video streaming service or a chat group.

## 1.2 Research problem and objectives

The design of a resource discovery system is a tradeoff between several factors. During the last few years, most of the research on resource discovery has been concentrated on structured systems, i.e. systems with strict rules for the topology and location of information. In these, the requirements in terms of versatility have been lowered in favor of high efficiency. The main restriction of structured systems is that they are fundamentally restricted to *exact-match single-key* queries without any possibility to express complex queries [RM06]. Complex queries are required in several types of applications, where the desired resource is specified using multiple attributes, value ranges, numerical and string operations, substrings etc.

On the other hand, the high overhead of unstructured systems restricts their scalability, particularly in mobile networks. To improve scalability, the search scope is reduced in several systems. These systems are only able to locate a resource with a given probability and they are unable to guarantee finding a resource consequently. While such systems may be adequate for file sharing where the number of replicas is high and the harm of not detecting a file is low, reliable methods are needed to provide support for several other types of applications where resources may be unique or the number of replicas may be low.

Another common way to improve efficiency has been to increase the degree of centralization. This implies an unevenly distributed load, which may demotivate users from participating in the system. Our objective is to allocate the load according to the available capacity while maintaining a motivation to participate in the system. When all devices are similar, it is meaningful to allocate an equal load to each device. When some of the devices have a significantly higher capacity, and are maintained by a party willing to provide this capacity to the system, the aim is that these high-capacity devices bear a higher load. The load is primarily measured as the network traffic and storage.

In this work, we study distributed resource discovery systems with a special focus on mobile networks. We study approaches to designing a system that is efficient yet generic. The design requirements common to all architectures proposed in this work are:

- *Reliable and predictable operation*. If a requested resource exists in the system, then it must be found in each search.
- *Support for complex queries*. The application should be able to define any kind of query without customizing the overlay according to the query type.
- *Load distribution*. Load should be distributed uniformly between nodes or, where justifiable, according to device capacities.
- *Practically implementable*. The architecture must be feasible to implement in practice, considering the requirements on topology, maintenance, and usage. Nodes should be able to join and leave the system without causing large restructuring of the network.
- *Minimized traffic*. Considering all other requirements, the control traffic must be minimized.

In this work, we aim to address these properties by proposing a modeling framework and based on that framework we develop new architectures. The architectures developed in this work are intended for, but not limited to mobile networks. The mobile networks considered include cellular networks and ad hoc networks. These environments call for solutions that reduce the required bandwidth and energy consumption in the devices. The resource discovery is seen as a general service provided to a wide range of applications. Thus, the resource discovery can be implemented as middleware utilized by several applications running on the device. For the cellular network, we also aim to propose scenarios how resource discovery architectures can be utilized in the implementation of a resource sharing service – either provided by the cellular operator or running among the users themselves. We base the scenarios on the input from user studies. We further aim to evaluate the technical feasibility of such a service.

Although we recognize that security is important in all distributed systems, we do not attempt to analyze security threats or to develop solutions against malicious attacks, such as denial of service, as part of this work. We only consider the functional aspects of the resource discovery, such as hiding resources to which the requester has no access rights.

In our study, a resource can be any information or service that should be located. The resources may reside in any of the participating devices. We exclude systems relying heavily on a globally agreed classification of resources into different types as this may be impractical in open and heterogeneous scenarios. This eliminates systems where the nodes must be pre-classified according to their interest or where resources must be manually categorized into fixed categories. We do not consider how resources are specified, i.e. the format and content of the metadata. We aim to provide a general service that is not designed for any specific application. We exclude strictly structured systems. Finally, we also exclude work on the methods for accessing a located resource, as the method is specific to the type of resource.

## 1.3 Methodology

This work proposes several new algorithms and architectures. System and algorithm design is therefore the main method used in the work. The proposed solutions are tested and verified using various methods. For the technical evaluation, we use mathematical analysis, simulation and prototype implementations. The user study is based on questionnaires and interviews.

Mathematical analysis allows systems to be quickly and accurately studied. However, the model must often be simplified and generalized in order to allow analysis. In this work, mathematical analysis is used for determining parameters, for analyzing the expected performance of a system, and for explaining the experimental results.

Simulation allows a system to be implemented and tested in a controlled environment. The model of the system is more complex and realistic, and resembles the final implementation closely. However, the evaluation is exposed to the assumptions and approximation of the simulated external environment. All proposed architectures have been evaluated using simulation.

Implementation provides information about the actual performance of a system, usually using a simplified prototype. While prototyping gives real-world information, it is usually limited to tests with a few devices. This work uses prototype implementations to evaluate the performance of the mobile device and the wireless connections. Prototypes are also implemented to verify that the simulated architectures work in real networks and to detect shortcomings in the practical applicability.

Questionnaires and interviews provide information about how the user thinks and behaves. In this work, questionnaires and interviews have been used to aid the design of group-based resource discovery systems, to provide motivation for design choices and to provide input parameters. Large-scale user testing with implemented systems is, however, out of the scope of this work, which concentrates more on technical questions.

## 1.4 Our contribution

This thesis is written as a monograph. It is partially based on several publications authored or coauthored by the present author. The work presented in most of the publications has been extended. Additionally, this thesis contains material that has not been published earlier.

The main contributions of our work are the following.

1. A model for analyzing resource discovery systems. In particular, we propose a model that analyzes the balance between proactive and reactive operations.
2. Three new resource discovery architectures and the related algorithms. The first architecture extends the Parallel Index Clusters [CG03] architecture, allowing arbitrarily interconnected clusters. The second architecture minimizes the traffic by making the division between proactive and reactive operations optimal. This architecture is able to adapt to changing network conditions. The third architecture is fully proactive with a low traffic overhead.
3. Deployment scenarios and implementation considerations for resource discovery in a cellular network, both when the service is provided by an operator and when the service is maintained by the users in a distributed fashion. The contribution includes proposals to signaling schemes based on the Session Initiation Protocol.
4. Studies on the feasibility of resource discovery in a cellular network both from the user perspective and from a technical standpoint. The former is based on questionnaires and interviews, and the latter on prototype implementations.
5. Methods to organize the nodes in an ad-hoc network into clusters or into a backbone in order to allocate the largest load to the nodes with the highest capacity. The organization is the basis for a resource discovery system.
6. A simulator for evaluating the performance of overlay networks.

The original contributions of the present author can be found in this work and the following publications.

- The modeling work in Chapter 2 is mostly new unpublished work, except for the Search/Index Space model published in [Bei10]. This represents the sole work of the present author.
- The three architectures in Chapter 3 have been published in [Bei07a], [Bei10], and [Bei07b], respectively. These are the sole work of the present author.
- Results from one of the user studies of Chapter 4 have been published in [MBL+07] and [MBL+06b]. The operator-controlled architecture, the SIP-based signaling schemes and some measurements have been published in [BML+05], [MBL+06b], and [MGB+07]. The signaling has also been proposed for standardization in [GMB+06]. The fully distributed architecture has been published in [Bei07b], with the protocol specified in [Bei07c]. This work extends the published works and integrates the different approaches.

- The fundamental ideas for the categorization of devices and the creation of the virtual backbone in Chapter 5 are published in [CBK02], [CBK04], [CGK+04], [CKB05], and [CVK+06]. The algorithms for clustering the network according to capacity are published in [BKC05]. A shorter version of this publication has also been published as a book chapter [BKC06]. This work extends the published work.
- In addition to the publications directly related to resource discovery, the author previously participated in research on IP telephony and interoperability [KCB00], [KCB01], [Bei04]. These have given background on the methods in mapping, routing and distribution. The author has also worked on resource discovery in peer-to-peer SIP (P2P-SIP) [HHB09].

Consequently, part of the results of this work can be found in existing publications. This work, however, complements and extends the work in these publications. The work is structured as a monograph to give a more comprehensive view and a progressive approach to the research.

All simulations of overlays are performed using PONGsim [PONGsim], a simulation package developed by the present author. PONGsim is described in a technical report [Bei09]. Publication [BKC05] uses a predecessor of PONGsim for simulating ad hoc topologies.

Related work has been published in student work instructed by the present author. Parts of the prototype implementations have been presented in the Master's theses [Leh06] and [Rey07] as well as in the special assignments [Hyy06] and [Pan09]. The Master's thesis [Soi09] categorizes and proposes access control methods. Some of the surveys are described in the Bachelor's theses [Lag09], [Nor09], and [Pal09] and in the special assignment [Lev09].

We present the detailed contributions for each chapter separately in Sections 2.3, 3.3, 4.3, and 5.3.

## 1.5 Structure

The work studies resource discovery architectures and the applications of resource discovery in mobile networks. The work is therefore divided into a generic architectural part and a network-specific part. The purpose of the architectural part is to model and analyze resource discovery architectures and to propose new architectures. This allows us to compare algorithms and architectures in the general case without involving implementation specific details. The network-specific part applies resource discovery systems to cellular and ad hoc networks.

The rest of the work is divided into five chapters. Chapter 2 defines the terminology and concepts used in the thesis. It provides models for describing and analyzing resource discovery architectures. In particular it analyzes the optimal degree of index distribution, which is the basis for the rest of the work.

Chapter 3 presents three new types of architectures involving index distribution. For all architectures, we require that it should be possible to

locate all existing resources, and that the load is evenly distributed between nodes. We also aim to support a wide range of query types.

Chapter 4 studies the use of resource discovery in cellular networks. We evaluate the feasibility of peer-to-peer systems in cellular networks both from the user perspective and from a technical standpoint. We propose architectures for two different scenarios, depending on whether external centralized processing resources are available. We also design signaling schemes for the scenario.

Chapter 5 studies resource discovery in an ad hoc network formed between consumer devices. As the capabilities of the nodes are expected to vary significantly in such a network, we aim at allocating the load to the nodes with most capacity. The study focuses on forming an overlay according to the node capacities and utilizing this overlay to support resource discovery.

Chapter 6 presents the results, summarizes the work and suggests topics for future research.

# Chapter 2

# Modeling and analyzing indexing

This chapter presents techniques for modeling and analyzing indexing in a resource discovery system. First, the terminology and the concepts used in this work are defined and a technology background is provided. Then, we provide models for describing and analyzing resource discovery architectures. This includes the definition of the metrics and the desired properties as well as the modeling of overhead, update frequencies, and centralization. We present the simulator used in this work. We propose a model called the search/index space model for analyzing uniform indexing. In particular, we use this model to analyze the optimal degree of index distribution, and to determine when a reactive, proactive or hybrid system is optimal.

## 2.1 Introduction

In a collaborative system, *resource providers* make a selected set of their resources available to other users. A *resource* can be a piece of information or a service that the resource provider provides through his or her device. Information is usually in the form of a file, but equally well could be retrieved from a relational database. A service implements a function that a device provides for other devices, for example a printing service. In general, information has the characteristic that it is easy to replicate, but services are rather tied to a specific node. Information generated in real-time, such as a video stream, is viewed as a service. While the differences between information and services need to be considered in caching and replication, resource discovery operates in a similar way for all resources.

A resource is described, automatically or manually, by a set of *index terms* or *attributes*. Fundamental attributes include the type of resource and the name identifying the resource. The attributes form the resource's *metadata*. The metadata is used in the search to determine, both manually and automatically, whether a given resource meets the requirements of

the user. A *resource descriptor* includes access information in addition to the metadata of a resource. The *access information* is a set of attributes specifying the location and access methods of the resource, or it can be a *key* used in a subsequent lookup.

A *resource requester* is a user who intends to access a resource. Often, the resource requester does not specifically know the resource to access, but is able to specify the desired attributes of the resource using a *search criterion*. For example, a user wishing to access a printing service must at least specify the type of resource, a printer, for it to be found. The user might additionally specify other attributes, such as the type of printer, e.g. PostScript printer, or the color capability.

The objective of a resource discovery system is to be able to locate resources that reside in the devices of the participating users. The resource discovery process can be divided into a resource publication process performed by the resource provider, and a resource retrieval process performed by the resource requester. We identify four phases in the entire resource discovery process:

1. A resource provider *publishes* the resource descriptor in order to make the resource known to other devices.
2. A resource requester *searches* for resources matching a specified search criterion.
3. The resource requester *looks up* the devices providing the resources.
4. The resource requester contacts the resource provider in order to *access* a located resource.

The phases are sequential and connected to each other. Phases 1 and 2 are connected by the metadata that matches between the resource descriptor and the search criterion. Phases 2 and 3 are connected by a resource key, for example a hash of the resource name, uniquely identifying the resource in the system. Phases 3 and 4 are connected by the access information of the resource, for instance in the form of an IP address and a port number, or as a Uniform Resource Identifier (URI) [BFM98]. The phases are independent: a published resource may be found in several searches, a search may find several published resources, a resource may be available in several devices, and a located resource may be accessed several times and in different ways. The access is generally not considered as part of resource discovery – the resource discovery rather aims to provide the information to make access possible.

The resource retrieval process is depicted in Figure 2.1. Resource discovery systems can be divided into systems providing lookup and systems providing search. In the former, phase 2 is omitted and the user must know the identity (e.g. the unique name or key) of the resource through some external means. For example, in a structured system the user may need to search for the key with a separate web search engine, as in BitTorrent [BitTorrent]. In the latter, the user knows some properties of the desired resource. In those systems, phase 2 and 3 are commonly integrated into a single search phase. Unstructured systems allow this type of search. The search involves feedback: the user may re-specify the search criterion based on the obtained metadata, location and resources.

To enable feedback, the found information is presented to the user. The search process is successful if the user obtains the desired resource.

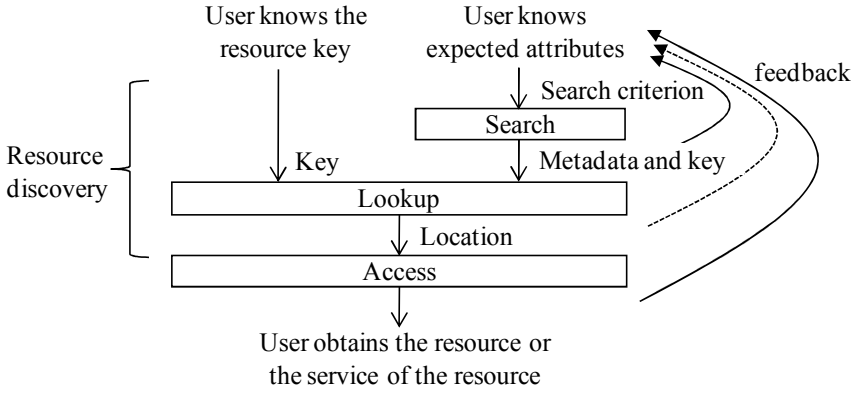


**Figure 2.1. The resource retrieval process.**

An *index* is the collection of resource descriptors of several resources. An *indexing architecture* is a resource discovery architecture, where the resource descriptors are distributed to nodes other than the resource provider. This chapter analyzes the use of indexing in resource discovery systems. Before going into the specific study of indexing, we present overlay networks, common topologies and basic distribution algorithms as a background to the analysis.

### 2.1.1 Overlay networks

An *overlay network* is a network built on top of another network. It is formed when a set of nodes, those running a given application, form a network by connecting to each other. The links between nodes in the overlay network are logical, and each overlay link can be thought of as a path of physical links. Nodes that are neighbors in the overlay network are not necessarily neighbors in the physical network. The underlying network is transparent from the overlay network's view and the traffic between two nodes in the overlay network is routed using the underlying routing protocol. Peer-to-peer systems are based on overlay networks. Other uses of overlay networks include adding structure to ad hoc networks and implementing application-layer multicast.

The overlay network can be modeled as a graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of vertices, or *nodes*, and $E = \{e_1, e_2, \ldots, e_M\}$ is the set of edges, or *links*, in the graph. The number of nodes in the network is denoted by $N$. Two nodes $v, u \in V$ are *neighbors* if a direct link $e = (v, u) \in E$ between them exists. The set of neighbors of a node $v$ is denoted $N(v)$. The number of neighbors of a node $v$ is the node's *degree*, $d(v) = |N(v)|$. Links are usually assumed bidirectional, but in order to generalize, we define a bidirectional link as two unidirectional links: one per direction. A node $v$ with unidirectional links may have an *indegree* $d_{in}(v)$ (the number on links ending at the node) that differs from the *outdegree* $d_{out}(v)$ (the number of links starting from the node)

[RW95]. To avoid ambiguity, we define that in a directed graph $N(v)$ only includes the nodes to which $v$ has a link, whereas $d(v) = d_{out}(v)$. The *distance* $dist(v, u)$ between two nodes $v$ and $u$ is the lowest number of links in the graph needed to connect the two nodes. The *diameter* of a network is the largest distance found between two nodes in the network.

In practice, a link in the overlay network is usually implemented as a Transmission Control Protocol (TCP) connection. When the User Datagram Protocol (UDP) is used instead of TCP, a link is only known at the application-layer through the knowledge about the peering node. Generally, each link has state information associated with it. Since an overlay link is purely conceptual, it is also possible to establish *temporary links* in order to transmit a packet directly between two nodes. The typical use of temporary links is to transmit a search reply directly to the resource requester without traversing intermediary nodes. For example, this method is used in the semi-recursive routing model in [JZM+08].

Since the topology of the overlay network is independent of and non-coinciding with the underlying network's topology, routing is often inefficient. Nodes that are neighbors in the overlay may be very distant physically, and a search operation may require a high number of round trips globally. The inefficiencies due to this mismatch between the overlay and physical topology have often been neglected as sufficient bandwidth has been available in the fixed Internet. The topology mismatch makes the effects of an inefficient resource discovery method, such as flooding, even more severe. The problem has been recognized and methods (e.g. [LZX+03], [LZX+04], and [WIH05]) have been developed for modifying the overlay topology to better match the physical topology. Especially in ad hoc networks the consideration of the physical topology is critical, and emphasized by the low bandwidth, high packet loss, changing physical topology, and the effect of collisions on the Medium Access Control (MAC) -layer. Overlays designed for ad-hoc networks often follow the physical topology to some degree.

### 2.1.2 Topologies

The connections between nodes in a network form the topology. The considered network can either be an overlay network or a physical network. We now describe some important topologies.

#### Random power-law topologies

Several types of natural networks, including social networks and computer networks, are shown to have a power-law distribution of node degrees [WS98]. A network whose degree distribution follows a power-law is referred to as a scale-free network [BA99]. In such a network, the probability $\Pi(d)$ that a node connects to $d$ nodes is $\Pi(d) \propto d^{-\gamma}$, where the exponent $\gamma$ characteristically is between 2 and 3 [BA99]. The power-law distribution implies a high number of low-degree nodes, and a small number of high-degree nodes.

A random topology is created by *unstructured systems*. As a node joins the overlay network, it creates links to a set of known nodes. These

nodes may be advertised through some external method (e.g. a www-page), build into the application, or remembered from previous sessions. The joining node may ask the known nodes for the addresses of other nodes to which it may connect. Consequently, the topology of the network forms rather randomly. The topology is easy to create and requires minimal maintenance. It has been demonstrated that the node degrees of most unstructured topologies follow a power-law distribution [SGG03], although not perfectly. The power-law property of such a topology is often self-enforcing: a node with a high degree has a higher probability of receiving new connections, i.e. further increasing its degree [BA99]. This preferential attachment mechanism is most often associated with scale-free networks, although it is only one of several mechanisms that can produce power-law networks [LAD+05].

Power-law topologies are demonstrated to be robust in the face of a random node breakdown [SGG03]. The network diameter of a power-law topology is low, improving routing efficiency – the highly-connected nodes essentially form a backbone which connects the nodes in the network with a small number of hops. The load on high-degree nodes is higher than the load of low-degree nodes, so ideally these nodes should be the most powerful and well-connected nodes. In most systems high-degree nodes appear quite randomly without any controlled selection. However, Gia [CRB+03] includes a method to adapt the topology so that high-capacity nodes have a higher degree and low-capacity nodes are within a few hops from a high-capacity node.

### Unit disk graphs

Another type of random topology is generated when nodes distributed in a two-dimensional plane interconnect with nodes within each other's radio coverage. Broadcast networks are often modeled as *unit disk graphs*, where every node is assumed to have an identical radio coverage with radius one. A node $v$ has a link to a node $u$ if and only if the Euclidean distance between $v$ and $u$ is at most 1. This model is slightly over-simplified as the transmission power of practical devices often varies. Neither does it consider obstacles and interference [KWZ08]. Regardless of these shortcomings, the model is frequently used to model ad hoc networks [KMW04].

### Structured topologies

The above topologies are formed rather randomly or depending on the physical restrictions. In an overlay network, any kind of topology can be constructed in a controlled way. Well-known topologies in networking include rings, stars, trees and fully connected topologies. In these topologies it is easy and efficient to send a message to all nodes. These can also successfully be used in overlay networks. The difference to random topologies is that they require continuous maintenance as nodes join and leave. Structured topologies, together with strict data placement and actively maintained routing information, are used in a type of peer-to-peer systems called *structured systems*, described more thoroughly in Section 3.2.

### 2.1.3 Distribution algorithms in random topologies

Delivering a message to all (or a defined set of) nodes in a random (or arbitrary) topology is a common problem with applications in, for instance, routing and multicasting. Two basic search algorithms are commonly used in random topologies: flooding and random walk. Both algorithms allow for several variations.

#### Flooding

Flooding is a simple and popular method to deliver a message to all nodes in a network using a breadth-first search. Typical applications for flooding include the delivery of routing updates in link state routing protocols, including OSPF [Moy94], and locating the destination node in reactive routing protocols for ad hoc networks, including AODV [PBD03] and DSR [JHM07]. In this work, we are concerned with using flooding as a search algorithm whereas the flooded message is a search request. Most popular unstructured systems, including Gnutella [Gnutella] and Kazaa [Kazaa], use flooding for distributing search requests. We call an architecture using global flooding of search requests in a random topology a *search flooding architecture*[1].

In flooding, the resource requester $v$ sends a search request message to all its neighbors simultaneously. Each node receiving the message processes the message (e.g. by looking for a requested resource in its index), and then forwards a copy of the message to all neighbors, except the one from which it was received. In order to avoid forwarding the same message several times, each message is identified with a unique request identity (ID). A node remembers the IDs of the forwarded messages, and if the same ID is seen in a received message then the received message is discarded before processing and forwarding. Flooding is a solid search method: if there is a path from the resource requester to a given node, then the request will be delivered. The request will reach a given node on the fastest (often shortest) path. However, even though the algorithm guarantees that a message will not be forwarded twice, it cannot guarantee that the message is not received twice. In fact, the more neighbors a node has the more copies of the message will it receive.

In order to reduce the traffic, the maximum number of hops is usually limited with a time-to-live (TTL) field $T_{TTL}$ that is initialized to a given value $T_{InitialTTL}$ and decreased each time the message is forwarded. When $T_{TTL} = 0$, the message is discarded. For example, Gnutella limits the scope of a search request to a maximum of $T_{InitialTTL} = 7$ hops [RFI02]. The disadvantage of this method is that it cannot find resources on a node $u$ for which $D(v, u) > T_{InitialTTL}$. Fortunately, resources are typically well replicated in file sharing applications, so a copy can, with reasonable probability, be found within the hop limit. Rare resources may still be undetected.

[1] For clarity we avoid the more common but ambiguous term *fully distributed architecture* as several other architectures can be classified as fully distributed.

Iterative deepening [YG02] [BCF+03] (also called expanding ring search or adaptive TTL) is a well-known technique where $T_{InitialTTL}$ is successively increased until a match has been found or until $T_{InitialTTL}$ exceeds a maximum value. The resource requester must wait a defined period and observe potential replies before repeating the search with an increased $T_{InitialTTL}$. This increases the search delay but allows all existing resources to be found. For resources near the resource requester, the delay and message overhead is low. A variant of iterative deepening is Hurricane Flooding [JJ07], where the resource requester divides its neighbors into groups, and upon each iteration sends the message to one group with a $T_{InitialTTL}$ higher than in the previous iteration. In this algorithm, the network is searched in a spiral pattern centered in the resource requester.

Flooding has also been extended by forwarding the query only to selected neighbors chosen randomly, such as in Modified Random BFS [KGZ02] and Normalized Flooding [GMS05], or according to some heuristics, such as the number of past results in Directed BFS [YG02], similarity of queries in Intelligent Search [KGZ02], or similarity of neighbors in [CDN+05]. None of these approaches are able to guarantee finding an existing resource.

### Random walk

Random walk is a message distribution algorithm that, like flooding, can be used to distribute a search request in any topology. In random walk, the resource requester generates one or several *walkers*, with $T_{TTL}$ initialized to $T_{InitialTTL}$. A walker is in practice a search message that is sent to a randomly chosen neighbor. Each node that receives the walker checks its index for matches and forwards the walker to a randomly chosen neighbor with a reduced $T_{TTL}$, excluding the neighbor from which the walker was received. The difference to flooding is thus that the search is sent only to one neighbor instead of to all. The search can be stopped when one or a given number of matches have been found. Otherwise, the walker traverses the network until $T_{TTL} = 0$. The walker may return to an already visited node. It has been shown that a walker gravitates toward high-degree nodes in a power-law network [ALP+01].

One major weakness of random walking is the long delay. As nodes are visited sequentially, one node at a time, the maximum delay is proportional to $T_{InitialTTL}$. To reduce the delay, several parallel walkers can be generated by the resource requester, but this reduces the efficiency as the walkers are likely to visit the same nodes. The question how many walkers should be generated is difficult as it depends on the network size and node connectivity, both of which are unknown by the resource requester.

Walkers can also be replicated during their walk. For example, a walker can be split into two walkers each time it is forwarded. This allows random walk to adapt to networks of different size: the longer the path the more walkers are generated. Overlapping between different walkers is still a major problem. The method can be seen as an intermediary between pure random walks and flooding, i.e. flooding with

a limited number of considered neighbors. In [ZLZ+05] the walkers avoid overlapping paths by storing the visited nodes in a Bloom filter. However, after the walker has been replicated, the new walkers do not communicate and the Bloom filter contains only the nodes visited by the other walker before replication.

Random walks have been applied to unstructured peer-to-peer networks in several variations. In [ALP+01] the walker is sent to the highest-degree neighbor, utilizing the power-law property of the overlay. Freenet [Freenet] uses hints to guide a walker towards nodes with higher probability of having the requested resource. Gia [CRB+03] selects the highest capacity neighbor for which there is an available token. Gia nodes further remember queries, so that when the same query returns to a node, it is forwarded to a different node than earlier. Matches reported to the resource requester serve as keep-alive messages, indicating that the query has not been lost due to topology changes.

## 2.2 Related research

The scalability of search flooding networks has been analyzed in several publications. Ripaeanu and Foster [RF02] analyze the Gnutella network using packet traces and determine the degree distributions and path lengths in Gnutella. Saroiu et al. [SGG02] use a similar method to obtain comprehensive statistics about traffic, sessions and shared resources in Gnutella. Ritter [Rit01] models the peer-to-peer traffic patterns using a tree and concludes that the traffic grows exponentially with the number of hops, which causes scalability problems. Schollmeier and Schollmeier [SS02] show that this exponential model is too simplified and unrealistically pessimistic, partly because it does not consider the finite network size.

As only a few architectures utilize large-scale index distribution (see Chapter 3), the number of works modeling and analyzing such architectures is low.

Cooper and Garcia-Molina propose the Search/Index Link (SIL) model [CG03] to analyze architectures involving indexing. In this model, links are divided into *non-forwarding search links*, *forwarding search links*, *non-forwarding index links*, and *forwarding index links*. Forwarding links implement flooding while non-forwarding links implement sending over a single hop. A search message is sent on all search links starting from the resource requester. A search message received over a forwarding search link is forwarded on all search links (except for the link on which the message was received), provided that this search message has not been forwarded earlier. On a non-forwarding search link the message is only sent over a single hop and not forwarded by the receiving node. Copies of a node's local index are distributed correspondingly over forwarding and non-forwarding index links, and stored in the remote indices of the nodes along the path. Links are unidirectional.

The SIL-model is enhanced by Deng and Lv into a Query/Index Link (QIL) model [DL04], with only two types of links: *search links* and *index*

*links*. QIL adds an optional parameter to the link that specifies the $T_{InitialTTL}$ of a message whose first hop is over this link. This replaces the concept of forwarding and non-forwarding links: for a forwarding link $T_{InitialTTL} = \infty$ and for a non-forwarding link $T_{InitialTTL} = 1$. Graphically search links are marked with solid arrows and index links with dashed arrows. Figure 2.2 shows a simple example of a network modeled with QIL: nodes A and B maintain an index of all resources in the network and other nodes can search by querying one of these nodes.

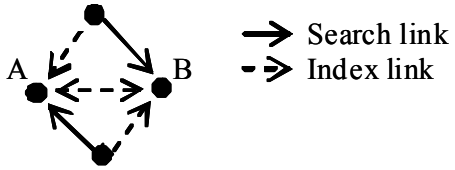


**Figure 2.2. Example network modeled with QIL.**

The models allow coverage to be analyzed. The *coverage* of node *v* is defined in [CG03] as the fraction of nodes in the network that can be searched by a query generated by *v*. In a network with *full coverage* all nodes have a coverage of one. The models can also be used to analyze redundancy. A network has *redundancy* if it contains a link that can be removed without reducing the coverage for any node [CG03].

Both the SIL and QIL models are able to describe most of the existing unstructured architectures. Unfortunately, the models only consider flooding and single-hop updates as search and index distribution algorithms. Architectures utilizing random walks and other less common distribution methods cannot be described. Despite this limitation, the division into search and index links can be used to model a wider range of architectures, provided that the used search and index distribution algorithms (if not flooding) are stated explicitly. In this work, we use the QIL model to visualize the overlay topology.

Three index replication strategies are compared in [CS02], which shows that a square root allocation minimizes the number of queried nodes. A square root allocation refers to an allocation where the number of copies of a resource is proportional to the square root of the search frequency for this particular resource. The work assumes a search process that can pick unvisited random nodes regardless of the topology until the resource is found. As the work does not consider how search and replication is implemented, it can be considered more of a theoretical framework. Neither does it consider the message overhead for replication and searching.

## 2.3 Our contribution

In this chapter, we provide methods for modeling and analyzing resource discovery systems and the use of indexing in these. Of particular interest are such resource discovery systems where centralized components are avoided and the load is distributed evenly among the participants.

In Section 2.4 we define the central concepts and operations in a resource discovery system. We define four important properties: determinism, complex queries, index allocation invariance and uniformity. We specify the parameters modeling a resource discovery system and the metrics used to evaluate the performance of a system. We discuss the overhead of common distribution algorithms and examine the relation between the node degree and the overhead of flooding. We study when and how frequently index updates should be generated. We evaluate the use of indexing in the basic peer-to-peer architectures and the performance implications of centralization. The material is unpublished and produced as a reference framework for the rest of this work. We also present a simulator developed by the present author. The simulator has been published as open source [PONGsim] and is described in a technical report [Bei09]. The simulator has been used in other projects, including [Soi10].

In Section 2.4.9 we construct a model called the Search/Index Space for examining the optimal balance between reactive and proactive operations in a uniform architecture. We apply the model to analyze how the ratio between the frequency of index updates and the frequency of search requests determines how reactive and proactive operations are optimally combined. In particular, we evaluate when a fully proactive or reactive solution is optimal. The model is published in [Bei10] and complemented in this work to consider the message size. We further apply the model to determine the optimal cluster size in the Parallel Index Cluster and Parallel Search Cluster architectures.

This chapter and the related publications are the sole work of the present author.

## 2.4 Modeling indexing

In order to be able to analyze and describe the properties of resource discovery systems, we need to model them. In this section, we provide a model used in the rest of the work.

### 2.4.1 Terms and definitions

Let us first define some fundamental concepts. Each resource discovery system studied in this work is based on an *overlay network* $G = (V, E)$. The overlay network is formed by a set of *nodes* $V$ and a set of logical *links* $E$ that connect these nodes. The arrangement of links between the nodes forms the *topology* of the overlay network. The links are used for transmission of *messages*. There are four main types of messages: *index updates*, *search requests*, *search replies*, and *maintenance messages*. Index updates and search requests are typically *one-to-many* distributions while the search reply is a *one-to-one* transmission. Some architectures send the reply message using a temporary link instead of using the fixed links of the overlay network. A *temporary link* connects two nodes for the duration of a single message transmission. Maintenance messages may be required in a topology with a defined structure.

A new message is *generated* by a node. The generated message is then *sent* to a set of neighboring nodes. Assuming error-free transmission, each of these nodes *receives* the message. A node may *forward* a received message, whereas the same message or a modified version of the message is *sent* to a set of neighboring nodes. Thus, the term sent message includes both generated and forwarded messages. Each node that either generates or receives a message is said to *process* the message. Unambiguous definitions of these terms are especially important in performance evaluation as these events are counted.

A subset $\mathbb{Z}$ of the nodes are *resource providers*. When a resource provider makes a resource available, it creates a *resource descriptor* containing the resource provider's address and the *metadata* describing the resource. The collection of resource descriptors of node $v$ forms the *local index* $\mathcal{L}_v$, which describes the resources available at node $v$. Certain architectures, called *indexing architectures*, distribute the local index of a node $v$, either explicitly or implicitly, to a set of *indexing nodes*, $\mathbb{P}_v$. Most indexing architectures distribute resource descriptors over a single hop, whereas $|\mathbb{P}_v| \leq 1$. We call those *non-forwarding indexing architectures*. In a *forwarding indexing architecture*, the resource descriptor can be forwarded multiple times.

The local index is transported in one or several *index updates* to the indexing nodes. The *index distribution algorithm* defines the local decisions made by a node in order to generate and forward index updates. Consequently the indexing nodes $\mathbb{P}_v$ are determined by the index distribution algorithm together with the topology. Each node $u \in \mathbb{P}_v$ stores the resource descriptors of a received index update in its remote index $\mathcal{R}_u$. A node $u$ may therefore, in addition to its local index, maintain a *remote index* $\mathcal{R}_u = \{\mathcal{L}_w \mid \forall\, w \in \mathbb{N}_u\}$ describing resources available at a set of *indexed nodes* $\mathbb{N}_u$. The *index* $\mathcal{J}_u = \mathcal{L}_u \cup \mathcal{R}_u$ of a node $u$ consequently contains the entries of the local index and the potential remote index. The resource descriptors are maintained in the remote index until they are either replaced by new versions of the same resource descriptors or explicitly removed. Additionally, most indexing architectures remove descriptors that have not been renewed within a given expiration time.

A *resource requester* $v$ that desires to access a resource formulates a *search criterion*. The search criterion is transported in a *search request* or *query* to a set of *queried nodes* $\mathbb{R}_v$ according to the local decisions specified by the *search algorithm*. The query covers the index of a set of *covered nodes* $\mathbb{H}_v = \cup_{u \in \mathbb{R}_v} \mathbb{N}_u$. When the search request encounters a node whose index contains a resource descriptor that satisfies the search criterion, a *match* has been found. A *search reply* is then generated and sent to the resource requester. The search reply may be transported to the resource requester over a single logical hop or by traversing the reverse path of the search request.

We call the distribution of a search request or index update to a set of node a *transaction*. Borrowing the terminology from ad hoc networks [Fee99], we separate between proactive and reactive transactions. The

distribution of index information to $\mathbb{P}$ is a *proactive* transaction while the distribution of a search to $\mathbb{R}$ is a *reactive* transaction.

The rules for forming the topology, the search and index distribution algorithms and the other fundamental properties of the overlay together constitute the *architecture* of a resource discovery system.

### 2.4.2 Fundamental properties of a resource discovery architecture

The choice of architecture is largely determined by the properties that are required from it. In this section we define the fundamental properties that we aim to support in most of our proposed architectures.

#### Determinism

Some resource discovery systems cannot guarantee a resource to be found in all searches and by all nodes. Instead, the success is determined probabilistically and the results may vary between consecutive searches. Jin and Jiang [JJ07] define *deterministic* search strategies as strategies guaranteeing that at least one copy of the resource is found, provided that the resource exists in the system. *Non-deterministic* strategies do not provide such guarantees. Non-determinism may be a result of the use of randomness, such as forwarding a search request to only a few randomly chosen neighbors. It can also be due to a limited search scope, for example forwarding a search request only to the few closest neighboring nodes. In these cases, the system is non-deterministic by design, because even in the lack of external forces the system may not find all matching resources. Repeating a search request multiple times in such a system will not guarantee finding the resources.

A different kind of failure is caused by external forces, such as message losses or node failures. Because these are independent of the system design, we do not take them into account in our classification into deterministic and non-deterministic systems. As these are temporary in nature, they can be solved by repeating the search. We therefore refine the term deterministic system to consider the behavior only in static situations. Specifically, in this work, we define a deterministic resource discovery system in the following way.

**Definition 2.1.** An architecture is *deterministic* if each existing resource, after a bounded time after its publication, is found in a bounded time in each search performed by each node in a network that is static for the duration of the search operation.

This definition allows a bounded time for index distribution and search distribution. As a consequence of the definition, a system is considered deterministic even if a search fails due to a topology change during the short search transaction. In any architecture, a search may fail due to a topology change, no matter how well-designed the architecture is. The purpose of the definition is to identify architectures that are deterministic in the absence of external forces. Unless otherwise stated, we use the above definition for the term deterministic.

The concept of determinism is related to the concept of *full coverage* defined in [CG03] for networks modeled with SIL. The coverage of a node *v* is defined as the fraction of nodes in the network that can be searched, either directly or indirectly, by a query generated by a node *v*. A network has full coverage if the coverage by every node is one. Networks with full coverage are deterministic when flooding is the distribution method (which is assumed in SIL). When another search method, e.g. random walk, is used in a system with full coverage, the system is not necessarily deterministic.

### Complex queries

Resource discovery systems may allow different degrees of complexity in defining the search criterion. In the simplest case, each resource is identified by one or several keys (e.g. keywords or names). A user locates the resource by specifying one of the keys as a search criterion. The key must be specified exactly in the same way in the query as in the resource publication and only a single key is allowed in a query. Systems based on distributed hash tables (DHT), as described later, are fundamentally limited to this type of *exact-match single-key* queries [RM06]. These systems implement a mapping from a key to a value (typically the address of the resource).

In practice, users tend to have partial information about the desired resource and they submit broad queries [GFB+04]. Many applications therefore need to support *complex queries*. Especially substring searching is popular. For example, a user entering the search word “travel” might want to obtain resources containing the words “travel”, “travels”, “travel’s”, “travelling” (British), “traveling” (American), “traveller” (British), “traveler” (American), as well as compounds such as “travel tips”. Combinations of several keywords are common: Reynolds and Vahdat [RV03] report that 71.5 % of the queries sent to a Web search engine contain two or more keywords and over 40% of the queries contain three or more keywords.

In [TP03] the following types of complex queries are identified:

1. *Multi-attribute queries* define desired values for several attributes and the attributes are combined using a logical operation. Typically, the logical *and* is used and the query matches if each attribute matches with the required value. An example multi-attribute query locates a device providing printing services *and* providing color prints.
2. *Range queries* define a range of values allowed for a given attribute. Example range queries locate files larger than a given size, music encoded with given bitrates, or services within given geographical coordinates.
3. *Aggregation queries* combine results from a large number of nodes. Examples of aggregation queries are Count, Sum, Maximum, Minimum, Average, Median and Top-K [RM06].
4. *Join queries* combine records from two tables in a relational database.

We complement the above list with evaluated queries:

5. *Evaluated queries* use an expression to evaluate matches. For example, the desired resource can be defined using a regular expression.

The class of evaluated queries has two important subclasses:

6. *Substring queries* produce matches with string attributes containing a specified substring. Substring queries are popular in file sharing applications, where the user can search for files containing a given substring in their file name.
7. *Similarity queries* produce matches with string attributes similar to a specified search string according to some definition. Such queries can match with words that are in a different form (e.g. singular vs. plural), with words that sound similar, or words containing minor spelling mistakes.

Techniques for supporting certain types of complex queries using multiple exact-match single-key queries exist but these currently have severe limitations (described in Section 3.2.2). Above all, each technique supports only a few types of queries and often *a priori* knowledge of the expected content is required.

On the other hand, a generic complex query can be based on any attribute, any combination of the attributes and any part of an attribute. Therefore, to support all types of complex queries without *a priori* knowledge, the whole resource descriptor must be available for matching with the search criterion. Searching therefore requires each search request to be compared with the index of each resource provider in the system. We define the requirement for complex queries in the following way.

**Definition 2.2.** An architecture *supports complex queries* if $\mathbb{H}_v = \mathbb{Z}$ for every node $v \in V$, where $\mathbb{Z}$ are the resource providers and $\mathbb{H}_v$ are the nodes whose indices are covered by searches of $v$.

### Index allocation invariance

A system must reduce stale resource descriptors, i.e. resource descriptions containing aged information or referring to a non-existent resource, as they consume storage, cause false matches and give the user incorrect information. If several versions of an entry exist in the system, a querying node is not able to determine whether an obtained resource descriptor is the most recent one unless the search is guaranteed to cover all indices in the system. A system allowing a high share of stale descriptors must explicitly check the validity of the resource descriptor on resource access. Still it may not be deterministic as it may miss potential matches. To reduce stale resource descriptors, the validity time of the index must be short and the index often refreshed. This is not enough to guarantee determinism.

In order to avoid stale resource descriptors, the set of indexing nodes $\mathbb{P}_v$ must be invariant: consecutive updates of a resource descriptor must reach the same indexing nodes. This ensures that all remote indices are correctly updated with a modified or removed resource descriptor. We

define the requirement of index allocation invariance in the following way.

**Definition 2.3.** An architecture is *index allocation invariant* if the set of indexing nodes $\mathbb{P}_v$ and $\mathbb{P}_v$' in two consecutive index updates performed by a node $v$ always are the same (i.e. $\mathbb{P}_v = \mathbb{P}_v$') in a static network. If $n_{join}$ nodes join and $n_{leave}$ nodes leave between two consecutive updates, the number of nodes changed in the set of indexing nodes must be $|\mathbb{P}_v \Delta \mathbb{P}_v'| \leq n_{leave} + n_{join}$, where $\Delta$ denotes symmetric difference.

#### Uniformity

Borrowing from the terminology of ad hoc networking [Fee99], we define uniform architectures in the following way.

**Definition 2.4.** An architecture is *uniform* if there is no distinction in the roles of nodes.

In uniform indexing architectures, all nodes participate in indexing. As a consequence of the lack of specific roles, the indexing load is typically distributed equally among the nodes. A stronger definition requires that the number of indexed nodes $|\mathbb{N}_v|$ is approximately similar for each node $v$ in the network.

**Definition 2.5.** An architecture has *uniform index allocation* if the number of indexed nodes $|\mathbb{N}_v|$ is similar (within reasonable limits) for each node $v$ in the overlay network.

### 2.4.3 Network model parameters

The performance of a resource discovery architecture is dependent on several input parameters that describe the modeled network and usage. We model the network using a selected set of parameters that contribute significantly to the performance and that can be measured. These input parameters are expressed as network-wide averages. The central parameters used in this work are listed in Table 2.1.

**Table 2.1. Input parameters.**

| Parameter | Symbol | Description |
|---|---|---|
| Network size | $N$ | The number of nodes in the system. |
| Degree | $D$ | The average node degree. |
| Frequency of generated search messages | $f_s$ | The average number of generated search messages per node per time unit. |
| Frequency of generated index messages | $f_i$ | The average number of generated index update messages per node per time unit. |
| Frequency of generated messages | $f$ | $f = f_s + f_i$. |
| Search/index ratio | $r$ | $r = f_s / f_i$. |
| Churn frequency | $f_{churn}$ | The average number of nodes joining the network per time unit. As we assume that the long-term network size is constant, this is also the average number of nodes leaving per time unit. |
| Replication | $R_{replication}$ | The percentage of nodes that have identical copies of a resource. Each resource is assumed to be replicated to the same number of nodes. |
| Size of a search message | $S_s$ | The average size of a search message. |
| Size of an index message | $S_i$ | The average size of an index update message. The size varies largely as it depends on the number of resources contained in the message and the level of detail in the descriptions. |

### 2.4.4 Performance metrics

A performance metric is used to evaluate the performance of a system as a function of the input parameters. We divide the metrics into traffic metrics, quality metrics and transaction metrics.

#### Traffic metrics

Traffic metrics describe the frequency of events. Events include generating, sending, forwarding, and receiving messages. The events can be measured either as a count of messages or as a count of bytes per time unit. Message types related to resource discovery include search requests, search replies, index updates, and overlay maintenance messages. Furthermore, the traffic can be measured on a per-node basis or as a network-wide average. Although a simulator typically produces all these metrics, we need to select a subset for evaluating the performance.

As we aim to evaluate the algorithms and architectures independently of the implementation details and the protocol design, we measure the

message frequency instead of the bandwidth. An estimate of the bandwidth can then be calculated by multiplying the message frequency with the average packet size, once the packet size is determined. The packet size depends on the packet format, data encoding, protocol overhead and the number of included resources. For example, a significantly higher packet size results from transporting data in the text-based SIP protocol than using a binary protocol. Multiplying the message frequency with the message size only gives an estimate of the bandwidth, as the size of different packet types can vary, but within a given type of transaction (search or index update) the variation is relatively small.

Assuming error-free transmission, the network-wide number of sent messages equals the network-wide number of received messages. Therefore, it is irrelevant which metric is used for measuring the network-wide traffic. From the view of a single node, the numbers of received and sent messages differ. We prefer counting the received messages rather than sent messages, as it gives a simple load indicator. The motivation is that each received message has to be processed separately (e.g. with an index lookup), while sending a message (e.g. with flooding) duplicates the same message to several neighbors.

In our analysis we ignore the messaging for reporting the search results. The number of reports depends on the number of matches and the size of a report depends on the number of matches on a given node. These are unrelated to the performance of a search or index distribution algorithm. In most cases, as we study unstructured systems, there are no overlay maintenance messages.

When not explicitly indicated, the used traffic metrics are network-wide averages. However, in some cases we present the node perspective, which represents the performance as experienced by the end user. Especially in non-uniform indexing architectures, nodes are typically assigned different roles and the performance experienced by different nodes varies widely. The performance can then be evaluated as averages for each role separately. In examining the degree of uniformity, the maximum, the minimum and the variation of a metric are interesting in addition to the average.

The used traffic metrics are defined in Table 2.2.

Battery consumption is not considered separately in this work, but the consumption resulting from network access is assumed to be proportional to the sum $F + f$.

**Table 2.2. Traffic metrics.**

| Parameter | Symbol | Description |
|---|---|---|
| Frequency of received search messages | $F_s$ | The number of search messages received by a node per time unit. |
| Frequency of received index messages | $F_i$ | The number of index messages received by a node per time unit. |
| Frequency of received index messages | $F_m$ | The number of maintenance messages received by a node per time unit. |
| Frequency of received messages | $F$ | The total number of messages received by a node per time unit. |
| Search load | $L_s$ | The number of search messages processed (generated or received) by a node per time unit. |
| Index load | $L_i$ | The number of index messages processed (generated or received) by a node per time unit. |
| Maintenance load | $L_m$ | The number of maintenance messages processed (generated or received) by a node per time unit. |
| Load | $L$ | The total number of messages processed (generated or received) by a node per time unit. |

As both load and message frequencies are commonly used in literature, we include both as metrics. We define the index load and search load based on the definitions in [CG06].

**Definition 2.6.** The search load is the average number of search messages processed by a node per time unit. This includes messages originating from the node itself, corresponding to searching the local index:

$$L_s = F_s + f_s . \tag{2.1}$$

**Definition 2.7.** The index load is the average number of index messages processed by a node per time unit. The index load is the sum of received index messages and generated index messages:

$$L_i = F_i + f_i . \tag{2.2}$$

As load can be derived from the frequency of generated messages and the frequency of received messages we do not indicate it separately.

### Transaction metrics

Transaction metrics describe the cost and the scope of a single transaction for a given distribution algorithm. A transaction is either a search distribution or an index distribution. Network-wide averages are used to

reduce the dependence of a given node's location in the network. The used transaction metrics are defined in Table 2.3.

**Table 2.3. Transaction metrics.**

| Parameter | Symbol | Description |
|---|---|---|
| Search message count | $M_s$ | Number of message transmissions in the network resulting from a single search operation, provided that all matching resources are found. |
| Index message count | $M_i$ | Number of message transmissions in the network resulting from a single index update. |
| Search visit count | $N_s$ | Number of nodes (other than the requester) contacted in a search operation. |
| Index visit count | $N_i$ | Number of nodes (other than the resource provider) storing the index of a node. |

### Quality metrics

Quality metrics describe the satisfaction of the user. The quality metrics used in this work are presented in Table 2.4.

**Table 2.4. Quality metrics.**

| Parameter | Symbol | Description |
|---|---|---|
| Search delay | $T_s$ | The average time to discover a unique resource located at a random node in the network. This delay is the typical latency experienced by the user. |
| Full search delay | $\widehat{T}_s$ | The time to search all indices in the system. This is also the maximum delay to find a given resource located at a random node. |
| Full index distribution delay | $\widehat{T}_i$ | The time to distribute an index update to all indexing nodes. This delay determines the time from a resource being published to that the resource can be found in any search. |
| Success ratio | $R_{success}$ | The ratio of the number of found matches to the number of existing matching resources. A ratio below one implies that the user cannot be satisfied on all searches. |

If the success ratio is below one, the user cannot be satisfied in all searches. As this work studies resource discovery algorithms that are deterministic according to Definition 2.1, we require a success ratio of $R_{success} = 1$ in a static system. In real systems both resources and nodes are dynamically added and removed. Even if the system is deterministic according to our definition, a search may fail if the resource is removed

during the search distribution ($T_s > 0$) or if the search is performed before the index distribution is completed ($T_i > 0$). We require a measured success ratio over 99%.

The delay is affected by the packet size, the underlying topology and the link delays. To give a more generic view of the performance, free from these assumptions, we measure the delay with a fixed link delay $T_{link}$ independent of the packet size. The delay can be linearly scaled to take other link delays and an average message size into account.

The index distribution delay has only a marginal effect on the perceived performance, since it is unlikely that a newly published resource is immediately requested. Generally it is possible to use slower methods for index distribution than for search distribution. Nevertheless, the index distribution delay must be within reasonable limits not to affect the success ratio. While an excessive search delay only lowers the user satisfaction, an excessive index update delay may cause searches for the resource to fail even though the resource exists in the system.

### 2.4.5 Overhead

We define the central concepts of overhead of a distribution algorithm in the following way.

**Definition 2.8.** The *search overhead* of a search algorithm is defined as $\Omega_s = M_s / N_s$.

**Definition 2.9.** The *index distribution overhead* of an index distribution algorithm is defined as $\Omega_i = M_i / N_i$.

In an optimal distribution algorithm, each node receives a distributed message only once. Delivering a search message to $N_s$ receivers can therefore optimally be performed with $M_s = N_s$ transmissions, i.e. $\Omega_s = 1$.

**Definition 2.10.** A search distribution algorithm is *optimal* if $\Omega_s = 1$. An index distribution algorithm is *optimal* if $\Omega_i = 1$.

The traffic metrics and transaction metrics are related. Let us show the relationship between metrics by considering a single type of message: search messages. Each node generates search messages at an average frequency $f_s$. As each search message is distributed to $M_s$ nodes, and no message is assumed to be lost, each node $v$ give rise to

$$F_{s,v} = f_s M_s \tag{2.3}$$

receptions of search messages. As the network contains $N$ nodes, each producing search messages at the frequency of $f_s$, the total number of search message receptions in the network is

$$F_{s,network} = \sum_{v=1..N} F_{s,v} = N f_s M_s \quad . \tag{2.4}$$

The network-wide number of received search messages must equal the network-wide number of sent search messages. The average frequency of received search messages per node is then

$$F_s = F_{s,network} / N = f_s M_s \quad , \tag{2.5}$$

which is proportional to the search frequency.

The number of message transmissions depends on the overhead. Inserting Definition 2.8 into (2.5) gives

$$F_s = f_s M_s = f_s \Omega_s N_s \quad . \tag{2.6}$$

Equation (2.6) can also be rewritten as

$$\Omega_s = \frac{F_s}{f_s N_s} \quad , \tag{2.7}$$

giving a convenient way to determine overhead with simulations. The corresponding equation can be applied to index distribution as well. The total traffic (assuming $F_m$=0) in the network can be determined as

$$F = F_s + F_i = f_s M_s + f_i M_i = f_s N_s \Omega_s + f_i N_i \Omega_i \quad . \tag{2.8}$$

In most deterministic distribution algorithms, $N_s$ and $N_i$ can be described as functions of $N$. The overheads are typically constant and specific to the distribution algorithm and topology. This allows optimizing the network for given search/index ratios.

### 2.4.6 Overhead of distribution algorithms

An overhead of $\Omega = 1$ is achievable in certain structured topologies, for example rings, stars and fully connected topologies. In the lack of a structure, the efficiency is reduced. The standard distribution algorithms for random topologies, flooding and random walks, are far from optimal. Practical systems therefore often reduce the overhead by reducing the determinism.

#### Flooding

In flooding without TTL limitations, each node in the network forwards a message exactly once. After the first copy is received, other copies of the same message are ignored. A message forwarded by node $v$ is sent to $d(v)$-1 neighbors, where $d(v)$ is the degree of $v$. Therefore each node $v$ causes d($v$)-1 messages to be received by other nodes.

**Hypothesis 2.1.** The search overhead of flooding in a network consisting of nodes $v_i$ ($i$=1…$N_s$) with an average degree of $D$ is

$$\Omega_s = \frac{M_s}{N_s} = \frac{1}{N_s} \sum_i d(v_i) - 1 = D - 1 . \tag{2.9}$$

To test this hypothesis we perform a simulation of a network with 1000 nodes and a varying average degree. For each scenario, we simulate 10

different artificially generated random power-law topologies, each with 1000 searches, and the results are averaged. The experimental results presented in Table 2.5 confirm our hypothesis with a maximum error of 1.5%.

**Table 2.5. Overhead of flooding.**

| $D$ | $N_s$ | $M_s$ | $\Omega_s$ | *Expected* $\Omega_s$ |
|---|---|---|---|---|
| 4 | 999 | 2982 | 2.9850 | 3 |
| 6 | 999 | 4960 | 4.9650 | 5 |
| 8 | 999 | 6930 | 6.9369 | 7 |
| 10 | 999 | 8892 | 8.9009 | 9 |
| 12 | 999 | 10846 | 10.8569 | 11 |
| 14 | 999 | 12792 | 12.8048 | 13 |

Gnutella [Gnutella], which is commonly used as a reference, had an average degree of $D = 5.50$ in October 2000 [LCC+02]. Lv et al. [LCC+02] found through simulation on the actual Gnutella topology of October 2000 that a node, on average, receives the same query 4.5 times, thus $\Omega_s \approx 4.5 = D - 1$, which is in line with our hypothesis.

Decreasing the degree reduces the traffic but increases the network diameter (and consequently the search delay) and the risk of network partitioning. It is also difficult to reduce the degree in a controlled way: the lack of coordinated topology maintenance unavoidably creates a power-law network with a high degree for some nodes. A typical approach to reduce the overhead is to limit the maximum number of hops a query is forwarded using a TTL value. This reduces the set of queried nodes $N_s$ to a subset of all the nodes $N$. As the coverage is reduced, the system becomes non-deterministic. To obtain full coverage, the TTL can be gradually increased until the whole network is covered. The various techniques of iterative deepening, including Hurricane Flooding [JJ07], are therefore deterministic. The other improvements to flooding presented in Section 2.1.3 are non-deterministic.

### Random walk

Also random walk reduces the number of messages at the cost of a reduced coverage. If the resource requester generates a single walker with an initial TTL of $T_{InitialTTL}$, then the number of examined nodes is $N_s \leq T_{InitialTTL}$ because of the possibility of visiting the same node multiple times. The number of messages caused by the search is $M_s = T_{InitialTTL}$. Therefore $\Omega_s \geq 1$. For a small $T_{InitialTTL}$, revisiting occurs infrequently, and $N_s \approx T_{InitialTTL}$, i.e. $\Omega_s \approx 1$. To cover all $N$ nodes with a given probability, $T_{InitialTTL} >> N$, whereas the fraction of repeated visits increases significantly. If $N_{walkers}$ parallel walkers are generated by the resource requester, the number of examined nodes is $N_s \leq N_{walkers} T_{InitialTTL}$. While the delay is reduced, the number of messages is still $M_s = N_{walkers} T_{InitialTTL}$, thus $\Omega_s \geq 1$. The nodes covered by different walkers overlap especially

for a large $N_{walkers}$ as all walkers start at the same location. The problem is that a walker cannot know which nodes have been visited by the other walkers. The walkers could regularly communicate by sending messages through the resource requester; however, this extra communication creates additional overhead.

Random walk is a non-deterministic search method [JJ07]. In basic random walk it is impossible to guarantee that all nodes are covered by the search, but increasing $T_{InitialTTL}$ and $N_{walkers}$ increases the probability that most nodes are covered. Unfortunately, this increases $\Omega_s$ significantly. Because the non-deterministic behavior, we do not consider random walk in its basic form in this work. However, in Section 3.4 we provide extensions to guarantee finding all resources.

### 2.4.7 Frequency of index updates

We categorize systems into *push* and *pull* types depending on the party initializing and controlling the distribution of index updates. Indexing in today's peer-to-peer networks is mainly of the push type, where the index update is initiated by the resource provider. Search engines use a pull type of index updating: the indexing node itself controls the collection of index information, for instance, through a web crawler. Generally, push type index distribution gives better performance and freshness since the distribution is initialized when a resource has changed and the frequency of periodical updating can be low. Pull type index distribution is used in search engines partially because the web servers do not support push updates and cannot be aware of all search engines. Pull type index distribution also allows the indexing node to control the update frequency. However, the inability to detect when resources have changed leads to long update delays, inaccurate index information, and unnecessary index transfers. In this work, we only study the push type of updates.

A single update message may contain updates to several resource descriptors. The size of an update message is typically limited to a maximum number of resources, $R_m$. A full update of all $R_n$ resources available at a node therefore requires $R_n / R_m$ messages. Including $R_m$ resources in a message increases the message size less than $R_m$ times the original size, since the headers typically use a large fraction of the message.

We distinguish between event-driven and periodical updating of index information [MBB06]. *Event-driven updating* generates an index update describing all or part of its shared resources in the following situations:

1. *Entry index updates* are generated when a node joins the system. The local index of the joining node is distributed using one or several index update messages. A node generates these messages at an average frequency $f_{i,entry}$, which depends on the churn rate $f_{churn}$ of the network and the number of resources published per message. As $f_{churn}$ is a network-wide parameter and $f_{i,entry}$ is node specific, the relationship is $f_{i,entry} = f_{churn} \, R_n / (R_m N)$, where $N$ is the number of nodes in the network.

2. A *modification index update* is generated when a new shared resource becomes available, a resource is modified or when a resource is removed. Only the modified resource descriptor is included. A node generates these updates at an average frequency $f_{i,modification}$, which depends on the user's behavior. The modification index update frequency can be assumed proportional to the size of the node's local index.
3. An *exit index update* is generated when a node leaves the system to inform that all resources of the node are removed. A single indication removes all resource descriptors without explicitly listing them. In a stable system the long term frequency of nodes leaving equals the frequency of nodes joining, and we use the churn rate $f_{churn}$ for both. In practice, a fraction $p_{fail}$ of the nodes fail without sending exit index updates. A node therefore generates exit index updates at a frequency $f_{i,exit} = (1-p_{fail})f_{churn}/N$.

*Periodical updating* adds a fourth type of update:

4. *Periodical index updates* are required by systems where the index entries age. Aging prevents stale information left by, for example, a node leaving the system without sending an explicit exit index update or when this update is lost due to an error or topological change. Periodical index updates further allow nodes that were off-line during the entry index update or modification index update to receive the update later. Index entries are typically refreshed at a system specific frequency adjusted to balance the amount of traffic and the risk of stale index information. As the index can be divided between several messages, the frequency depends on the number of resources of the node. A node therefore generates periodical index updates at a variable frequency $f_{i,refresh} = f_{refresh,resource}R_n / R_m$, where $f_{refresh,resource}$ is the constant frequency of updating a specific resource. Alternatively, $f_{i,refresh}$ can be kept constant, whereas refreshing $R_n$ resources requires $R_n / R_m$ update rounds with a variable resource specific update frequency of $f_{refresh,resource} = f_{i,refresh}R_m / R_n$. A constant index update frequency is motivated when the index updates have additional functions, such as topology maintenance.

Often, both event-driven and periodical updating is used. This combines the responsiveness of event-driven updating with the reliability of periodical updating. The total frequency of index updates generated by a node is $f_i = f_{i,entry} + f_{i,exit} + f_{i,modification} + f_{i,refresh}$. It depends on the frequency of modifying resources, the churn rate, and the implementation parameters. As we discuss later in this chapter, the ratio $r$ between the search frequency $f_s$ and the index update frequency $f_i$ is an important parameter determining the best architecture for a given system. Unfortunately, both $f_s$ and $f_i$ are difficult to estimate in advance.

### 2.4.8 Common indexing architectures

The taxonomy [RM06] divides resource discovery architectures into three categories depending on the use of indexing:

1. In *local indexing* the nodes only store their own local index and no index distribution is used. All nodes must therefore be queried separately ($N_s = N - 1$, $N_i = 0$). The most common of this type of architecture is the *search flooding architecture* (also called fully distributed or pure architecture), where the topology is (nearly) random power-law and flooding is used as the search algorithm. All links are search links. The architecture is uniform. This architecture is depicted using QIL in Figure 2.3 (a).
2. In *centralized indexing*, the indices of all nodes are transferred to a single centralized node as depicted in Figure 2.3 (b). Each node has a search link and an index link to the centralized node. For the centralized node $N_s = 0$ and $N_i = 0$ while for the other nodes $N_s = 1$ and $N_i = 1$. This concentrates the entire load on the centralized node, whose capacity limits the scalability of the system. Centralized indexing is therefore, by definition, non-uniform. Even in a centralized architecture the index may be replicated to or divided between several nodes, for which various strategies have been proposed in [SG03]. Conceptually the architecture is still centralized and an ordinary node sees the server cluster as a single centralized server.
3. In *distributed indexing*, the load of indexing is distributed between several nodes. Only a subset of the nodes are queried ($0 \leq N_s \leq N$-1, $0 < N_i \leq N$-1).

Most current solutions use distributed indexing, which can further be divided into several subcategories.

### Centralization

The most common distributed indexing architecture is a two-layer hierarchical architecture, usually called *semi-centralized architecture* or super-peer architecture. The semi-centralized architecture divides nodes into $C$ clusters ($1 \leq C \leq N$), where each cluster contains one indexing node, called a *super-peer* or *super-node*. The other nodes are called *ordinary nodes*. Each ordinary node is connected with a search link and an index link to the super-node in their cluster, as depicted in Figure 2.3 (c). The ordinary node transfers its local index to the super-node ($N_{i,ON} = 1$) while the super-node does not distribute its index ($N_{i,SN} = 0$). The super-nodes are interconnected with search links, usually in a random power-law topology. Searching is performed by distributing the search request to each super-peer in the network with flooding. A searching ordinary node generates $N_{s,ON} = C$ messages while a searching super-node generates $N_{s,SN} = C$-1 messages. The architecture essentially forms a hierarchy, where the super-nodes form the upper layer and ordinary nodes form the lower layer.

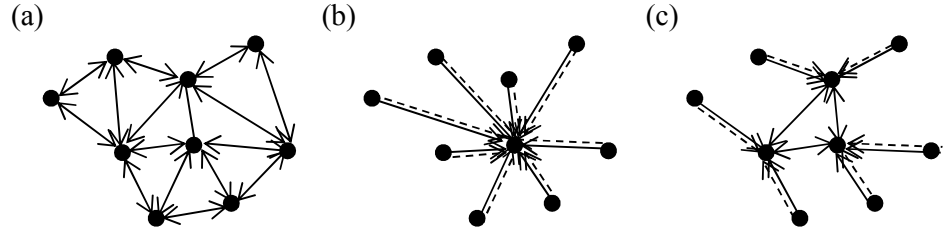


**Figure 2.3. (a) Search flooding, (b) centralized, and (c) semi-centralized architectures modeled with QIL.**

The fraction of the nodes being super-nodes is $C/N$ and ordinary nodes $(N\text{-}C)/N$. Applying Equation (2.8) to both these groups of nodes, we obtain

$$\begin{aligned} F &= F_s + F_i \\ &= f_s \Omega_s \left( \frac{C(C-1)}{N} + \frac{(N-C)C}{N} \right) + f_i \Omega_i \left( \frac{C \cdot 0}{N} + \frac{(N-C) \cdot 1}{N} \right) \\ &= f_s \Omega_s \left( \frac{C^2 - C}{N} + \frac{NC - C^2}{N} \right) + f_i \left( \frac{N-C}{N} \right) \\ &= \left( f_s \Omega_s - \frac{f_s \Omega_s + f_i}{N} \right) C + f_i . \end{aligned} \tag{2.10}$$

This is an increasing function of $C$ when the derivate $dF/dC$ is positive:

$$N > 1 + \frac{f_i}{f_s \Omega_s} , \tag{2.11}$$

which can be rewritten as

$$r > \frac{1}{\Omega_s (N-1)} . \tag{2.12}$$

In these situations, $F$ is minimized at the lowest allowed value $C$=1, whereas the architecture is centralized. Only in the rare situation that index updates are very frequent compared to search requests, and the network is small, a centralized solution is non-optimal. In that particular case, $F$ is a decreasing function of $C$, whereas $F$ is minimized at $C$'s maximum allowed value $C$=$N$.

Centralization thus generally improves the scalability from the network perspective, and the lowest traffic is generated in a centralized system. The lower the number of super-nodes, the less traffic is generated, and since fewer nodes need to be queried also the search delay is reduced.

For $C = 1$, the semi-centralized architecture reduces to centralized indexing, whereas the traffic is

$$F = f_s(1-1/N) + f_i(1-1/N) \ . \tag{2.13}$$

Note that the term (1-1/*N*) indicates the fraction of ordinary nodes, i.e. those nodes that need to transport their index and search requests with a single message. The centralized node itself generates no messages for its index updates and search requests. On the other hand, for $C = N$ the semi-centralized architecture reduces to a search flooding architecture:

$$F = f_s \Omega_s (N-1) = f_s \Omega_s N_s \ . \tag{2.14}$$

While the network perspective favors centralization, the node perspective prefers distribution. Assuming searches are uniformly generated among the nodes, the frequency of generated search messages in the network in total is $Nf_s$. In the semi-centralized architecture, the frequency of the search messages received by an ordinary node is $F_{s,on} = 0$ while the corresponding frequency of a super-node is $F_{s,sn} = Nf_s\Omega_s$. Whereas an ordinary node in a semi-centralized architecture does not receive any search messages from other nodes, a super-node receives all search requests of the network and stores a large index. It does not benefit from the improved network-wide efficiency. At the same time, the capacity of the ordinary nodes is unused.

Concentrating indices in a few nodes results in some nodes being bottlenecks for scalability. Assuming a message handling capacity of $F_{max,v}$ messages per time unit of super-node $v$, this node can operate in networks with at most $N_{max}$ nodes, where

$$N_{\max} = F_{\max,v} / \Omega_s f_s \ . \tag{2.15}$$

In the semi-centralized architecture, the search capacity of the system does not increase as new super-nodes are added since all queries must be distributed to all super-nodes. When the capacity of the weakest node is exceeded, all queries cannot be processed and the system becomes non-deterministic. Thus, the maximum number of nodes in the system is

$$N_{\max} = \min(F_{\max,1}, F_{\max,2}, ..., F_{\max,C}) / \Omega_s f_s \ . \tag{2.16}$$

Typically the super-nodes are selected rather randomly among the eligible nodes (having adequate bandwidth and no NAT-restrictions). The large difference between super-nodes and ordinary-nodes creates an incentive problem, which may discourage a node from becoming a super-node. The larger the network, the larger the difference in load between ordinary nodes and super-nodes. Therefore, semi-centralized architectures are mainly suitable for systems where nodes with substantially higher capacity exist and can be identified, where there is a clear third-party providing the system, or where there are additional rewards for centralized nodes. Moreover, centralization introduces critical points of failure – the centralized nodes must be more reliable and longer available than other nodes since they maintain the index of several nodes.

### Parallel Index Clusters and Parallel Search Clusters

In their work [CG03] on the Search/Index link model, Cooper and Garcia-Molina propose two new architectures, *Parallel Index Clusters*

(PIC) and *Parallel Search Clusters* (PSC), which use a defined topology but with arbitrary placement of index information. In the PIC architecture, nodes are divided into $C$ clusters. Index links connect all nodes within the cluster. For the index links, PIC can use any topology that guarantees distribution of index information from all nodes to all nodes within the cluster. Suitable topologies include rings, stars and random topologies with flooding. The index overhead depends on the chosen topology and distribution method. Each node has a search link to at least one node in each cluster. Thus, to search for a resource, at least one node in each cluster is queried ($N_s = C$) and consequently the indices of all nodes are accessed. Ideally, only one node in each cluster is queried whereas $\Omega_s = 1$. On average, $N_i = N/C$.

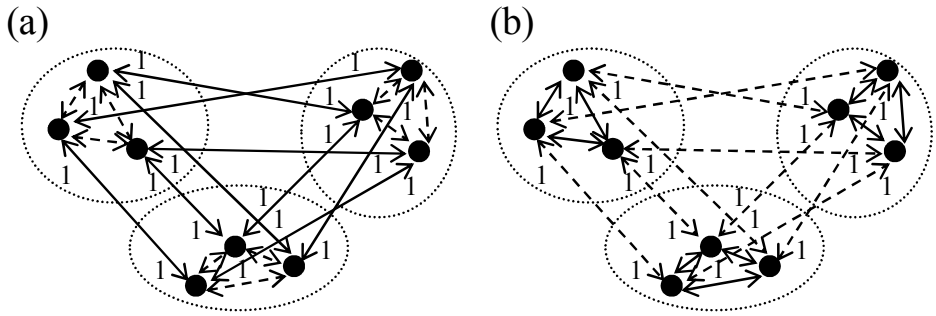


**Figure 2.4. A (a) PIC network and a (b) PSC network modeled with QIL.**

The PSC architecture works in a similar way, but with search and index links interchanged. The local index is distributed to one node in each cluster: $N_i = C$. Ideally, $\Omega_i = 1$. In searching, each node in the local cluster is queried. On average, $N_s = N/C$. Figure 2.4 depicts a PSC network and a PIC network using QIL in subfigures (a) and (b), respectively. Clusters are indicated with dotted circles[2].

### 2.4.9 Simulator for overlay networks

We develop a simulator, PONGsim (Python Overlay Network Graphical Simulator) [PONGsim], that implements the described modeling framework. This section gives a brief presentation of the simulator with which the architectures in this work are evaluated. We focus on presenting how the simulator relates to the modeling framework. For a technical presentation we refer to [Bei09].

Our simulator is a discrete-event simulator. In a discrete-time simulator (also called event-driven simulator) future events are scheduled using a queue ordered according to the time when the event takes place. The simulator picks the first-occurring event from the queue, advances the simulated time to the time of the event and performs the action related to the scheduled event [JBS92]. Each event may give rise to new future events. Typically events correspond to receiving a message or performing the actions of an expired timer.

[2] The notation for indicating clusters is not defined by QIL but rather included for clarity.

The simulator is implemented using the Python language, with the reasoning that the time won by faster prototyping of new algorithms is larger than the time lost by slower execution in an interpreted language. The simulator is implemented as a generic framework with a modular structure to allow using it in a wide range of purposes. To support development and debugging of algorithms, the simulator supports a graphical mode and the possibility to control the simulation speed and examine node properties and messages. For running multiple scenarios in sequence, the simulator provides a faster batch mode, with a common queue for scheduling parallel simulation across multiple processors or processor cores. The simulator allows collecting statistical information using several tools, and performs averaging across multiple instances of a simulated scenario. After all defined scenarios are performed the simulator can collect the results into tables showing the effect of various input parameters on the examined metrics. Scenarios are defined in scenario files as collections of parameters together with their values. The simulator allows specifying value ranges for repeating a simulation with all combinations of values of the defined parameters.

The simulator implements the following models:

- *Application model*. Each simulated node runs an application, which implements the behavior of the node and, in particular, the reactions to events from other simulator models. The application defines how received messages are processed, how timer expirations are handled, how the node connects to other nodes and how the node reacts to events of the user including search requests and resource modifications. The simulator comes with applications for standard algorithms including flooding, random walks, and PIC.
- *SIL model*. A particular feature of our simulator is that it allows defining the topology and message forwarding using the SIL or QIL models. The topology is then specified using search and index links with a TTL defining the forwarding properties. Additionally temporary links (e.g. representing UDP messages) can be modeled. The SIL/QIL model supports three classes of messages: index updates, search requests and generic messages (including e.g. search replies and maintenance messages). The model specifies the processing of messages using flooding (forwarding links) or one-hop updates (non-forwarding links). The simulator also allows extending the SIL/QIL models further to generalize for other forwarding behavior using the underlying search/index link topology. The application can then decide the neighbors to which a message is forwarded and modify the TTL. This allows search methods like random walks to be implemented.
- *Topology model*. The topology defines the links connecting nodes. A separate topology generator allows various topologies to be specified, including fully connected, random power-law, and ring topologies. Power-law networks are generated based on the method proposed by Barabási and Réka [BA99], which generates power-law topologies with the exponent $\gamma = 2.9 \pm 0.1$. At each step, a new node is added with $k$ links from the new node to $k$ different existing nodes. The

probability $\Pi(v_i)$ that a new link is established to a node $v_i$ is $\Pi(v_i) = d(v_i) / \Sigma_j\ d(v_j)$. Since each step increases the degree of two nodes by one, the topology generator can generate topologies with an even average degree given as parameter. The simulator also includes a modified version of this algorithm which additionally can produce fractional and odd average node degrees.

- *Resource model*. The resource model simulates the user-related resource events by controlling the generation of searches, and the addition and the removal of resources. Search requests are generated at exponentially distributed random intervals with the intensity $\lambda_s = f_s$. The requested resource is selected from the resources currently existing in the system using a uniform or a Zipf distribution. For the Zipf distribution, the query rate of the $i$th most popular resource is proportional to $i^{-\alpha}$ with the parameter $\alpha$ controlling the skewness. Resources are added to the system at exponentially distributed random intervals with the intensity $\lambda_r = f_r$ and remain in the system an exponentially distributed random interval $T_r = N_{resources} / \lambda_r$ determined using Little's law to keep the number of resources at a given average level $N_{resources}$. The location of resources can be uniformly random or selected using a Zipf distribution. The application model is notified with events from the resource model and can, depending on implementation, react by generating modification index updates. Additionally the application can issue periodical index updates. The resource module follows a search request through the system and collects statistics about, for example, the success ratio $R_{success}$ and search delay $T_s$.
- *Churn model*. Nodes can be created and destroyed during the simulation time to simulate churn. To these events the application can respond by updating the overlay topology and generating entry and exit index updates. The simulator also defines an initial topology.
- *Delay model*. In addition to fixed and uniformly random link delays, the simulator can import delays from a matrix of pair-wise delays stored in a file. In particular, this allows mapping simulated nodes to measured nodes, for instance, obtained from delay measurements between DNS servers using the King method [GSG02].

Additionally, the simulator provides support functions for interfacing nodes with other components and provides control for the graphical interface, logging, simulator performance monitoring and error handling.

Collection of statistics is started after a specified settling period. The simulator finishes when given stop criteria are satisfied, for instance, after a given time or a given number of generated search requests. The simulator collects three types of statistics: snapshots, node-specific statistics and simulator-wide statistics. Snapshots are generated periodically to examine global state at different points in time, for example, the load distribution between nodes. Node-specific statistics are available per node, but typically the average, maximum value, minimum value, or sum of all nodes is evaluated. The node-specific statistics can be divided by the simulation time (excluding the settling period) to obtain the frequencies $F_i$, $F_s$, $L_i$ and $L_s$. Simulator-wide statistics include

information not related to specific nodes, such as the number of links in the network.

## 2.5 Analyzing uniform indexing architectures

As we have seen, the lowest traffic and delay is normally obtained with a centralized system. However, this represents an extremely non-uniform architecture. To balance the indexing load, all nodes in the system must participate in indexing and each node must index a roughly equal number of nodes. To balance the search load, different resource requesters contact a different set of indexing nodes. Further, to support complex queries and guarantee deterministic behavior in an unstructured system, a search request must be compared with all resource descriptors in the system.

Both index distribution and searching involves distribution of a message to several nodes. Index distribution is a *proactive* transaction, i.e. it is performed before the resource is requested, while the search is a *reactive* transaction performed as a consequence of the resource request. In order to reduce the search traffic, the number of queried nodes must be reduced. At the same time, the index load must be minimized. This section studies how the balance between reactive and proactive operations is adjusted to minimize message overhead.

### 2.5.1 The Search/Index Space model

To model searching and indexing in uniform architectures, we introduce a model called the Search/Index Space. The model, originally published in [Bei10], is here slightly extended to account for non-optimal search and index distribution algorithms. The purpose of the model is to illustrate and analyze the balance between searching and indexing. The model describes architectures that are (1) deterministic, (2) uniform, and (3) support complex queries.

We assume that every node in the system can be a resource provider. Each node $v$ distributes its local index to a set of indexing nodes $\mathbb{P}_v$. Consequently, a node $w$ stores the index of a set of indexed nodes $\mathbb{N}_w = \{v \mid w \in \mathbb{P}_v\}$. A search request sent by node $v$ reaches a set of queried nodes $\mathbb{R}_v$ and locates all resources provided by the nodes $\mathbb{H}_v = \cup_{v \in \mathbb{R}} \mathbb{N}_v$. In order to guarantee finding all matching resources, the search must reach the index of all nodes. Thus, if the network consists of the set of nodes V, the requirement is $\mathbb{H}_v = V$ for all $v \in V$. Our goal is to minimize the sizes $|\mathbb{P}_v|$ and $|\mathbb{R}_v|$ of the sets while still fulfilling the above requirement.

#### Continuous model

The Search/Index Space model arranges nodes into a space with two orthogonal dimensions as depicted in Figure 2.5: a proactive and a reactive dimension. Each point in the space is allocated to a node. Index distribution is performed along the proactive dimension and searching along the reactive dimension. The requirement is that the index distribution must traverse each point in the proactive dimension, and the search distribution must traverse all points in the reactive dimension. The

nodes that are allocated to the points in space that the distribution crosses receive the distributed message. Since the dimensions are orthogonal, all possible search distributions converge with all possible index distributions. Consequently all indices are covered and the model models deterministic systems (in static networks). Since all indices are examined, complex queries are supported. The architecture determines how points are allocated to nodes and how the dimensions are traversed. If each node is allocated an equally sized part along both dimensions, the architecture is uniform.

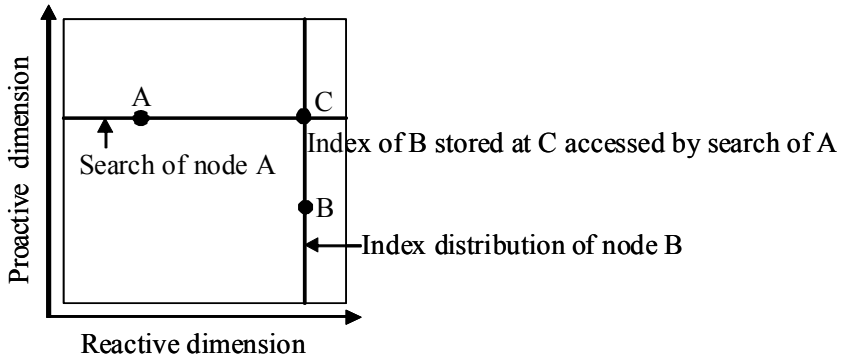


**Figure 2.5. The continuous Search/Index Space model.**

## Discrete model

For practical purposes we use a discrete version of the Search/Index Space model. The discrete version depicted in Figure 2.6 represents the network consisting of $N$ nodes as a $P$ x $R$ matrix. The $N \leq PR$ nodes are elements in the matrix. An index update is distributed to one node in each of the rows and a search request is distributed to one node in each of the columns. Thus, $P = |\mathbb{P}|$ and $R = |\mathbb{R}|$. Using our previous definitions, $N_i = P - 1$ and $N_s = R - 1$. The Search/Index Space models uniform index allocation if the nodes are distributed uniformly in the matrix, i.e. each row and each column contains the same number of nodes. Perfect uniformity is obtained if $N = PR$ but in practical cases the near-uniformity obtained with $N < PR$ is adequate whereas some slots are empty.

Referring to the SIL model, index links connect nodes in the proactive dimension and search links connect nodes in the reactive dimension. The model allows any topology of search and index links that results in the distribution to the correct nodes. The model does not specify the algorithms used for distribution of index updates and search requests. Instead, it models the performance under various distribution algorithms.

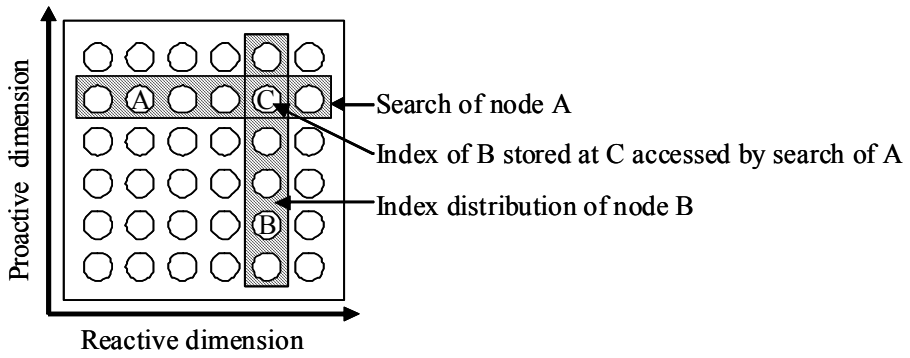


**Figure 2.6. The discrete Search/Index Space model.**

The model can describe various common architectures. A fully reactive architecture is modeled as a 1 x $N$ matrix shown in Figure 2.7 (a). Such an architecture is Gnutella, where flooding is used as a search algorithm and a random connected topology of search links connects the nodes. A fully proactive architecture, such as a global index network is modeled as an $N$ x 1 matrix represented by Figure 2.7 (b). The Direct Index approach described in Section 3.6 uses this approach. Hybrid proactive-reactive architectures are formed when $P > 1$ and $R > 1$. The PIC and PSC architectures are examples of architectures having both a reactive and a proactive dimension. In PIC, the $P$ clusters are the rows in the matrix, and nodes are fully connected with search links with nodes in the same column. In PSC, the $R$ clusters are the columns in the matrix, and index links fully connect all nodes in a row.

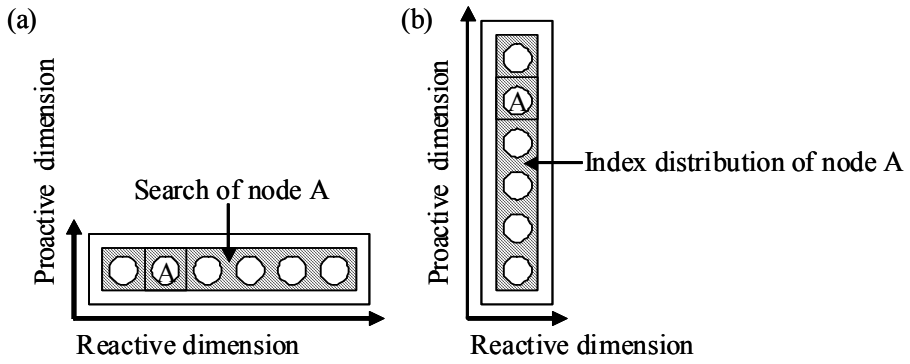


**Figure 2.7. The Search/Index Space model with (a) a reactive system and (b) a proactive system.**

The model can also be used to form new architectures. Any arrangement of search links connecting all nodes in a row as well as any arrangement of index links connecting all nodes in a column is valid. For example, connecting nodes with ring topologies in both the reactive and proactive dimensions results in a torus, as depicted using QIL in Figure 2.8.

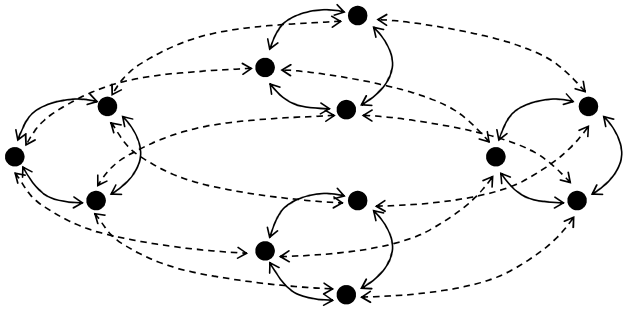

**Figure 2.8. Torus architecture.**

### 2.5.2 Optimal balance between proactive and reactive operations

We now want to determine $P$ and $R$ so that the total traffic received by the $N$ nodes is minimized. To simplify the presentation, let us first assume that both the search algorithm and the index distribution algorithm are optimal in terms of message overhead: $\Omega_s = \Omega_i = 1$. Optimally, distributing an index update from a node to all other nodes in the same column requires $M_i = N_i = P - 1$ messages. Correspondingly, distributing a search request from a node to all other nodes in the same row requires $M_s = N_s = R - 1$ messages.

We use the continuous version of the Search/Index Space model, whereas $P$ and $R$ are fractional numbers which finally are rounded to integers. In the continuous model, $N = PR$. We first determine the optimal $P$, from which we can calculate the corresponding $R = N / P$. As a node receives index messages from $P-1$ other nodes, each generating messages at frequency $f_i$, the number of index messages received by a node per time unit is

$$F_i = f_i(P-1). \tag{2.17}$$

The number of search messages received per time unit is correspondingly

$$F_s = f_s(R-1). \tag{2.18}$$

The total traffic (inserting $R = N / P$) is

$$F = F_i + F_s = f_i(P-1) + f_s((N/P)-1). \tag{2.19}$$

Deriving (2.19) with respect to $P$ gives

$$F' = f_i\left(1 - \frac{rN}{P^2}\right), \tag{2.20}$$

where

$$r = f_s / f_i \tag{2.21}$$

is the search/index ratio. As only positive values of $P$ are valid, (2.20) is zero when

$$P = \sqrt{rN}\,. \tag{2.22}$$

The second derivative of (2.19)

$$F'' = \frac{2rNf_i}{P^3} \tag{2.23}$$

is positive for $P > 0$ as $r > 0$, $N > 0$, $f_i > 0$, indicating that this is a minimum. Thus, (2.22) gives the value of $P$ that minimizes the total traffic. In that situation, the total traffic is

$$F = F_i + F_s = 2f_i\sqrt{rN} - f_i r - f_i = 2\sqrt{f_i f_s N} - f_s - f_i . \tag{2.24}$$

In practical networks, the values $P$ and $R$ need to be rounded to integers:

$$\begin{aligned} P &= \left\lceil \sqrt{rN} \right\rceil \\ R &= \left\lceil N/P \right\rceil = \left\lceil \sqrt{N/r} \right\rceil \end{aligned} . \tag{2.25}$$

In addition to the frequency of received messages we consider the frequency of processed messages, i.e. the message processing load. The total load according to (2.2) and (2.1) is

$$L = F_i + f_i + F_s + f_s = f_i P + f_i rN/P . \tag{2.26}$$

Deriving the load with respect to $P$ gives

$$P = \sqrt{rN} . \tag{2.27}$$

Thus, we observe that the $P$ that minimizes the network traffic also minimizes the processing load on the nodes.

### 2.5.3 Optimal balance for non-optimal search and index distribution algorithms

We now extend the analysis to the general case, where the search and index algorithms may be non-optimal. We model the required number of transmissions to distribute a search message to $N_s$ nodes using a linear overhead as

$$M_s = \Omega_s N_s + \omega_s . \tag{2.28}$$

For example, in flooding $\Omega_s > 1$ and $\omega_s = 0$. In an optimal search algorithm $\Omega_s = 1$ and $\omega_s = 0$. Correspondingly, the number of message transmissions required to distribute an index update to $N_i$ nodes is

$$M_i = \Omega_i N_i + \omega_i . \tag{2.29}$$

A node receives on average

$$F_i = f_i\left(\Omega_i N_i + \omega_i\right) = f_i\left(\Omega_i P - \Omega_i + \omega_i\right) \tag{2.30}$$

index messages and

$$F_s = f_s\left(\Omega_s R - \Omega_s + \omega_s\right) \tag{2.31}$$

search messages per time unit. By requiring the derivate of $F = F_i + F_s$ to be zero, we obtain the optimal $P$:

$$P = \sqrt{\frac{\Omega_s}{\Omega_i} rN} = \sqrt{N \frac{f_s \Omega_s}{f_i \Omega_i}}, \tag{2.32}$$

which gives the optimal $R$,

$$R = \frac{N}{P} = \sqrt{\frac{N\Omega_i}{r\Omega_s}} = \sqrt{N \frac{f_i \Omega_i}{f_s \Omega_s}}. \tag{2.33}$$

**Theorem 2.1.** A constant overhead $\omega_s$ and $\omega_i$ of the search and index algorithms does not affect the optimal $R$ and $P$.
**Proof.** Equations (2.32) and (2.33) do not depend on $\omega_s$ or $\omega_i$.

### 2.5.4 When are the extremes optimal?

Fully reactive protocols are popular today, partly because of their simplicity due to the lack of index distribution. We now apply the Search/Index Space model to determine in which situations a fully reactive protocol is optimal. For a reactive protocol to be optimal, the optimal $P$ must be $P \leq 1$, i.e. there must be no proactive transactions. Applying this requirement to Equation (2.32), gives the search/index ratio for which a reactive architecture is optimal:

$$r \leq \frac{\Omega_i}{N\Omega_s}. \tag{2.34}$$

We can see that the feasibility of the reactive algorithm increases when modification of the resources becomes frequent compared to searching. The feasibility of fully reactive architectures also decreases as the network size grows. Note that although there is no indexing in the reactive algorithm, the parameter $\Omega_i$ must be considered since it denotes the efficiency of the index distribution algorithm to which we compare. Comparing to an optimal index distribution algorithm thus yields $\Omega_i = 1$.

Correspondingly applying the requirement $R \leq 1$ to Equation (2.33) gives the criterion for a fully proactive algorithm to be optimal:

$$r \geq \frac{N\Omega_i}{\Omega_s}. \tag{2.35}$$

Combining (2.34) and (2.35), a hybrid proactive-reactive approach is feasible when

$$\frac{\Omega_i}{N\Omega_s} \leq r \leq \frac{N\Omega_i}{\Omega_s} \tag{2.36}$$

or written in an alternative way:

$$\frac{1}{N} \leq \frac{f_s \Omega_s}{f_i \Omega_i} \leq N. \tag{2.37}$$

It is worth noticing that the feasibility of both fully reactive and fully proactive architectures decreases as network size increases, while the hybrid architectures become feasible in large networks. The extremes are

thus feasible in very marginal situations. Consequently, fully reactive protocols like Gnutella need to artificially limit the network size (search scope) through a TTL. The areas of optimality for proactive, reactive and hybrid architectures are depicted in Figure 2.9 for a case where $\Omega_s = \Omega_i = 1$.

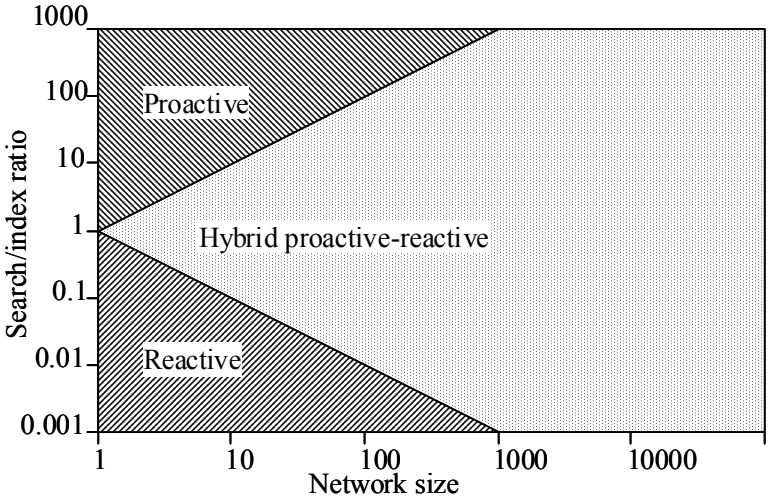


**Figure 2.9. Areas of optimality for proactive, reactive and hybrid architectures.**

In some cases, a hybrid proactive-reactive approach is not applicable, and only fully reactive and fully proactive solutions can be considered. Then, the frequency of received messages per node per time unit is $F = F_i = f_i(\Omega_i N + \omega_i)$ for the proactive solution and $F = F_s = f_s(\Omega_s N + \omega_s)$ for the reactive solution. The proactive solution is optimal when

$$f_s \Omega_s N + f_s \omega_s > f_i \Omega_i N + f_i \omega_i . \tag{2.38}$$

If we assume that there is no constant overhead $\omega_s = \omega_i = 0$, then the proactive solution is optimal when

$$f_s \Omega_s > f_i \Omega_i , \tag{2.39}$$

which can be written as

$$r > \Omega_i / \Omega_s . \tag{2.40}$$

We observe that in this particular case, the choice is independent of the network size.

### 2.5.5 Analyzing practical implementations

The model does not consider the size of search and index update messages. Typically, the number of messages received per time unit is more critical than the size of the messages. This is in particular true for evaluating energy consumption and processing load. Furthermore, the message size is heavily dependent on the actual implementation. Still, the model can easily be adapted to consider the average size $S_s$ of a search message and the average size $S_i$ of an index update message. The

message sizes affect the performance proportionally to the frequency of the messages. Multiplying the average message size with the average transmission frequency gives the average bandwidth. The average bandwidth of received index messages is $S_i F_i = S_i f_i(\Omega_i N_i + \omega_i)$ and the average bandwidth of received search messages is $S_s F_s = f_s S_s(\Omega_s N_s + \omega_s)$. Modifying Equation (2.32) to calculate the optimal cluster size when the average message size is known gives:

$$P = \sqrt{\frac{S_s \Omega_s}{S_i \Omega_i} rN} = \sqrt{N \frac{f_s S_s \Omega_s}{f_i S_i \Omega_i}} . \tag{2.41}$$

The size should be considered especially in the implementations that transports the index update of several resources in a single update message. In these solutions, the frequency $f_i$ is reduced while the size $S_i$ is increased.

### 2.5.6 Applying the Search/Index Space model to PIC and PSC

Equations (2.25) allow us to determine the optimal number of clusters in PIC and PSC for given search/index ratios. In PIC, the index information is proactively distributed within clusters and search requests reactively between clusters. In the Search/Index Space model, $R = C$ is the number of clusters and $P = N_c = N / R$ is the average cluster size. The search algorithm is optimal ($\Omega_s = 1$, $\omega_s = 0$) but the index distribution algorithm can be freely selected. In PSC, the reactive and proactive operations are swapped. In the model, $P = C$ and $R = N_c = N / P$. The index distribution algorithm is optimal ($\Omega_i = 1$, $\omega_i = 0$) but the search algorithm is arbitrary.

The inventors of PIC and PSC provide in their later paper [CG06] similar calculations for the optimal cluster size, which give identical results. Our model is, however, more general in that it considers the overhead of various index and search distribution methods. It can be applied to architectures other than PIC and PSC.

Marossy et al. [MCB+04] provide a simulation-based analysis of PIC where a "square-root law" is formulated: the average load over a set of different search/update loads is constant when the number of clusters is the square root of the number of nodes. We can determine their simulation results mathematically by inserting $P = \sqrt{N}$ into (2.26), which gives

$$L = \sqrt{N}\left(f_i + f_s\right), \tag{2.42}$$

confirming that under this condition the total traffic is independent of $r$ and only depends on the total number of generated messages $f_i + f_s$. The simulation of Marossy et al. also indicated that the load increases slower than linearly with an increasing network size. With (2.42) we can accurately confirm that the increase is in the order of $O(\sqrt{N})$. Also the square root allocation in [CS02] shows similar criteria for optimality in a different type of architecture.

**Table 2.6. Properties of the PIC and PSC architectures under optimal cluster size.**

| Architecture | PIC | PSC |
|---|---|---|
| Index topology | Arbitrary connected topology within a cluster * | A link from each node to at least one node in all other clusters |
| Search topology | A link from each node to at least one node in all other clusters | Arbitrary connected topology within a cluster * |
| Index distribution method | Any * | One-hop updates to a node in each cluster |
| Search distribution method | One-hop queries to a node in each cluster | Any * |
| Class | Hybrid proactive-reactive uniform | Hybrid proactive-reactive uniform |
| Links in network | $O(N^2)$ | $O(N^2)$ |
| Index entries per node | $O(\sqrt{N})$ | $O(\sqrt{N})$ |
| Index message scalability, $M_i$ | $O(\sqrt{N})$ | $O(\sqrt{N})$ |
| Search message scalability, $M_s$ | $O(\sqrt{N})$ | $O(\sqrt{N})$ |
| Search delay, $T_s$ | $O(1)$ | star: $O(1)$<br>ring: $O(\sqrt{N})$ |

* The topology and distribution algorithm within a cluster are undefined. For example ring, star and random topologies can be used. The indicated properties are valid for ring and star topologies.

Table 2.6 summarizes the properties of PIC and PSC when the cluster size is optimal. The properties may, however, be rather optimistic for practical networks as the optimal cluster size is difficult to maintain in a dynamic scenario. The practical implementation of a system adjusting the cluster size automatically is difficult as there is no global knowledge about the number of nodes and the search/index ratios available. Furthermore, rearranging nodes between clusters as the optimal cluster size changes is also difficult.

### 2.5.7 Application to non-uniform architectures

The Search/Index Space model is not limited to uniform architectures, even though its practical application is most valuable for these. Consider, for example, a semi-centralized architecture with two super-nodes and four ordinary nodes per super-node. This architecture can be modeled as depicted in Figure 2.10. The model illustrates that a search can be implemented by examining either all ordinary nodes or all super-nodes. The search of a node in its own local index is redundant as the same index information is examined in the super node as well. The model also shows that this architecture is non-uniform.

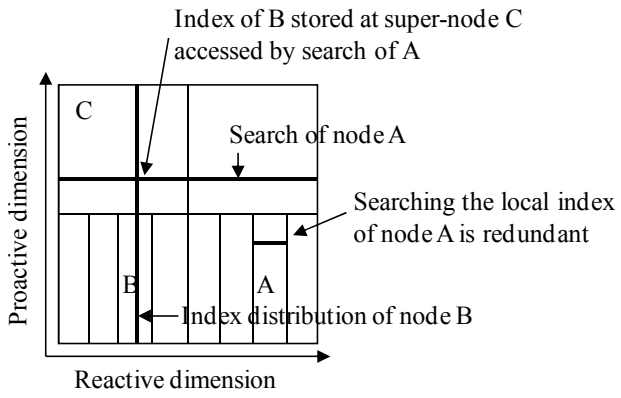


**Figure 2.10. A semi-centralized architecture in the Search/Index Space model.**

## 2.6 Summary

This chapter presented techniques for modeling and analyzing indexing in a resource discovery system. We defined central concepts, including determinism, complex queries, index allocation invariance, and uniformity. We established a set of metrics for evaluating the performance of an overlay. We defined the overhead $\Omega$ of a distribution algorithm and stated that a distribution algorithm is optimal when $\Omega = 1$. For search and index distribution algorithms, we denote the overhead with $\Omega_s$ and $\Omega_i$, respectively. We found that the overhead of flooding is $\Omega_s \approx D - 1$, where $D$ is the average node degree. We divided the frequency $f_i$ of generated index update messages into components: $f_i = f_{i,entry} + f_{i,exit} + f_{i,modification} + f_{i,refresh}$, where $f_{i,entry}$ is the frequency of entry index updates, $f_{i,exit}$ is the frequency of exit index updates, $f_{i,modification}$ is the frequency of resource modifications and $f_{i,refresh}$ is the frequency of periodical index updates. The search/index ratio $r$ is defined as $r = f_s / f_i$, where $f_s$ is the frequency of generated search requests. We concluded that centralization reduces the total network traffic when $r > 1/\Omega_s(N\text{-}1)$, i.e. in almost all practical cases. However, centralization causes unevenly distributed load and creates a capacity bottleneck and an incentive problem. Complete centralization is therefore undesired in many cases. We presented our simulator PONGsim which uses the described metrics to analyze the performance of overlay networks.

We proposed our Search/Index Space model for illustrating and analyzing searching and indexing in architectures that are deterministic, uniform and support complex queries. Above all, the model can be used to determine the optimal balance between searching and indexing, i.e. to determine the number of nodes $P$ to which an index update is distributed and the number of nodes $R$ contacted in a search. Denoting the number of nodes in the network with $N$, the optimal $P$ and $R$ are $P = \sqrt{Nf_s\Omega_s / f_i\Omega_i}$ and $R = \sqrt{Nf_i\Omega_i / f_s\Omega_s}$, respectively. A fully reactive solution is optimal in the marginal case when $r \leq \Omega_i/N\Omega_s$ and a fully proactive solution is optimal when $r \geq N\Omega_i/\Omega_s$. In all other situations, it makes sense to combine reactive and proactive operations.

# Chapter 3

# Uniform indexing architectures

This chapter presents three new types of architectures involving index distribution. The first architecture extends the Parallel Index Clusters [CG03] architecture by allowing the clusters to be arbitrarily interconnected. We present two search algorithms for this architecture. The second architecture minimizes the traffic by making the division between proactive and reactive operations optimal. We present how this architecture can dynamically adapt to changing conditions and how the delay can be reduced. The third architecture is fully proactive with a low traffic overhead. For all architectures we require that it should be possible to locate all existing resources, and that the load is evenly distributed between nodes. We also aim to support a wide range of query types. All architectures are evaluated by simulations. We start by presenting the research related to indexing in resource discovery systems.

## 3.1 Introduction

We have seen that a centralized architecture provides the lowest message overhead and delay. However, the capacity of the centralized node limits the capacity of the system. The centralized node can also be a single point of failure. When all participating nodes are similar, it is difficult to give a node the incentive to take the additional burden as a centralized node. In uniform architectures, every node maintains a part of the index information. Consequently, the load is distributed more evenly between the participating nodes. These systems are particularly useful in collaborative settings where nodes are similar and have an equal role. The goal of a uniform architecture is to minimize the traffic without concentrating the load in a few nodes.

Indexing is a proactive transaction that transfers information about available resources to other nodes in advance, reducing the number of nodes that need to be queried for a search. With the Search/Index Space

model, we have shown that a large group of scenarios benefit from indexing. In particular, large networks can benefit from a combination of proactive and reactive transactions.

Most resource discovery systems today use some form of indexing. Distributing index information to more than a single node is still a quite uncommon approach, as we will see in Section 3.2. The Parallel Index Clusters (PIC) and Parallel Search Clusters (PSC) are examples of this kind of large-scale index distribution approach. They represent hybrid proactive-reactive architectures, which can be adjusted to be optimal according to the Search/Index Space model. However, the clustered topology imposes certain restrictions. Above all, the topology is difficult to construct and maintain.

In this chapter we propose three new architectures. The first architecture enhances the PIC architecture so that clusters can be interconnected in an arbitrary manner. The second architecture represents a completely new approach to implement a hybrid architecture. A major motivator is the need to dynamically adjust the topology as nodes enter and leave the network. The third architecture shows that a completely proactive architecture can be implemented in an efficient way.

Although uniform architectures assume that all nodes have similar capacity, the proposed architectures can be seen as components of architectures that allow allocating load in a desired way. Firstly, uniform architectures can be used together with centralization in a hierarchical way. The lower layer is then centralized, concentrating the load on the centralized node to an appropriate level. The centralized nodes are connected with a uniform architecture. Secondly, if some nodes have a higher capacity and are willing to take a higher load, the load on these nodes can be increased in a controlled way using a *virtual node* concept. A single node then operates as several virtual nodes, whereas it receives the combined load of these virtual nodes.

The architectures studied in this chapter are uniform, deterministic, and index allocation invariant. Two of them natively provide complex queries and one provides complex queries with special considerations.

## 3.2 Related research

Let us first review the research on indexing, including both uniform and non-uniform architectures.

### 3.2.1 Unstructured systems

Centralized architectures were used in the early peer-to-peer file sharing systems. The most famous example is the original Napster [Napster] for sharing music. As the centralized architectures showed their weaknesses, above all the dependence on a single centralized point, search flooding architectures such as Gnutella version 0.4 [Clip2] were developed. To improve scalability and to support nodes behind firewalls and NATs, semi-centralized architectures have become popular. Semi-centralized architectures are used in Gnutella version 0.6 [KM02] and in the FastTrack protocol on which Kazaa [Kazaa] and Skype [Skype] are built.

Of these, only the search flooding architecture provides a relatively uniform load, but it does not utilize indexing.

In a *Global Index* network [CG03], the topology consists of only index links and the resource descriptors are available to all nodes before the resource actually is needed. This results in fast searches and eliminates the need for search distribution. The architecture is uniform, index allocation invariant, deterministic and fully proactive. Global indexing can be implemented using structured topologies (e.g. rings) or in random topologies, with index distribution based on flooding. Global indexing is used in some distributed database systems [ÖV99] but is often practically unfeasible in large-scale resource discovery, since it requires all nodes to store a complete index over all resources. However, a global index can be used in limited overlays. In [CG03] global indexing is considered for the upper layer of a two-layer hierarchical architecture. The resulting architecture is fundamentally a semi-centralized system with the search and index links exchanged. The architecture is index allocation invariant and deterministic but non-uniform.

In the *Local Indices* approach [YG02], each node maintains the indices of nodes within $r$ hops. The approach uses a method to learn nodes within $r$ hops and to maintain this knowledge in a dynamic topology. Search requests are sent by flooding, but only nodes spaced at $r$ hops examine their index. Thus, the processing power and search delay is reduced. However, as the message is still distributed to all nodes (within a TTL), the search overhead still equals flooding with an additional overhead due to index distribution. We therefore find the improvement to flooding marginal in terms of message overhead. The architecture is deterministic, index allocation invariant and uniform. Local indices with $r$=1 are also used in Gia [CRB+03].

In the *Routing Indices* scheme [CG02], a node maintains a routing table indicating the number of resources of given types reachable through each of its neighbors. Resources thus need to be classified according to a defined set of types. Depending of which of three proposed models is used, the routing table gives the compound number of documents under a branch, the number of documents for each hop up to a specified horizon, or an exponentially aggregated count of documents under a branch. A query is forwarded to the subset of neighbors that reaches most resources of the requested type. While the search requires a lower number of messages than in normal flooding, the index distribution has a cost comparable to flooding. If the manual classification of documents is omitted, as assumed in this work, the algorithm can only count the total number of documents under a branch, which gives a marginal benefit.

In *Percolation Search* [SBR04] the local index is replicated to a set of nodes using a short random walk, the length of which is a function of the network size $N$ and the power-law exponent $\gamma$. A query is implanted on a set of nodes using a similar short random walk. These nodes start a probabilistic flooding search, where the query is sent to a neighbor with a given probability. This results in a search delay in the order of $O(\log N)$. For random power-law networks, $O(\log^2 N)$ messages are generated per query, while in most grown graphs (with a maximum degree of $\sqrt{N}$) the

traffic scales as $O(\sqrt{N}\log^2 N)$. Even though the reported hit rate is fairly high (over 90% in given scenarios), the algorithm is non-deterministic. Additionally, the lack of index allocation invariance contributes to the non-determinism.

*BubbleStorm* [TKL+07] controls the number of index replicas and queried nodes so that the hit rate is arbitrarily high. It uses a concept called BubbleCast to distribute index updates and queries to a specified number of nodes determined by the weight *w*. Each node receiving the message reduces *w* by one and forwards it to *s* (the split factor) neighbors. The currently remaining weight is divided between the *s* neighbors. BubbleStorm provides an algorithm for generating a controlled topology, a random multigraph, where the node degree is proportional to the bandwidth. Even though the hit rate is theoretically controlled, BubbleStorm is fundamentally non-deterministic. A major problem is the lack of index allocation invariance. Consequently, each update round generates new replicas and leaves stale versions of the previous replicas.

In contrast to explicit index distribution, index information can be collected implicitly – a method usually referred to as *index caching*. Index caching must not be confused with content caching, where the actual resource is replicated. A simple caching method is to store copies of the search results in the nodes relaying the search reply to the resource requester, whereas no additional index distribution traffic is generated. This approach is used in [Mar02] and [Freenet]. The performance improvements provided by index caching are based on location locality (resources nearby tend to be more frequently used) and time locality (an accessed resource is often accessed again). Stale index replicas are left in the system when the original local index is modified. Because of the lacking index allocation invariance, the validity time of an entry must be short.

### 3.2.2 Structured systems

Recent research has concentrated on structured systems, where the overlay topology is strictly defined and the location of the index entries within the network is defined. This is in contrast to unstructured systems, where the topology is random and the index can be located anywhere in the network. A structured system utilizes an abstract key space, e.g. in the form of a ring in Chord [SMK+01], a torus in CAN [RFH+01] or a Plaxton tree [PRR97] in Tapestry [ZKJ01] and Pastry [RD01]. Each node has an identifier that identifies its location in the key space. Consistent hashing is used to prevent restructuring when nodes join and leave. Likewise, each stored element has a key identifying its location. The key is generated by a hash function, for example, from the file name, keywords, description, etc. The element is stored at the node whose identifier is "closest" to the key, where the definition of distance depends on the algorithm. The fact that the location of each resource's index is known makes searching efficient, with a scalability in the order of O(log *N*) for a network of *N* nodes in most algorithms. Maintenance of the strict structure is generally claimed costly under a high churn rate [CRB+03],

even if this claim has been questioned [QB06]. Today's peer-to-peer systems show high churn rates, with a median up-time of 60 minutes per node [SGG02].

In order to find a resource in a structured system, the key of the resource must be known. The structured system essentially maps a key to the corresponding data. Therefore, DHTs are naturally constrained to exact-match single-key queries. Chawathe et al. [CRB+03] point out that structured systems are ill-suited for file sharing systems where keyword searches are more prevalent and important than exact-match searches, and most queries are for relatively well-replicated files.

Some types of complex queries can be supported in DHTs through various techniques. Each technique is specific to one or a few query types. Most work, including [AX02], [BAS+04], [BHP+04], [GAA03], [GS04], [HJS+03], [CLG+04] and [SGA+04], aim to support range queries. Multi-attribute queries are supported in [BAS+04], [BHP+04], and [FBG+04], substrings in [HHH+04], equi-joins in [HHL+03], and longest-prefix matching in [BHP+04]. A comprehensive survey of these techniques can be found in [RM06]. A typical approach is to divide the attribute into several partitions, and to allocate each partition to a node. For example, ranges are divided into sub-ranges and strings into words or *n*-character sequences. Each partition must be stored separately in the DHT. Also queries are divided into partitions and each partition is queried separately. This increases both the storage and the query overheads, which decreases the efficiency of the DHT. Furthermore, since each technique enables only a few query types, supporting multiple query types entails implementing several techniques and, in the worst case, several overlays in parallel. Most solutions also require some *a priori* knowledge about the stored data, such as the data-types and expected values. The overlay must therefore be tailored separately for each application. While DHT-based techniques are valuable for many applications, they may not provide the expressiveness and flexibility required by a system shared between multiple applications. In contrast, an unstructured system natively supports all types of complex queries as the whole index is available at the node evaluating the query [BHP+04]. For this reason, several flooding-based overlays are in widespread use despite their high overhead.

Structured and unstructured systems can be integrated. In the *Assisted Search with Partial Indexing* [ZH05] scheme, an unstructured system is assisted by a structured overlay. A random part of the resources are registered in a DHT-based index, which is used to find common interests among peers. Peers with similar interest are connected together in an unstructured overlay used for searching with a reduced TTL, thus, complex queries can be supported. Additionally, the index covers unpopular resources which can be found using the DHT.

### 3.2.3 Loosely structured systems

Between structured and unstructured systems, a separate class of systems is emerging. These are often called *loosely structured systems*. The definition of a loosely structured system varies. Androutsellis-Theotokis

and Spinellis [AS04] define loosely structured systems as systems where the location of the content is not completely specified but is affected by routing hints, such as in Freenet [Freenet]. Ganguly et al. [GCD04] use the term for a system with a topology evolving according to the shared content. In fact, most of the recent "unstructured" systems involve a degree of structure and, as Risson and Moors [RM06] point out, the structure is rather determined by the type of index. The structured versus unstructured routing taxonomy is becoming less useful [RM06]. We therefore do not aim to provide a definition of loosely structured systems but rather use the term where either the structure or the placement of index information is not strictly defined, while not completely arbitrary as in an unstructured system. Several of the solutions studied in this work can be categorized as loosely structured systems. In particular, the Parallel Index Clusters (PIC) and Parallel Search Clusters (PSC), having a strictly defined topology but with a free placement of indices, can be considered as a loosely structured system. These architectures have been described in Section 2.4.8.

## 3.3 Our contribution

Our review of the existing architectures shows that very few architectures distribute index information to more than a single node, especially in a way that improves the system performance. Several of the reviewed architectures are non-deterministic. This chapter studies uniform architectures. Thus, we avoid the concentration of load in a few nodes, as common in centralized and semi-centralized systems. The aim is to minimize the number of messages in the network while providing a reasonable search delay. Furthermore, we require the architectures to be deterministic, index allocation invariant and provide a method to support complex queries.

In Section 3.4 we propose a modified version of the PIC architecture called IPIC that allows clusters to be interconnected arbitrarily. Removing the requirement of full connectivity between clusters allows a reduction of the number of links in a network with $N$ nodes from $O(N^2)$ to $O(N)$. Searching in this architecture requires a different search algorithm than the one used in PIC. Therefore, we propose two new search algorithms based on random walk. The first algorithm visits every cluster in the network exactly once, while the second algorithm trades efficiency for a reduced delay. We analyze the performance using simulations. The architecture and a first set of simulations are published in [Bei07a]. In this present work, we repeat the simulations of [Bei07a] but with larger networks and with metrics compatible with the other metrics of this work. These new simulation results are unpublished. We also provide an unpublished backtracking method to be used in certain scenarios.

In Section 3.5 we propose a new architecture called Zone Indexing based on a ring topology. The architecture allows the balance of reactive and proactive operations to be adjusted dynamically according to the user behavior and network properties. We propose an algorithm for determining the optimal balance according to locally available

information. We propose a method to limit the delay using shortcuts. We perform simulations to study the performance of the algorithms. The architecture and simulations are published as part of [Bei10].

In Section 3.6 we present a fully proactive architecture based on a link-state routing protocol. In contrast to a routing protocol, however, we utilize the possibility to communicate directly between nodes, making index distribution efficient. We propose compressing the index information to reduce storage space and communication overhead. The performance is evaluated with simulation. The work is reported in [Bei07b].

The research presented in this chapter and the related publications are the sole work of the present author.

## 3.4 Clustered indexing architectures

The Parallel Index Clusters (PIC) architecture provides searching at an overhead that is significantly lower than search flooding when $1/N \leq f_s\Omega_s/f_i\Omega_i \leq N$. However, the requirements on topology may restrict the scalability in certain networks. This section describes the scalability problem and proposes solutions to it. Let us first define some concepts.

The *cluster topology* is a specified topology of connections between clusters that controls how the overlay topology is implemented. The cluster topology can be thought as an upper hierarchical layer of the overlay network. Let $E_s \subset E$ denote the set of search links and $E_i \subset E$ denote the set of index links in an overlay network $G = (V, E)$. We define the cluster topology as a graph $T = (C, K)$, where the vertices $C$ are clusters and the edges $K$ the connections between clusters. A cluster $C' \in C$ is a set of nodes $C' = \{v_1, v_2, \ldots, v_n\}$, where each node $v \in V$ belongs to a single cluster $Cluster(v) \in C$.

**Definition 3.1.** An overlay network $G = (V, E)$ *implements* a cluster topology $T = (C, K)$ if and only if a link $e = (v_1, v_2) \in E$ exists for all $k = (C_1, C_2) \in K$ from every node $v_1 \in C_1$ to some node $v_2 \in C_2$. A cluster topology can be implemented with either search links ($e \in E_s$) or index links ($e \in E_i$).

**Definition 3.2.** An overlay network $G = (V, E)$ *exclusively implements* a cluster topology $T = (C, K)$ if $G$ *implements* $T$ and additionally a connection $k = (Cluster(v_1), Cluster(v_2)) \in K$ exists for all $e = (v_1, v_2) \in E$, where $Cluster(v_1) \neq Cluster(v_2)$.

**Definition 3.3.** The *peer clusters* of node $v$ is the set of clusters $C_{peer}(v) = \{C' \in C \mid (Cluster(v), C') \in K\}$.

Thus, if a cluster $C_1$ is connected to a cluster $C_2$ in an implemented cluster topology, then for each node in $C_1$, there is a link to a node in $C_2$ in the overlay network. The specific node within the target cluster can be freely chosen, i.e. there are several possible ways to implement a cluster topology in the overlay network. If some link defined by the cluster

topology is missing in the overlay, then the cluster topology is not implemented. If there is a link between clusters that are not connected in the cluster topology then the cluster topology may be implemented but not exclusively implemented.

### 3.4.1 Scalability of the PIC topology

PIC uses one-hop queries as its search method: the node in the other cluster is queried in a single roundtrip with a minimal delay. Consequently, the architecture requires that each node is connected with a search link to at least one node in each cluster, i.e. PIC implements a fully connected cluster topology. The peer clusters of every node $v$ are $C_{peer}(v) = C \setminus Cluster(v)$. The number of search links in the network therefore increases in the order of $O(NC)$ as the number of clusters $C$ increases. Assuming the number of clusters increases in proportion to the network size $N$, the number of network links increases as $O(N^2)$.

In practical applications, the cost of maintaining a search link may be high. There is usually state information related to the link and some traffic is needed in order to keep the connection open and to check whether it is operating (e.g. with Hello packets or pings). Furthermore, the maximum number of links may be limited. For instance, the maximum number of concurrent open TCP connections is limited in several systems. The topology also makes PIC less error resilient: if a search link is down, the resources in a cluster becomes unavailable. Adding a new node is complicated as the node must be connected to all the existing clusters. Often the joining node is not aware of all the existing clusters in the system. Rearranging the network, for example, by introducing a new cluster, is difficult. Especially, if the clusters are operated by different administrative parties, as assumed in Section 4.5, there may be administrative costs and overhead related to the establishment of peering relations – it may be unfeasible to require connections between all parties. Because of these reasons, it would therefore be desirable that the clusters could be connected in an arbitrary topology instead of requiring every cluster to be connected to all other clusters.

### 3.4.2 IPIC: Indirectly connected Parallel Index Clusters

We propose an architecture, called IPIC (Indirectly connected Parallel Index Clusters), based on a modified PIC architecture. IPIC can use an arbitrary cluster topology. Instead of connecting all clusters together, each cluster has a limited number of neighbor clusters. A node is only required to maintain state information for a node in each of these neighbor clusters. The solution requires that the cluster topology is connected, i.e. that there is a path between each pair of clusters in the cluster topology. We further assume the possibility to use temporary links to send search messages between nodes that are not connected by permanent search links. A temporary link is stateless and corresponds to sending a single message (using e.g. UDP) to a node without establishing a permanent link.

Searching in the IPIC architecture is more challenging than in the ordinary PIC architecture. The direct query method of PIC cannot be used since a node does not know about all clusters in the network. Queries must be forwarded more than one hop. Basic flooding operates at the node-level (not the cluster-level). Hence, although an individual node is able to detect duplicate receptions, several nodes in the same cluster may still be visited. The goal is to visit each cluster only once. A visited node could distribute knowledge that the cluster is visited to all other nodes in the cluster, but this distribution adds overhead and is too slow to prevent all redundant visits. Thus, the search message itself must contain the information about visited clusters. With such a trace, flooding can be used as a search method in the IPIC architecture.

Flooding with a trace is a feasible search method but not a perfect one. A cluster can receive the same query several times if each copy reaches the cluster through a dissimilar path, as illustrated in Figure 3.1. As the trace is local to a given instance of the search query, the different instances are not aware of the path taken by other instances. We therefore need to minimize the number of instances. We propose two new search methods based on this observation: Stack-Based Random Walk (SBRW), and Replicating Stack-Based Random Walk (RSBRW). The IPIC architecture and the SBRW and RSBRW algorithms are published in [Bei07a]. We here extend the solution with an unpublished backtracking method and the changes necessary to support it.

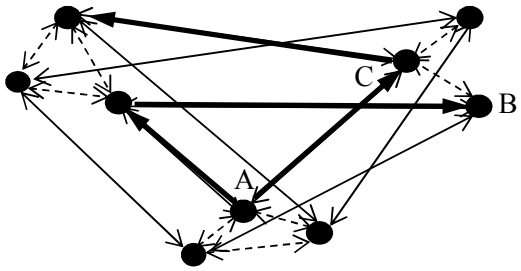


**Figure 3.1. Flooding with a trace starting from node A resulting in receptions by two nodes, B and C, in the same cluster.**

### 3.4.3 Stack-based random walk (SBRW)

*Stack-based random walk* (SBRW) is a search method that visits every node in any topology. It is especially feasible in a clustered network, where it operates in the cluster topology. Using SBRW requires each cluster to have a unique identifier and that the cluster identifier *Cluster*(*v*) is known to each node *v*.

SBRW extends random walk by adding two features:

1. The walker does not visit the same cluster twice (unless virtual links are prohibited).
2. The walker visits all clusters in the network.

To implement the first feature, the walker carries a set *T* of visited clusters – a *trace*. The walker is not forwarded to a neighboring node *v* if

*Cluster*($v$) $\in$ $T$. In order to save space, the trace can be represented as a Bloom filter as in [ZLZ+05] but in a reasonably sized network this is not needed. To implement the second feature, the walker contains a set $S$ of known but unvisited clusters. The entries in $S$ are tuples ($v$, *Cluster*($v$)), where $v$ can be any node in the unvisited cluster *Cluster*($v$). The set $S$ can be unordered (Set Based Random Walk), or ordered in the form of a stack (Stack Based Random Walk) or queue (Queue Based Random Walk). In the following, we assume that $S$ is a stack.

As the walker is received by a node $v$, the stack is updated by adding an entry ($w$, *Cluster*($w$)) for each neighbor $w$ for which *Cluster*($w$) $\notin S \wedge$ *Cluster*($w$) $\notin T$. Thus, $S$ contains neighbors in unvisited clusters that are scheduled for visit in the future. The node then removes the first entry $D$ = ($w$, *Cluster*($w$)) from $S$ and forwards the walker to $w$. The definition of "first" depends on the order (random, stack, or queue) of $S$, and determines whether the clusters are traversed in a random, depth-first, or breadth-first order. When $S$ is empty, the walker has visited one node in each cluster in the network. Before forwarding the walker, node $v$ updates the trace: $T \leftarrow T \cup$ *Cluster*($v$).

Using SBRW as a search method allows interrupting the search when a given number of matching resources have been found. The maximum path length can be limited using a TTL field, which makes the method non-deterministic. Information about matching resources can be stored within the walker or sent directly to the resource requester.

SBRW is a suitable search method at the node-level as well. However, the advantage of the trace is most obvious at the cluster-level. As the knowledge is stored in the walker instead of the node, the risk of visiting different nodes in the same cluster is eliminated.

### 3.4.4 Backtracking

In this section, we provide a backtracking mechanism that is used when virtual links are unavailable. Possible reasons for this situation include strict firewall policies and NATs with address-dependent filtering [AJ07]. The walker may then not be able to contact the following unvisited cluster directly from the current cluster. Instead, it must follow the traversed path back to a cluster that can reach the unvisited cluster. Consequently, the walker is only sent on the permanent links in the overlay topology.

The backtracking mechanism requires an additional field in the search message: the backtracking stack, $B$. This is a stack storing all visited nodes on the shortest path between the searching node and the current node.

The backtracking mechanism modifies the forwarding procedure of the SBRW algorithm. When a node $v$ examines the first entry $D$ = ($w$, *Cluster*($w$)) in $S$, it checks whether it has a direct link to $w$. If a link to $w$ exists, the walker can be forwarded normally using this link to $w$, whereas the entry $D$ is removed from $S$ and *Cluster*($v$) is added to $T$. Additionally, the address of the current node is added to the end of $B$. If no link to $w$ exists, the last node $v'$ in $B$ is removed and the walker is sent to $v'$. The trace $T$ is then not updated. When $v'$ receives the search

message, it can deduce that it is a returning message as it sees itself already in *T*, whereas it tries to find a link to the first entry in *S* using the above procedure. The procedure is repeated until a link to the first entry in *S* is found.

Because the walker must revisit visited nodes, backtracking adds overhead compared to SBRW. In practice, *S* must be a stack (producing depth-first searching) instead of a queue (producing breadth-first searching) to avoid the significantly higher overhead created when branching nodes are multiple times traversed in a breadth-first search. Random selection of entries in *S* unfeasible because of the high overhead.

### 3.4.5 Replicating stack-based random walk (RSBRW)

Like traditional random walks, the SBRW search method suffers from long delays due to the lack of parallel operations. As clusters are visited sequentially, the delay is proportional to the number of clusters in the network. In order to reduce the delay, parallel operations must be introduced. Sending multiple parallel walkers from the resource requester is not feasible, as all walkers will visit all clusters in the network. We therefore apply a method of periodical self-replication of the walker. In self-replication, a single walker becomes several walkers. This gives a degree of adaptability as the number of walkers increases with an increasing network size. Instead of blindly sending replicated walkers to random neighbors, we utilize the knowledge gathered before the replication to send walkers on different paths.

We implement self-replication by duplicating the walker into identical walkers. The only detail that differs between the walkers is the stack. The stack of the original walker is divided between the new walkers, thus, the new walkers are scheduled for visiting different clusters. The new walkers contain identical traces and will not visit the already visited clusters. The walkers are forwarded to the first node of their respective stack. The frequency of duplication (every hop, every second hop, etc) and the number of walkers generated in the replication are selectable parameters. In our experiments, we replicate the walker at each hop into two walkers.

Although the stack is divided between the new walkers, some overlapping visits appear. All clusters encountered after the replication will be visited by all replicated walkers. Since walkers cannot communicate, there is no way to assign a newly encountered cluster to a single walker. Neither can a walker indicate to other walkers that it has visited one of the newly encountered clusters. Consequently the overhead is higher than the overhead of the single walker in SBRW. Still, the overhead is significantly lower than in flooding. In [Bei07a] we refer to the replicating SBRW as a hybrid between flooding and SBRW, since it operates as flooding to a limited number of neighbors on the cluster level.

### 3.4.6 Topology construction

The inventors of PIC do not specify methods to construct and maintain the topology. The need to implement a given cluster topology is common to PIC and IPIC. In this section we describe a connecting procedure for

constructing a static PIC/IPIC network. We assume a fixed number of clusters. Further, we assume that each node knows its peer clusters and one node in each of them.

**Definition 3.4.** A cluster $C' \in C$ is *indegree balanced* if the indegree $d_{in}(v)$ is roughly the same for each $v \in C'$.

The procedure implements a specified cluster topology so that all clusters are indegree balanced. For balancing, the procedure requires a Hello protocol that distributes the current indegree of each node to all other nodes in the cluster. The Hello messages are generated periodically and distributed on the index links with the used index distribution algorithm. Alternatively, the indegree can be transported as an additional field in index messages. As a consequence, each node stores the degree of all nodes in the cluster in a *degree table*.

The purpose of the connecting procedure is to connect a joining node to the node with the lowest indegree in a given cluster. Initially the joining node has a list of one known node $w$ in each of its peer clusters. For each of its peer clusters, the joining node $v$ sends a Connection Request to $w$. Node $w$ forwards the request to the node $w'$ with the lowest indegree according to its degree table. The request can be recursively forwarded if node $w'$ knows about a less loaded node. A path trace in the Connection Request is used for loop prevention. If recursive operation is restricted (e.g. by NATs) node $w$ can send a Connection Redirect message to $v$, indicating the node $w'$ to which $v$ should reconnect in an iterative way. If a node $w'$ is unaware of nodes with lower indegree, a connection reply is sent to the joining node $v$ and the search link is set up between $v$ and $w'$.

### 3.4.7 Hierarchical PIC and IPIC

A semi-centralized network is created by connecting each ordinary node with a search link and an index link to a super node. The super nodes are interconnected with a search flooding architecture. Replacing the search flooding architecture with a PIC or IPIC architecture creates a two-layer hierarchical PIC or IPIC network, respectively. This allows attaching low-capacity or restricted nodes to an overlay of higher-capacity nodes.

The proposed connecting procedure can be reused to construct a hierarchical PIC/IPIC network with a balanced number of ordinary peers per upper-layer node. A joining ordinary node sends a Connection Request to a known node $w$ in the PIC/IPIC network. Using the previously defined iterative or recursive search, the node $w'$ with the lowest indegree is found. The joining ordinary node $v$ establishes an index link and a search link to $w'$. Indegrees of links from an ordinary node and another cluster should be counted separately to better balance load.

The node $w'$ can later send $v$ a Reconnect Request to reallocate $v$ to a node $u$ with a lower indegree if the difference $d_{in}(w') - d_{in}(u)$ is large. Upon receiving this message, $v$ restarts the connecting procedure to node $u$. The index of $v$ does not need to be removed during the reconnection.

### 3.4.8 Performance of architectures based on PIC

We compare the performance of IPIC to PIC and standard flooding using simulation. The simulations in this work represent a generic use of the architectures. Simulations modeling small operator networks are presented in [Bei07a]. We compare five schemes:

1. Flooding in a power-law topology with the average degree of $D = 4$.
2. Flooding in a power-law topology with the average degree of $D = 8$.
3. PIC in a fully connected cluster topology.
4. IPIC with the SBRW search method in a power-law cluster topology with the average degree of 4.
5. IPIC with the RSBRW search method in a power-law cluster topology with the average degree of 4.

The two flooding architectures are included for comparison. The degrees are selected to give realistic upper and lower bounds for the performance of flooding, whereas a typical Gnutella topology has an average degree of $D = 5.5$ [LCC+02]. IPIC assumes that temporary links are available, i.e. backtracking is not used.

We denote the number of nodes in the network with $N$. In PIC and IPIC, the network is clustered into clusters with $N_c = 10$ nodes each, in total $C = N / N_c$ clusters. PIC and IPIC implement the overlay topology by forming overlay links with the proposed connecting procedure so that clusters are indegree balanced. For this purpose a Hello protocol operates with messages sent at 60 s intervals. We do not count these messages as they are used for topology construction and do not affect search performance. For all schemes we test both a forwarding and non-forwarding variant. In the non-forwarding variant, the search is terminated when the first matching resource is found while the forwarding variant searches the whole network. The search scope is not limited by a TTL. All schemes are deterministic, and the simulations give a success ratio $R_{success} > 99.5\%$. The reason that it is less than 100% is because of the dynamically added and removed resources: index updates are distributed with a fixed period of $1/f_i = 360$ s with no event based updates. Nodes are static during the simulation. Each scenario is simulated with 10 simulation runs, each 36360 s with an initial 3600 s stabilization period, and the results are averaged. Search requests are generated with exponentially distributed intervals with an average of 360 s. Resources are uniformly distributed between the nodes and requests are made for random resources. Resources are not replicated unless otherwise stated. Matching results are reported to the resource requester directly.

Figure 3.2 shows the number of links in the network. In PIC, the number of links increases exponentially with the network size as each node must be connected to a node in all other clusters. This high number of links is the reason for the development of IPIC. In all other schemes the number of links grows linearly with the network size. The exact number depends on the average degree and number of clusters. IPIC requires a number of links that corresponds to a random power-law topology with the average degree of $D = 4$.

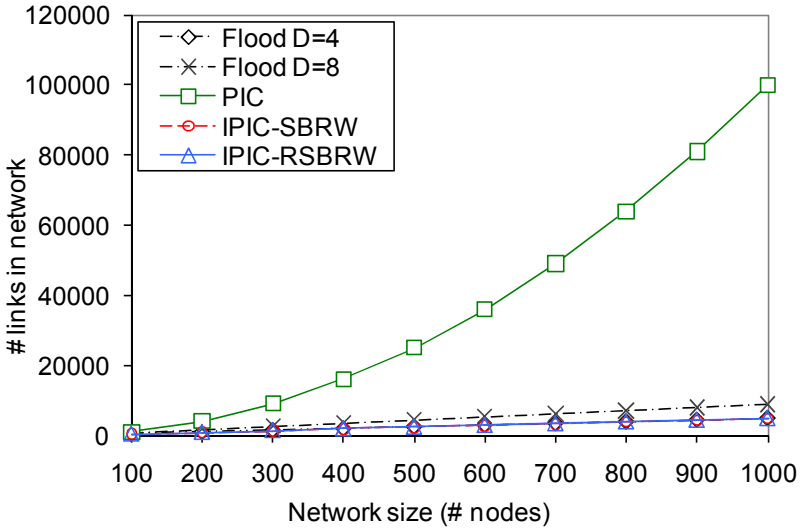


**Figure 3.2. Number of links in the network.**

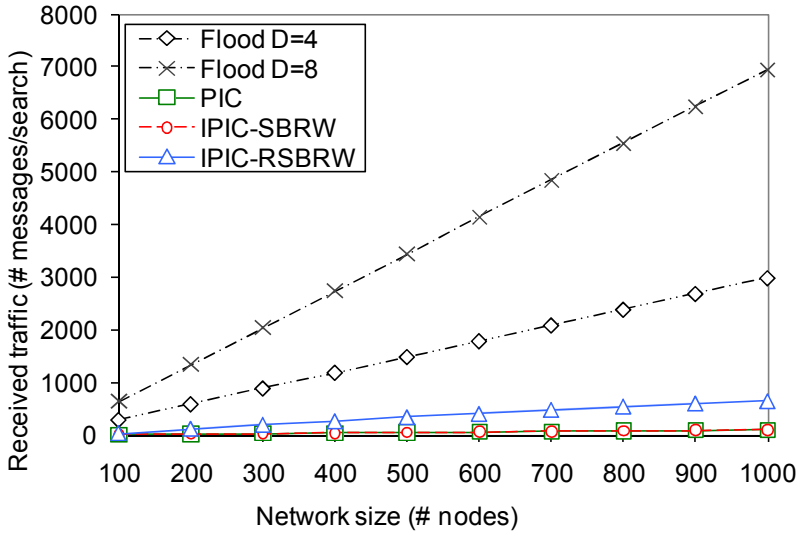


**Figure 3.3. Scalability in terms of $M_s$.**

Figure 3.3 shows $M_s$, i.e. the average number of receptions of a search message (excluding the response message) per search in a network of increasing size. As both PIC and IPIC-SBRW visit each cluster exactly once, $M_s = C - 1$, as expected. For flooding the number of receptions depends on the average degree $D$ of the topology, and according to Hypothesis 2.1 it is expected to be $M_s \approx (D-1)N$, which also the simulation shows. Analytical results from IPIC-RSBRW have not been obtained but our simulation shows that IPIC-RSBRW generates about 6 to 7 times more messages than PIC and IPIC-SBRW in the given scenario.

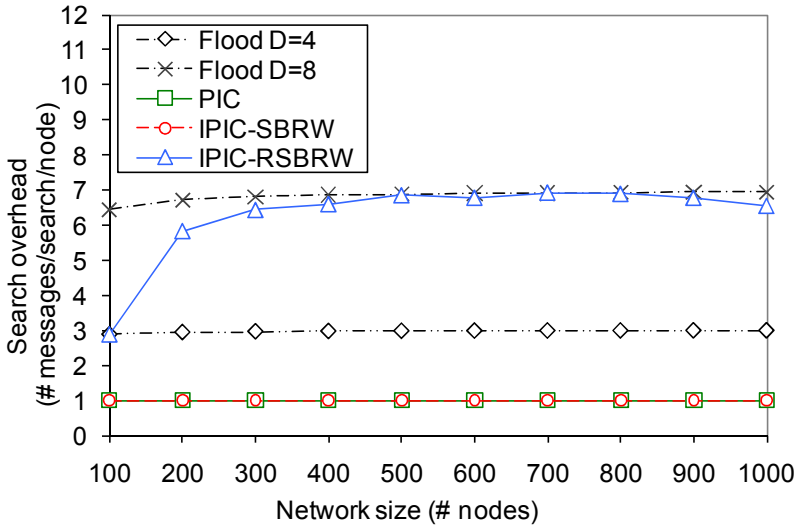


**Figure 3.4. Scalability in terms of $\Omega_s$.**

Dividing $M_s$ by $N_s$ gives the overhead $\Omega_s$ shown in Figure 3.4. For PIC and IPIC, $N_s = C$, while for flooding $N_s = N$. As expected, PIC and IPIC-SBRW are optimal $\Omega_s = 1$. For flooding, $\Omega_s \approx D - 1$. IPIC-RSBRW has a low overhead for small networks but the overhead stabilizes at $\Omega_s \approx 7$ when $C > 30$. This indicates that knowledge about visited clusters is useful only during the first replication steps, after which the knowledge becomes aged. In large networks, the frequency of replication therefore needs to be reduced. One approach is to decrease the frequency of replication with the number of hops. The overhead does not reveal the full truth, however: although the overhead is larger than the overhead of flooding in a Gnutella topology, the $M_s$ is lower because of indexing.

Figure 3.5 shows the delay for locating the first copy of a resource assuming a link delay of $D_L = 20$ ms. The delay includes the delay of the reply message, which is sent on a single hop. In PIC, the delay corresponds to one roundtrip, $2D_L$. The delay is independent of $N$. In SBRW, all clusters are visited sequentially with a total of $C + 1$ hops. For the first hit the delay is $D_L(C+1) / 2$ on average. The delay thus increases linearly with the network size. In all clustered architectures, the measured delay is slightly lower because of the possibility to find the requested resource in the same cluster as the requesting node. The delay in flooding is proportional to the network diameter, which depends on the average degree. In [CL02] the diameter of a power-law random graph with exponent $\gamma > 3$ and $D > 1$ is shown to increase logarithmically with the network size. RSBRW trades efficiency for a reduced delay. The delay is slightly higher than in flooding.

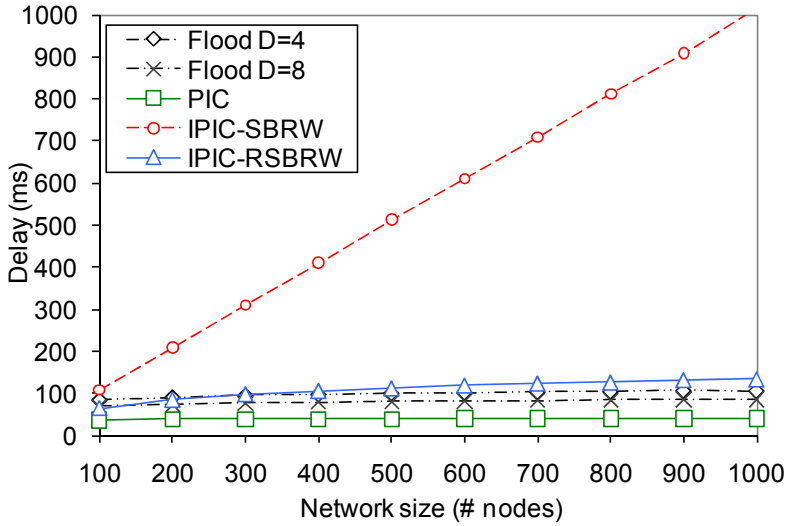


**Figure 3.5. Scalability in terms of search delay.**

For IPIC-RSBRW the replication frequency has a central role. A given performance that is between IPIC-SBRW and IPIC-RSBRW, both in terms of message overhead and delay can be obtained by adjusting the frequency of replication.

### 3.4.9 Replicated resources

When several copies of a resource are available, the sequential search of SBRW (and to some extent in RSBRW) becomes an advantage. The search can be interrupted when one or a given number of matching resources have been found, which reduces the message overhead and search delay. We evaluate networks with $N$=500 nodes and each resource available at 1 to 30 random nodes. The resource is thus replicated to 0.002% to 6% of the nodes.

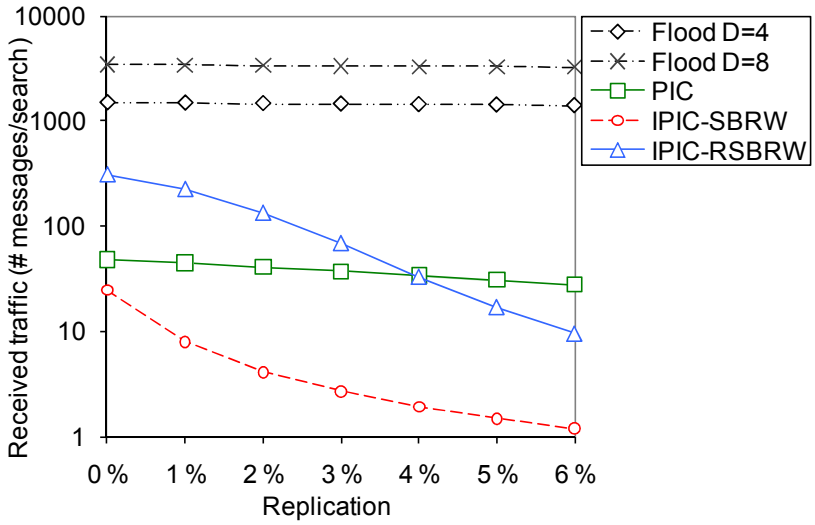


**Figure 3.6. Effect of replication on $M_s$.**

Figure 3.6 shows the number of received messages per search for the non-forwarding variants. Because of the large differences in traffic, a logarithmic scale is used. We can see that flooding and PIC cannot benefit from replication whereas the traffic in both IPIC variants decreases to a level lower than PIC. The forwarding variant is not shown: the redundant copies do not reduce the traffic since the whole network must be examined.

Figure 3.7 shows the search delay for finding the first matching resource. Since only the first match is required, both the forwarding and non-forwarding variants behave similarly (only the non-forwarding is shown). The reduction in search delay for IPIC-SBRW is significant – the existence of a few copies halves the delay, whereas a 3% replication reduces the delay to a level comparable to flooding. IPIC-SBRW, given its low overhead, is particularly useful for replicated resources. Also IPIC-RSBRW benefits from replication in terms of delay.

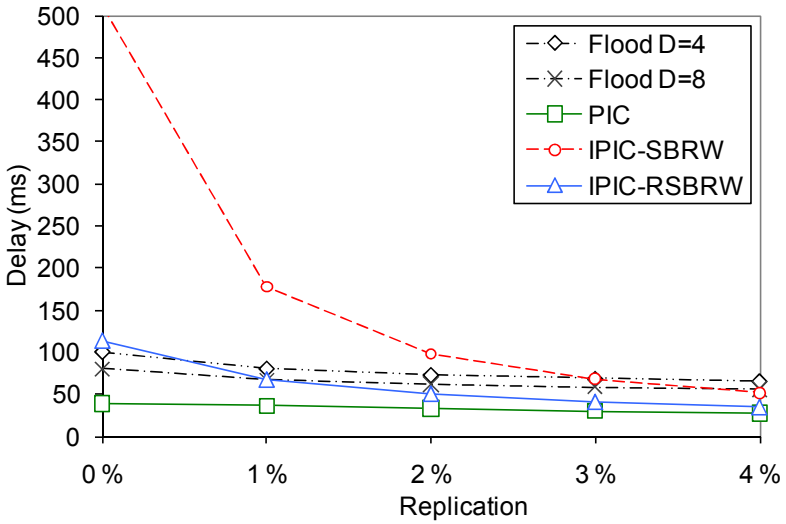


**Figure 3.7. Effect of replication on search delay.**

### 3.4.10 Difference in cluster sizes

So far we have assumed an identical number of nodes $N_c$ in each cluster. If the number of nodes per cluster varies, it is expected that the load is unevenly distributed among the participating nodes. We simulate six scenarios with different allocations of nodes to clusters. The total number of nodes $N = 500$ and the number of clusters $C = 10$ are identical in all scenarios. The number of nodes in the scenarios is presented in Table 3.1. Figure 3.8 shows the average and maximum number of received messages per generated search. As expected, the traffic of the most loaded node increases as the variation in size increases. The average traffic is constant. In Figure 3.9 the maximum traffic is normalized in respect to the average traffic. The figure shows that all clustered architectures react similarly to the increasing difference in cluster sizes. However, the IPIC architectures show a slightly higher traffic for the most loaded node.

**Table 3.1. Number of nodes per cluster in the examined scenarios.**

| Cluster | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Scenario 1 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Scenario 2 | 40 | 40 | 40 | 50 | 50 | 50 | 50 | 60 | 60 | 60 |
| Scenario 3 | 30 | 30 | 40 | 40 | 50 | 50 | 60 | 60 | 70 | 70 |
| Scenario 4 | 10 | 20 | 30 | 40 | 50 | 50 | 60 | 70 | 80 | 90 |
| Scenario 5 | 10 | 15 | 20 | 30 | 40 | 60 | 70 | 80 | 85 | 90 |
| Scenario 6 | 5 | 10 | 15 | 20 | 25 | 75 | 80 | 85 | 90 | 95 |

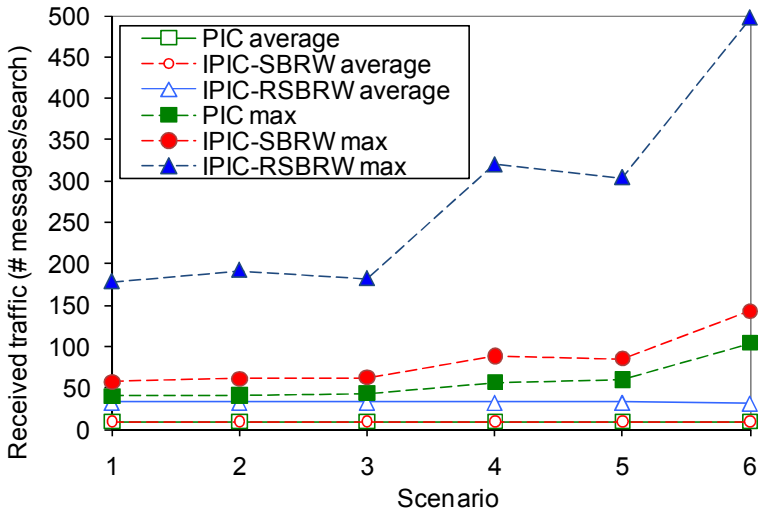


**Figure 3.8. Average and maximum number of received messages per search under different cluster configurations.**

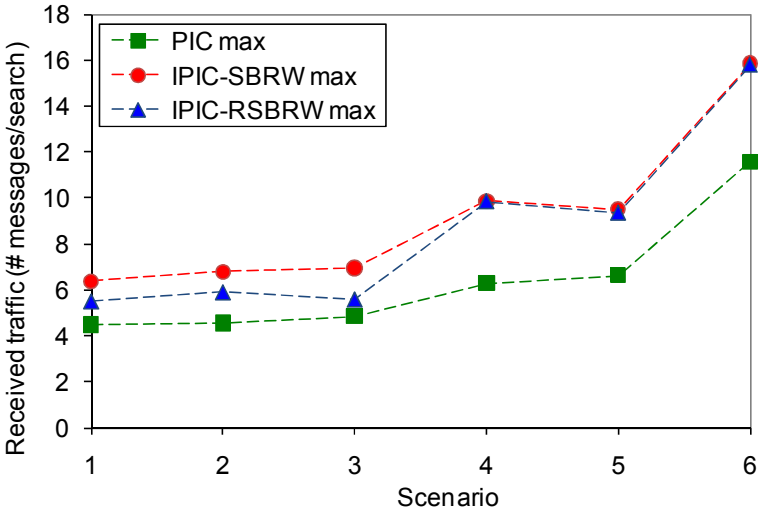


**Figure 3.9. Normalized maximum number of received messages per search under different cluster configurations.**

### 3.4.11 The cost of index distribution

The efficiency of PIC and IPIC does not come without a cost. The index must be distributed to all nodes within a cluster. An efficient topology, such as a ring or fully connected topology, of index links allows the index to be distributed with $M_i = N_c - 1$ messages in a cluster containing $N_c$ nodes, i.e. with $\Omega_i = 1$. If the index needs to be updated $f_i$ times per time unit on average, the index message traffic per node is $F_i = f_i(N_c-1)$. Since we have determined $\Omega_i$ and $\Omega_s$ for the various architectures, we can use the Search/Index Space model to calculate the optimal cluster size for a given search/index ratio.

### 3.4.12 Related architectures

For completeness, we also examine whether the Parallel Search Cluster (PSC) architecture could be modified to allow indirect cluster connectivity in the same way as in IPIC. In this approach, index updates are distributed to one node in each cluster using SBRW or RSBRW. All nodes in a cluster are queried in a search.

Index updates are not time-critical but they must be reliably distributed to all clusters. It is also important that only one (or very few) nodes per cluster receive the index update, since otherwise the efficiency is reduced by storing multiple copies. The main problem, however, is that this type of architecture is not index allocation invariant. Because each consequential update of a node $v$ may follow a different random path, the indexing nodes $\mathbb{P}_v$ constantly changes and the previous versions of $v$'s resource descriptor remain in the system until they expire. Generally, index allocation invariance is difficult to provide with any multi-hop index distribution method without a rather strict structure of the network. Given this limitation we conclude that this architecture is not feasible in practical implementations. The IPIC architecture, however, does not suffer from this problem as the set of indexing nodes $\mathbb{P}_v$ is stable.

### 3.4.13 Comparison

Table 3.2 compares the properties of the discussed architectures. All the clustered architectures (PIC, PSC, IPIC) provide a significant performance increase, both in terms of overhead and delay, compared to flooding, provided that the search/index ratio is sufficiently high. While PIC and PSC are able to provide optimal performance with correct cluster size, the IPIC architectures allow an arbitrary cluster topology and with a linear growth of state information but at the cost of either a longer delay or a larger overhead. The architectural choice therefore depends on whether these properties are required.

**Table 3.2. Properties of the PIC, IPIC-SBRW and IPIC-RSBRW architectures and comparison to standard flooding.**

| Architecture | Search flooding | PIC | IPIC-SBRW | IPIC-RSBRW |
|---|---|---|---|---|
| Index topology | - | Arbitrary connected topology within cluster | Arbitrary connected topology within cluster | Arbitrary connected topology within cluster |
| Search topology | Random power-law | A link from each node to at least one node in all other clusters | A link from each node to at least one node in a subset of the other clusters | A link from each node to at least one node in a subset of the other clusters |
| Index distribution method | - | Any (e.g. ring or star within cluster) | Any (e.g. ring or star within cluster) | Any (e.g. ring or star within cluster) |
| Search distribution method | Flooding | One-hop queries to a node in each cluster | SBRW | RSBRW |
| Class | Reactive uniform | Hybrid proactive-reactive uniform | Hybrid proactive-reactive uniform | Hybrid proactive-reactive uniform |
| Links in network | $O(N)$ | $O(N^2)$ | $O(N)$ | $O(N)$ |
| Index entries per node | $O(1)$ | $O(N_c) = O(N/C)$ | $O(N_c) = O(N/C)$ | $O(N_c) = O(N/C)$ |
| Index message scalability, $M_i$ | - | $O(N_c) = O(N/C)$ | $O(N_c) = O(N/C)$ | $O(N_c)$ |
| Search message scalability, $M_s$ | $O(DN)$ | $O(C) = O(N/N_c)$ | $O(C) = O(N/N_c)$ | $O(kN)$ |
| Search delay, $T_s$ | $O(\log N)$ | $O(1)$ | $O(C) = O(N/N_c)$ | Unknown (nearly logarithmic) |
| Benefit from replicated resources | Low | Low | High | Medium |

## 3.5 Zone Indexing – a proactive-reactive hybrid architecture

Architectures combining proactive and reactive operations can be implemented in several ways, including the PIC, PSC and IPIC architectures. Such architectures can be made optimal using the Search/Index Space model. Despite their good performance, the mentioned architectures suffer from crucial practical problems. Firstly, it is difficult to construct and maintain the topology as nodes join and leave. To our knowledge, no automatic algorithm has been proposed. The closest attempt to create such an algorithm is [Dun], which however lacks several critical properties and has not been experimentally or theoretically verified. Secondly, as the performance depends on the cluster size, the overlay must be updated when the network size and user behavior changes. This update may require moving nodes between clusters and rearranging the links both within clusters and between clusters. The overlay network must itself be able to detect the need for restructuring. We therefore develop a new architecture implementing the Search/Index Space model. Our goal is to develop an architecture that

- provides near-optimal search and index distribution algorithms,
- provides automatic construction of the overlay network,
- dynamically adapts to the current network size,
- dynamically adapts to the current search/index ratio,
- provides a search delay less than O($N$), and
- can be implemented in practice with reasonable effort.

Additionally, the architecture must support our general requirements of

- uniform index allocation,
- support for complex queries,
- determinism, and
- index allocation invariance.

This research has resulted in an architecture called *Zone Indexing,* which is presented in [Bei10].

### 3.5.1 Topology

Like Chord [SMK+01], Zone Indexing organizes the nodes into a ring. Each node $v$ knows the node that precedes it on the ring, its predecessor $Pred(v)$, and the node that succeeds it on the ring, its successor $Succ(v)$. The successor of the successor is called the second successor and is denoted $Succ_2(v) = Succ(Succ(v))$. Correspondingly, the $k$th successor is denoted $Succ_k(v)$. The $k$th predecessor of $v$ is $Pred_k(v)$. A node actively maintains knowledge about its first successor only but several predecessors can be learned through indexing.

In a ring topology, searching can be performed reactively by simply forwarding a search request along the ring until it reaches the resource requester again or encounters a match. The search message overhead is then O($N$) and the delay is O($N$). To improve search performance, additional state information and index replication are needed. Here, Chord and Zone Indexing take dissimilar approaches, which determine the difference in the type of supported queries and the performance.

Chord places a resource-specific index in a single node which is identified by a key, while Zone Indexing replicates a node-specific index to several nodes. The searching node in Chord needs to know the key of the requested resource, whereas Zone Indexing allows for complex searches using any logic based on the metadata of the resource. The ability to support complex searches is the main advantage of Zone Indexing. The ring structure in both approaches is similar and can be reused in scenarios where both lookups and complex searches are needed.

Zone Indexing divides the ring into overlapping zones. The number of nodes within the zone of node $v$ is called the *zone size* of node $v$ and is denoted $Z_v$. The zone of node $v$ of size $Z_v$ contains the nodes $\{v, Succ_1(v), Succ_2(v), \ldots, Succ_{Zv-1}(v)\}$. The algorithm allows each node to select a different zone size, which can vary dynamically. Various algorithms for selecting the zone size can be developed – we later propose one such algorithm.

### 3.5.2 Index distribution algorithm

Each node $v$ maintains a unidirectional index link to its successor. Index updates are sent with a time-to-live value of $R = Z_v$. The local index of $v$ is consequently distributed to all nodes in node $v$'s zone. Thus, $\mathbb{P}_v = \{v, Succ_1(v), Succ_2(v), \ldots, Succ_{Zv-1}(v)\}$. Using our previous definition, a node $w$ stores the index of the nodes $\mathbb{N}_w = \{u \mid w \in \mathbb{P}_u\}$, i.e. of all nodes in whose zone $w$ resides. For searching (described later), each node maintains a unidirectional search link to its *border node*. The border node, $Border(v)$, of a node $v$ is defined as the closest predecessor not included in the remote index of $v$. Thus, $Border(v) \notin \mathbb{N}_v$.

Figure 3.10 depicts an example network from the perspective of a node E. All nodes use the same zone size $Z = 4$. The index of node E is distributed to the nodes $\mathbb{P}_E = \{E, F, G, H\}$. Node E stores the index of nodes $\mathbb{N}_E = \{B, C, D, E\}$. The border node of E is node A, as this is the closest node that is not indexed by E. The zones overlap, and therefore for example, node D distributes its index to a different set of nodes, $\mathbb{P}_D = \{D, E, F, G\}$.

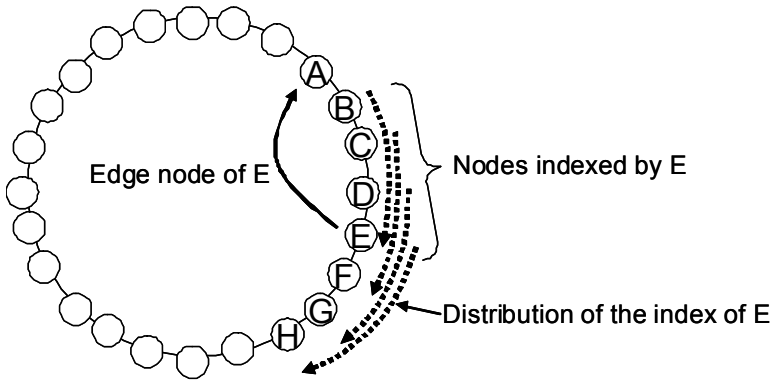


**Figure 3.10. An example Zone Indexing network from the perspective of node E.**

An index update message ($v$, $Pred(v)$, $R$, $E$, $T$) sent by node $v$ contains the address of the sender $v$, the address of the sender's predecessor $Pred(v)$, the remaining hops $R$, a set of index entries $E$, and a timeout value $T$. The remaining hops is initialized by the sender to $R = Z_v - 1$. The timeout specifies the length of time the index entries are stored in a node's remote index and is initialized to $k$ (an application specific constant) times larger than the periodical index update interval: $T = k / f_{i,refresh}$. The format for the index entries depends on the application.

A node $w$ receiving the index update message through a node $u$ first updates its predecessor $Pred(w) = u$. This accounts for a potential new node that has joined the ring as a new predecessor. Node $w$ stores information about the sender $v$: the predecessor $Pred(v)$ is stored in the neighbor cache $P_{neighborcache}$ and the index entries $E$ in the remote index $E_{cache}$. If $R > 0$, the message is then forwarded to $Succ(w)$ with $R = R - 1$. The expiration timer for the received entry and the timeout timer for the predecessor are restarted. The reception algorithm is summarized in Figure 3.11.

```
1: receive index (v, P, R, E, T) via u
2: Pred(w) ← u
3: P_neighborcache(v) ← P
4: E_cache(v) ← E
5: restart timer for E_cache(v) with T seconds
6: restart timer for u with T seconds
7: if R > 0 then
8:     send index (v, P, R-1, E) to Succ(w)
9: end if
```

**Figure 3.11. Index reception algorithm of node w.**

To distribute the index to $N_i$ nodes, $M_i = N_i$ messages are required. The overhead of the indexing distribution algorithm is therefore $\Omega_i = 1$, $\omega_i = 0$ according to Equation (2.29) and the algorithm is optimal.

### 3.5.3 Search algorithm

Search requests are forwarded in reverse direction compared to the index updates. The search algorithm utilizes border nodes in order to avoid examining the same index multiple times. A unidirectional search link is maintained to the currently chosen border node. Each time the neighbor cache is updated (e.g. on each received index update) the border node is re-evaluated. The algorithm for determining the border node is presented in Figure 3.12. If the border node changes, the search link to the former border node is closed and a search link to the new border node is created.

```
1: b ← Pred(w)
2: while P_neighborcache(b) is defined do
3:     b ← P_neighborcache(b)
4: end while
5: if b ≠ Border(w) then
6:     remove search link to Border(w)
7:     Border(w) = b
8:     create search link to Border(w)
9: end if
```

**Figure 3.12. Algorithm for determining the border node of node w.**

Before presenting a more efficient search method, we present the idea using a simple form of searching. The simplest form of searching is implemented by using border nodes only. A search message ($v$, $Q$, $S$) contains the address $v$ of the resource requester, the query $Q$ and a list of stop nodes $S$. In the simplest form of searching, a single stop node, the resource requester itself, is defined: $S = \{v\}$. The search request is forwarded by a node to its border node, which in turn forwards the request to its border node, until the whole ring is traversed. On each step, the request is forwarded to the first predecessor with new index information. To determine when the search has traversed the ring, the set of stop nodes is examined on each hop. The search finishes when it either (1) reaches a node $w$ that has an entry for a stop node in its neighbor cache, or (2) when a stop node is the border node of $w$. In other words, the search is forwarded until

$$S \cap (Border(w) \cup \mathbb{N}_w) \neq \varnothing. \quad (3.1)$$

Consequently, the search covers all indices between the resource requester and the stop node. In the simplest form, the search covers all indices in the ring. We define the *search path* of node $v$ as the ordered set of nodes that a search request generated by $v$ will traverse, including node $v$ itself.

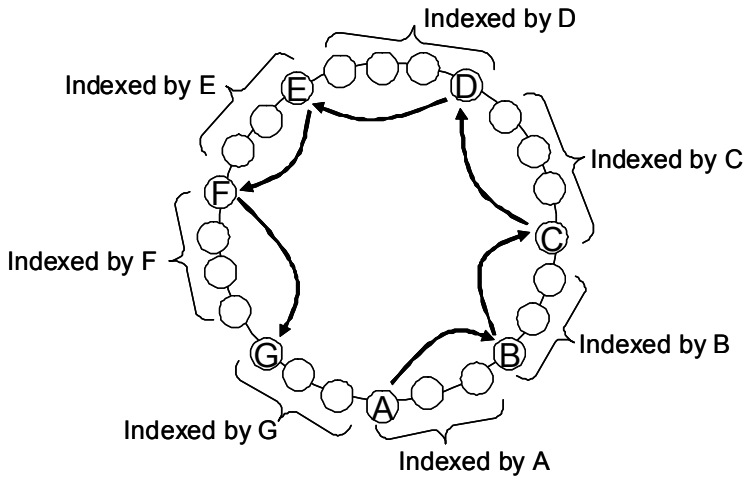


**Figure 3.13. Forwarding a search request in a Zone Indexing network.**

Figure 3.13 shows an example of searching. The search request generated by node A is forwarded to node B, which forwards to its border node C. The request is forwarded until it reaches node G, whose border node is A. As node A is the stop node, the search finishes. The search path of node A is {A, B, C, D, E, F, G}.

### 3.5.4 Experimental setup for simulations of Zone Indexing

We implement Zone Indexing in our simulator [Bei09] to prove the concept and to validate our mathematical analysis. Differently to our paper [Bei10], we here present the simulation results together with the corresponding analysis of each separate mechanism.

The simulated system consists of $N$ nodes, which are static unless otherwise mentioned. The resources are modified (added or removed) at exponentially distributed random intervals at a frequency of $f_{i,update}$ = 1/2000 $s^{-1}$, and each update triggers an index update. Additional periodical updates are sent at the frequency $f_{i,periodic}$ = 1/2000 $s^{-1}$. The total index update rate is $f_i = f_{i,update} + f_{i,periodic}$ = 1/1000 $s^{-1}$. Search requests are performed by uniformly randomly selected nodes for resources that at the time of the request exist in the network. The popularity of requested resources follows a Zipf distribution[3] where the query rate of the $i$th most popular resource is proportional to $i^{-\alpha}$ with $\alpha = 1.0$. Search requests are generated at exponentially distributed random intervals at a frequency of $f_s$. To obtain the desired search/index ratios $r = f_s / f_i$, the search frequency $f_s$ is adjusted. The search examines all indices, i.e. the search message is forwarded after a matching resource is found. In all simulations, resources are non-replicated and located at random nodes selected from a Zipf distribution with parameter $\alpha = 1.0$. Transmission delays between simulated nodes are based on pair-wise delay measurements between DNS servers using the King method [GSG02]. The average unidirectional link delay is 90 ms with a standard deviation of 66 ms. The collection of statistics is started after a 3000 s settling period, whereafter the simulation is continued until 5000 search requests have been generated. The measured success ratio is $R_{success} > 99.9\%$ in all static scenarios (a slight variation is due to dynamically added and removed resources). The system is considered deterministic.

### 3.5.5 Performance of the basic version of Zone Indexing

As the index update is distributed to all nodes within the zone with $Z$ - 1 messages, the frequency of received index messages per node is

$$F_i = f_i (Z - 1) \quad . \tag{3.2}$$

To find all matching resources, the search request is forwarded along the search links so that the whole ring is traversed. Assuming that all nodes use the same zone size $Z$, the search message is forwarded to

[3] We performed the simulations with both uniform and Zipf distributions. No differences were observed. This independence of distribution comes from the fact that the indices of all nodes are examined. The results from the Zipf distribution are shown.

$$N_z = \lceil N / Z \rceil \approx N / Z \tag{3.3}$$

nodes. The number of search messages received per time unit is therefore

$$F_s = f_s \left( \frac{N}{Z} - 1 \right). \tag{3.4}$$

Figure 3.14 shows the frequency of received index and search messages in a network with $N = 1000$ nodes for varying zone sizes (all nodes use the same zone size). The used search/index ratio of $r = 2$ is obtained by selecting $f_s = 1/500$. The figure shows the simulation results; the analytic results from (3.2) and (3.4) are practically indistinguishable from the simulation results if plotted on the same graph. The index traffic increases with an increasing zone size while the search traffic decreases.

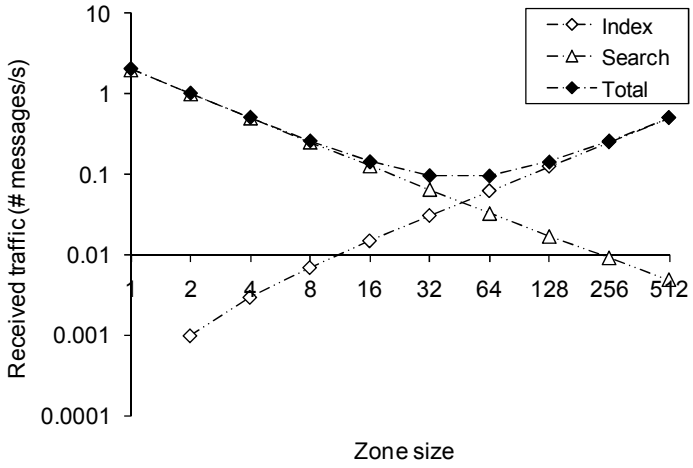


**Figure 3.14. Received index and search messages vs. zone size.**

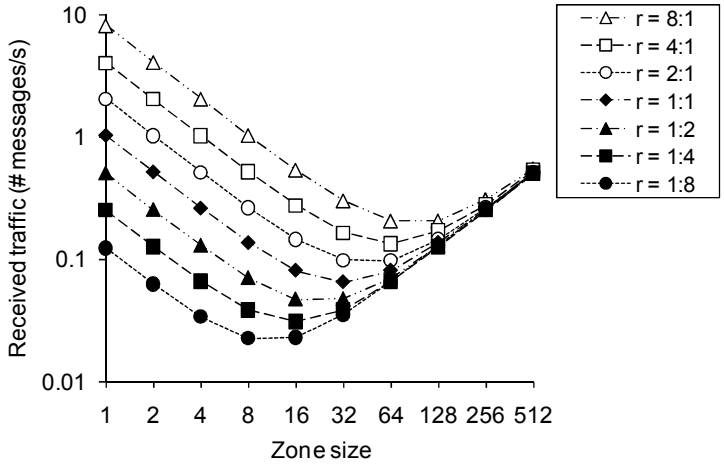


**Figure 3.15. Received messages vs. zone size for different search/index ratios.**

In Figure 3.15 the total traffic is shown for search/index ratios varying between $r = 1/8$ and $r = 8$. As suggested by the Search/Index

Space model, the traffic is minimized by a certain zone size for each value of $N$ and $r$.

To locate all matching resources, $N_z = N/Z$ nodes are queried according to (3.3). If each overlay link has a delay of $T_{link}$, the maximum delay for finding a resource is

$$\hat{T}_{search} = T_{link} N_z = T_{link} \left\lceil \frac{N}{Z} \right\rceil \approx \frac{N T_{link}}{Z}, \tag{3.5}$$

including the search reply. This is also the delay for finding all matching resources. The delay increases linearly with the size of the network.

If only one matching resource exists, the average number of queried nodes is $N_z/2$, assuming the resource is located at any node with uniform probability. The average delay for finding a matching resource is then

$$\overline{T}_{seearch} = \frac{T_{link}}{2} \left\lceil \frac{N}{Z} \right\rceil \approx \frac{N T_{link}}{2Z}. \tag{3.6}$$

Figure 3.16 depicts the analytic and simulated average search delays in an $N = 1000$ node network where the average link delay is $T_{link} = 90$ ms. As the delay is independent of $r$, only $r = 2$ is shown. A larger zone size decreases the search delay. The delay is high, which motivates using an optimal zone size and adding parallel operations (in Section 3.5.10). The delay increases linearly with the network size (not shown).

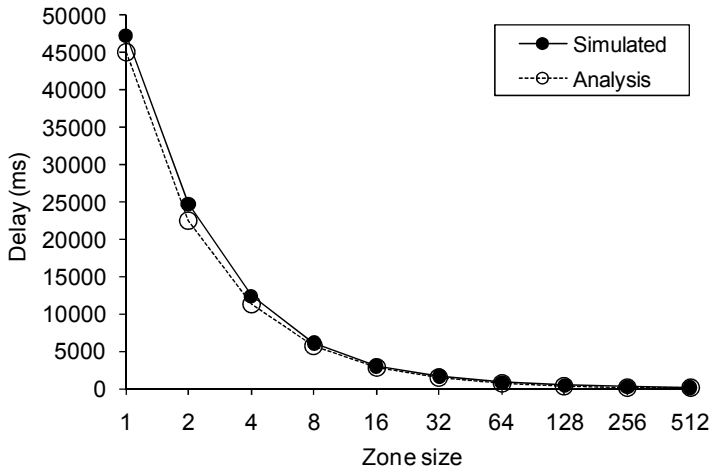


**Figure 3.16. Average search delay when no shortcuts are used.**

### 3.5.6 Performance under optimal zone size

Zone indexing implements the Search/Index Space model. A zone consists of one node in each row in the matrix. Because of the overlapping clusters, these nodes can reside in two adjacent columns, as depicted in Figure 3.17. A search request is distributed to one node in each column in the matrix. Consequently $P = Z$ and $R = N / Z$. Both searching and indexing are optimal: $\Omega_i = \Omega_s = 1$.

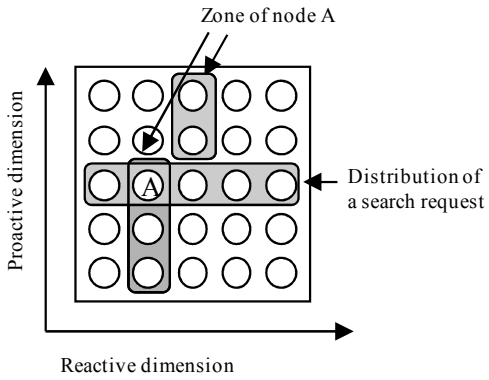


**Figure 3.17. Zone indexing in the Search/Index Space model.**

The analysis of optimal zone size follows the one in Section 2.4.9. By deriving the total traffic $F = F_i + F_s$ given by (3.2) and (3.4) with respect to $Z$, we obtain the optimal zone size

$$Z_o = \sqrt{rN} \ . \tag{3.7}$$

When the zone size is optimal, the frequency of received index messages is

$$F_i = f_i\left(\sqrt{rN} - 1\right) = \sqrt{f_i f_s N} - f_i \ , \tag{3.8}$$

and the corresponding frequency of received search messages is

$$F_s = f_s\left(\sqrt{N/r} - 1\right) = \sqrt{f_i f_s N} - f_s \ . \tag{3.9}$$

The total frequency of received messages when the zone size is optimal is consequently

$$F = F_i + F_s = 2\sqrt{f_i f_s N} - f_s - f_i \ . \tag{3.10}$$

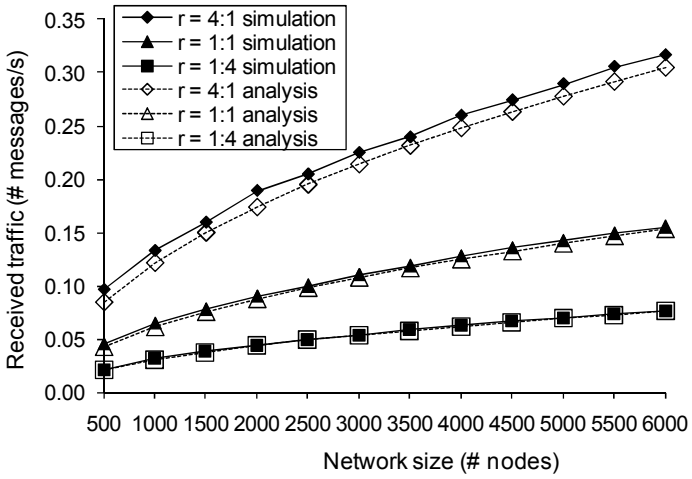


**Figure 3.18. Received message vs. network size when zone size is optimal.**

The frequency of received messages for optimal zone sizes is shown in Figure 3.18. The network size varies between $N = 500$ and $N = 6000$

nodes and the search/index ratio takes the values $r = 4$, $r = 1$, and $r = 1/4$. The analytic and simulation results plotted in the same graph correspond well to each other. The traffic grows in the order of $O(\sqrt{N})$ when the zone size is optimal. A high search/index ratio increases the traffic.

Inserting (3.7) into (3.3) gives the number of nodes queried when the zone size is optimal:

$$N_{z,o} = N / Z_o = \sqrt{N} / \sqrt{r} \quad . \tag{3.11}$$

Inserting (3.7) into (3.5) gives the maximum delay for finding a resource when the zone size is optimal:

$$\overline{T}_{seearch,o} = T_{link} N_{z,o} = \frac{T_{link} \sqrt{N}}{\sqrt{r}} \quad . \tag{3.12}$$

Correspondingly, inserting (3.7) into (3.6) gives the average delay for finding a single random resource when the zone size is optimal:

$$\overline{T}_{seearch,o} = \frac{T_{link} N_{z,o}}{2} = \frac{T_{link} \sqrt{N}}{2\sqrt{r}} \quad . \tag{3.13}$$

Figure 3.19 shows the average search delay when the zone size is optimal. The traffic grows in the order of $O(\sqrt{N})$. An increasing proportion of searches to index updates decreases the search delay. The simulation results coincide with the analytic results.

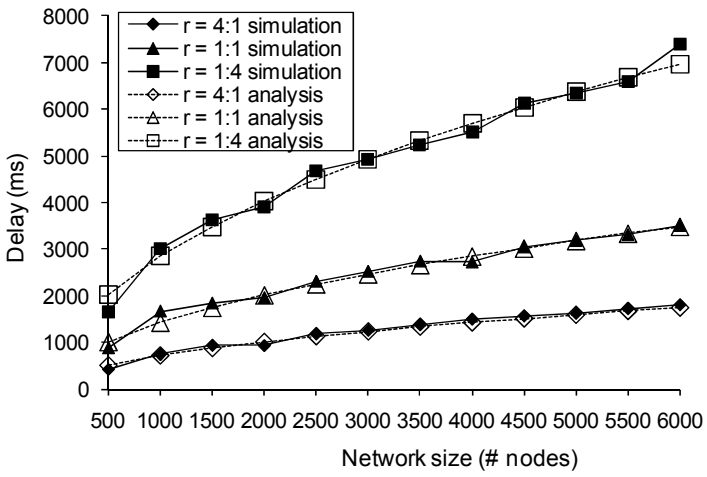


**Figure 3.19. Average search delay vs. network size when zone size is optimal.**

**Theorem 3.1.** *If the zone size is optimal, the search load equals the index load.*

**Proof.** Inserting (3.8) into (2.2) gives

$$L_i = \sqrt{f_i f_s N} \ .$$

Correspondingly, inserting (3.9) into (2.1) gives

$$L_s = \sqrt{f_i f_s N} \ . \ \square$$

### 3.5.7 Algorithm for dynamic control of zone size

The goal of the Zone Indexing architecture is to be able to adjust the number of index replicas so that it is close to optimal. The overlapping of zones allows the adjustment to be made with low effort, much lower than the effort of rearranging the links and clusters in a PIC or PSC network as the optimal number of clusters changes. The change of zone size only affects the initial value of the hops left field in the index update message.

As Equation (3.7) indicates, the optimal zone size depends on the network size $N$ and the search/index ratio $r$. Unfortunately, both variables are difficult to measure. A node is able to determine its local frequency of generated search messages and index updates and calculate a local search/index ratio, but this represents the behavior of only one node, which can be very different from the average behavior. Moreover, the process of obtaining a global node count does not scale. However, Theorem 3.1 states that $L_i = L_s$ when the zone size is optimal. When $L_s > L_i$ the zone size needs to be enlarged and vice versa. The load is easy to measure locally. The local index load $L_{i,v}$ of node $v$ is calculated by measuring the interval $T_i$ between processed (generated and received) index messages. The load is then the inverse $1/T_i$ of the time interval. To give a longer time perspective and better stability, an exponentially weighted average with parameter $\alpha$ ($0 < \alpha < 1$) is used to include the load of earlier measurement periods:

$$L_{i,v}(t) = \alpha / T_i + (1-\alpha) L_{i,v}(t-1) \, . \qquad (3.14)$$

Our simulations showed that the selection of $\alpha$ has a marginal effect on the performance. We use $\alpha = 0.1$. The search load is measured in a similar way.

To reduce the variation in zone size between neighboring nodes, we approximate the network-wide average load by averaging the local load of a larger number of nodes. The choice of zone size of node $v$ directly affects the index load of $v$'s successors: when $v$ changes its zone size, the load balances of its successors change. Therefore, we use the load of node $v$'s current successors as a basis for the zone size decision. As the search load is dependent on the index load (through the zone size), it is sufficient to adjust the index load in order to affect the $L_i/L_s$ ratio.

To measure the load of the successors, the index update sent by node $v$ contains $v$'s current local search load and index load. On forwarding the index update, each node adds their current search and index load to these fields. The last node in the zone sends an acknowledgement message back to node $v$ containing the averaged index and search loads. Node $v$

thereby learns the average load of the nodes in its zone. If the search load is higher than the index load, the zone size is increased by one. If the search load is lower than the index load, the zone size is reduced by one. If the difference is small (defined by a threshold) the zone size is unmodified. As the zone size changes with a fixed maximum rate of ±1 nodes per index update, the change rate is sufficiently slow to give stability. The acknowledgement message further doubles as an indication that the index update was correctly delivered. The acknowledgement message adds a constant overhead of $\omega_i = 1$. As stated by Theorem 2.1 this does not affect the optimal zone size.

### 3.5.8 Avoidance of local redundancy

The Search/Index Space model requires the nodes to be uniformly distributed in the space. Applying this requirement to Zone Indexing, translates into a requirement that all nodes should use the same zone size to guarantee optimal performance in a network-wide perspective.

A particular case of non-optimal performance results from local redundancy. This is caused by a large variation in the zone size between consecutive nodes. To illustrate the situation, consider the example in Figure 3.20. The zone sizes are $Z_A = 4$, $Z_B = 2$, and $Z_C = 2$. Node D will select B as its border node as it is the first predecessor not included in the neighbor cache. When a search message is forwarded by node D to the border node B, the index entries of node A are examined twice: at both D and B. Thus, the distribution of A's index entries to node D is redundant. Node A should select a zone size of $Z_A = 3$ or less.

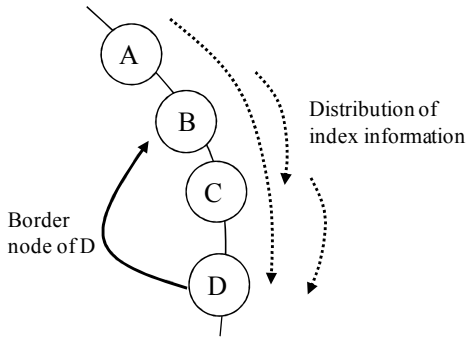


**Figure 3.20. Redundancy caused by large variation in zone size.**

Local redundancy is avoided if

$$Z_n \le Z_{Succ(v)} + 1 \tag{3.15}$$

for each node $v$ in the network. As periodical messages are received only from the predecessor, it is more practical to express this condition in an equivalent form:

$$Z_n \ge Z_{\Pr ed(v)} - 1 . \tag{3.16}$$

In some cases it may be beneficial to allow local redundancy in a limited form. To increase the speed of adaptation when the target zone size

changes rapidly, it may be justified to allow a larger variation in the zone size between neighboring nodes. The maximum variation is then a system parameter $Z_{AllowedDifference}$ generalizing (3.16) to

$$Z_v \geq Z_{\Pr ed(v)} - Z_{AllowedDifference} . \quad (3.17)$$

For symmetry, an application can additionally consider an upper limit for the allowed zone size, whereas the condition is

$$Z_{\Pr ed(v)} - Z_{AllowedDifference} \leq Z_v \leq Z_{\Pr ed(v)} + Z_{AllowedDifference} . \quad (3.18)$$

### 3.5.9 Evaluation of the algorithm for zone size control

We first test the adaptation speed of the algorithm using a network with $N = 2000$ nodes. All nodes are started simultaneously with an initial zone size of $Z_v = 1$. We implement the algorithm described in Section 3.5.7 for zone size control with a parameter of $\alpha = 0.1$. We limit the variation in zone size using both an upper and a lower limit as defined in (3.18). The cases $Z_{AllowedDifference} = 1$ and $Z_{AllowedDifference} = 2$ are examined. Figure 3.21 shows the development of the zone size during 110000 s. The vertical bars indicate the measured standard deviation. The final zone size is reached faster with $Z_{AllowedDifference} = 2$ with a cost of a slightly higher deviation, especially during the settling time. In the stable state, both cases behave similarly. As the zone size can change only with ±1 node on each index update and updates are sent in 1000 s interval, the smallest possible time for reaching the final size $Z = 36$ is $Z / f_i = 36000$ s. The measured convergence time is between 50000 s and 60000 s.

One should be reminded that the situation of all nodes starting from scratch is artificial and only shows the adaptation speed. In a practical implementation, a joining node learns the current zone size from its predecessor, whereafter the adjustment reacts to relatively slow changes in the network.

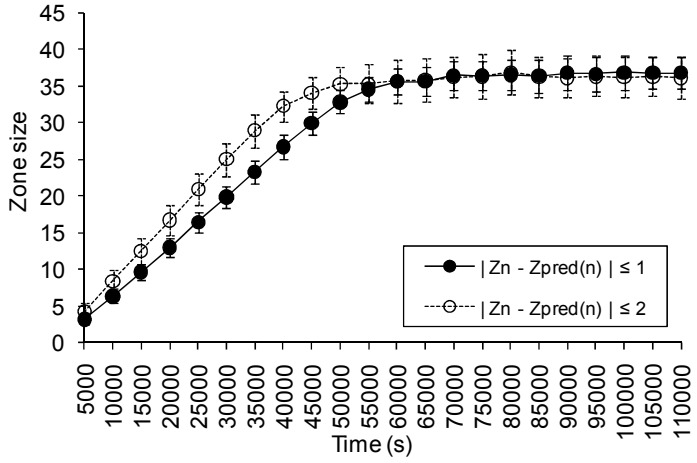


**Figure 3.21. The adaptation of zone size in time.**

We then examine the precision of the algorithm. Figure 3.22 shows how close the dynamically adjusted zone size is to the analytic optimal size 100000 s after starting the nodes. Networks have a size varying between $N = 250$ and $N = 2000$ nodes and a search/index ratio of $r = 2$ and $r = 1/2$. The simulations represent a scenario of $Z_{AllowedDifference} = 1$. The resulting size is relatively close to the optimal one although a slightly larger zone size is obtained.

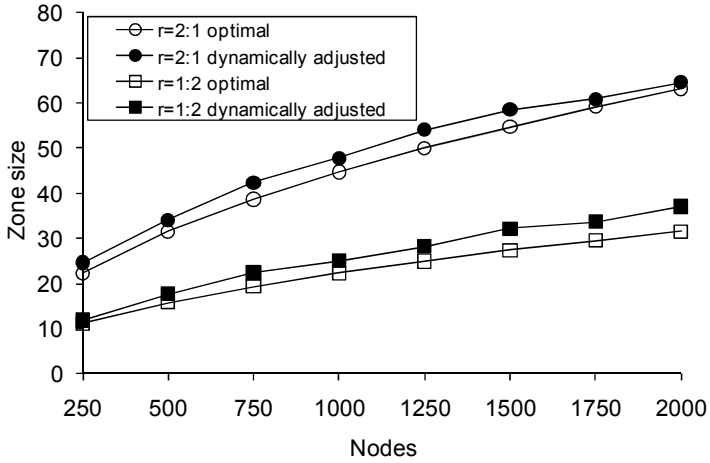


**Figure 3.22. Dynamically adjusted zone sizes compared to optimal ones.**

### 3.5.10 Searching using shortcuts

Using a single search message traversing the whole ring, as in the simple form of the search algorithm, creates a long search delay in large networks. To reduce the delay, we divide the ring into *search sectors* which are examined in parallel with separate search messages. The ring is divided by *shortcut nodes* which, together with the resource requester, separate consecutive search sectors. The more sectors the ring is divided into, the lower the delay, but the more state information in the form of shortcuts is required.

A separate search request is sent to each shortcut node, which forwards the request according to the normal search algorithm. These are sent in addition to the normal search request originating from the resource requester. Each search request is forwarded along the ring until it encounters one of the stop nodes listed in the request, as defined in (3.1). Consequently, the resource requester $v$ has several parallel search paths, each starting in $v$. All the shortcuts and the resource requester itself are given as stop nodes in the search. For comparison, recall that in the basic version of Zone Indexing, the only stop node is the resource requester, resulting in a single search sector covering the whole ring. The reason that all shortcut nodes must be included as stop nodes is that the resource requester cannot determine in which order the shortcut nodes appear in the ring.

Consider the example in Figure 3.23. The resource requester A and the shortcuts D and F divide the ring into three search sectors. The search

message contains the list of stop nodes {A, D, F}. The search message is processed by node A and forwarded to A's border node B as normally. A copy of the search message is also sent to each of the shortcut nodes, who process the query and forward the message to their border nodes. Node C detects that its border node D is one of the stop nodes and finishes the search. Also node G stops forwarding as its border node A is one of the stop nodes. Node E stops forwarding as it has the index of node F, which is a stop node. The search paths of node A are {A, B, C}, {A, D, E}, and {A, F, G}.

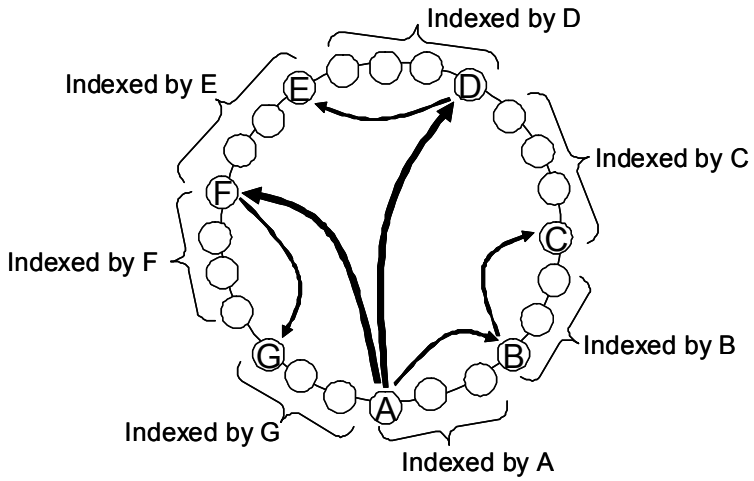


**Figure 3.23. Searching using shortcuts.**

Any known node can be used as a shortcut node. For instance, if a search reply has been received from a node *w*, node *w* can be stored as a shortcut. If zone indexing is used together with Chord, the fingers can be shortcuts.

### 3.5.11 Algorithm for delay control

This section proposes a method to populate the list of shortcuts in order to obtain a controlled delay. It can be used either as the only way to create shortcuts or to complement another method.

A desired maximum delay is obtained by adjusting the maximum length of a search path. To implement controlled delay, we add two fields to the search message. The search message contains a counter $H$ initialized to zero, counting the number of hops the search message is forwarded. The maximum delay is specified in terms of a maximum allowed hop count, $H_{max}$. When $H$ reaches $H_{max}$, a Shortcut message containing the address of $u$ is sent to the resource requester, which can establish a shortcut to node $u$. The counter $H$ is then reset and the forwarding is continued, whereas more shortcuts may be generated by other nodes.

Assuming all nodes have the same zone size $Z$, the maximum length of a search sector is $ZH_{max}$ nodes, and the network is divided into at least $N / ZH_{max}$ search sectors. With an average delay $T_{link}$ of each overlay link the maximum delay for finding a resource is

$$\hat{T}_{search,shortcuts} = H_{\max} T_{link} \tag{3.19}$$

and the average delay for finding a single copy of a resource is

$$\overline{T}_{search,shortcuts} = H_{\max} T_{link} / 2 . \tag{3.20}$$

As a new node is unaware of any shortcuts, the delay of the first search may be long. However, once some shortcuts are known, the list of shortcuts only needs to be maintained.

Topology changes cause new shortcuts to be allocated as the distance between two shortcut nodes can become too large. The maximum number of shortcuts can be limited to a value $N_s$ in order to reduce the state information. When a new shortcut is established and the limit is reached, a randomly selected shortcut is removed. The randomness is used to distribute the shortcuts evenly around the ring since a node cannot know the distance between different shortcuts. Since a search sector contains $ZH_{max}$ nodes, a network of $N$ nodes needs to be divided into at least

$$N_{s,\min} = N / ZH_{\max} \tag{3.21}$$

search sectors. If the $N_s < N_{s,min}$, all generated shortcuts cannot be maintained, with the consequence that new shortcut messages are continuously triggered. In this exceptional case, the application needs to increase $H_{max}$ as it detects a high number of continuously received Shortcut messages.

If all shortcuts are used and evenly distributed, the expected average delay is

$$\overline{T}_{search,allshortcuts} = \frac{T_{link} N_z}{2(N_s + 1)} . \tag{3.22}$$

### 3.5.12 Performance of delay control

The delay control algorithm is simulated in a network with $N = 3000$ nodes and a search/index ratio of $r = 1$. All nodes use an optimal zone size of $Z_o = 55$ according to (3.7). The shortcut interval is varied between $H_{max} = 5$ and $H_{max} = 55$. The number of available shortcuts is varied between $N_s = 0$ and $N_s = 16$. At $N_s = 0$ shortcuts are disabled. The delays are presented in Figure 3.24 and Figure 3.25.

Figure 3.24 reveals that the delay is most dependent on $N_s$. Already a small number of shortcuts reduce the delay. The delay further decreases for each added shortcut, i.e. as $N_s$ increases. The delay is slightly higher than the analytic delay given by (3.22). The difference between the analytic and the simulated delays is due to the fact that randomly distributed shortcuts are not perfectly evenly distributed. As $H_{max}$ is reduced, the distribution of shortcuts becomes more even, but still not perfect.

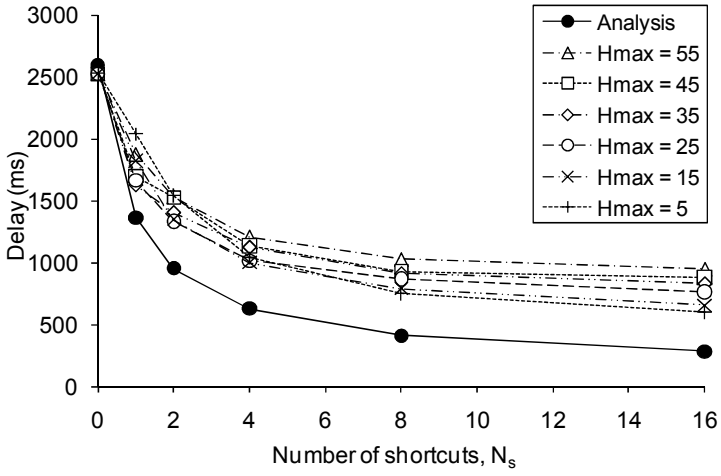


**Figure 3.24. Average search delay when shortcuts are enabled.**

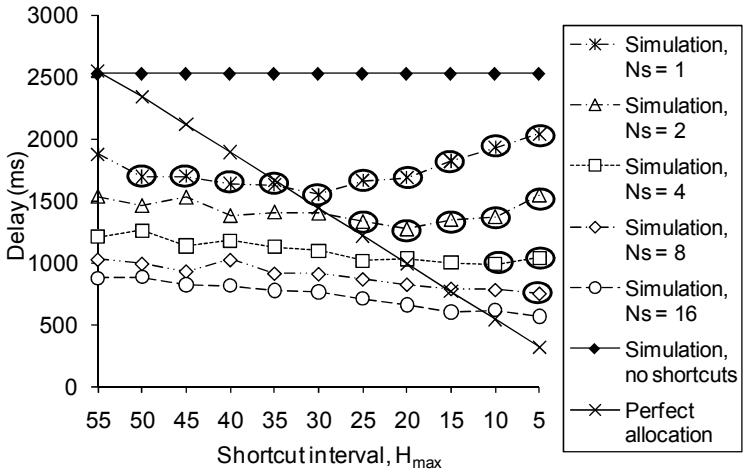


**Figure 3.25. Effect of the shortcut interval on the average search delay.**

Figure 3.25 shows that the parameter $H_{max}$ affects the delay only slightly. For comparison the figure also presents the perfect allocation given by (3.20), which is expected when exactly $N_s = N_{s,min}$ shortcuts are evenly distributed. Perfect allocation is not achieved in a practical scenario. Instead, the delay depends almost solely on the number of available shortcuts, because all available shortcuts will be used even though they do not fulfill the spacing requirement given by $H_{max}$. When the minimum number of shortcuts $N_{s,min}$ given by (3.21) is exceeded, the additional shortcuts are distributed randomly across the network reducing the delay further. This distribution takes place when the network size is varied (e.g. during network construction) and before the index cache is fully populated.

The scenario is selected to also demonstrate the particular case where the number of available shortcuts is too low to fulfill the spacing requirement set by $H_{max}$. This occurs when $N_s < N_{s,min}$ as given by (3.21). The affected points are encircled in Figure 3.25. For these points, the

delay no longer decreases with a decreasing $H_{max}$ and may instead slightly increase as shortcuts become unevenly distributed and the allocation of shortcuts never converges.

Figure 3.26 shows the message overhead added by the use of shortcuts. In the simulated case, the increase in traffic is between 0% and 5%, which can be considered as a low cost for the reduced delay. An exception appears in the particular case when $N_s < N_{s,min}$ when the shortcuts are never stabilized, causing continuous transmission of Shortcut messages with an resulting overhead of up to 9%. This can be avoided by increasing $H_{max}$.

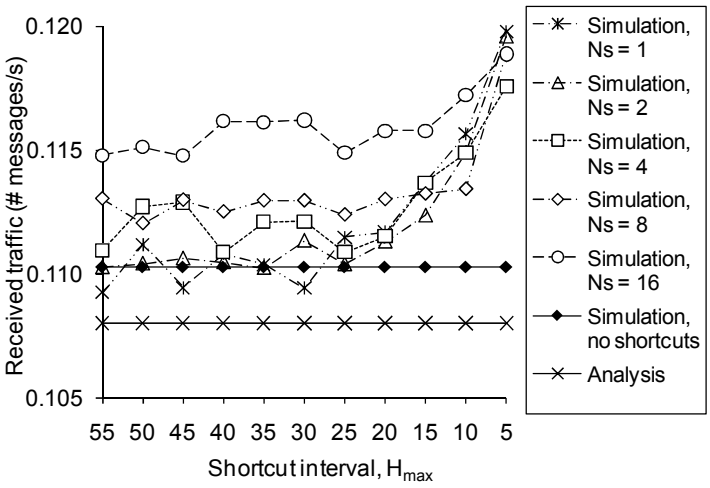


**Figure 3.26. Received messages when shortcuts are enabled.**

Table 3.3 presents how the number of shortcuts affects the traffic and delay reduction using averages of values for which $N_s < N_{s,min}$. For $N_s = 1$ the value for $H_{max} = 1$ is used as no values satisfy the condition.

The choice of $N_s$ is a tradeoff between delay and message overhead: increasing $N_s$ adds overhead linearly while reducing the delay asymptotically toward zero. A practical application could adjust $N_s$ until a desired maximum delay is obtained, at the same time modifying $H_{max}$ according to (17).

**Table 3.3. Additional traffic and delay reduction caused by shortcuts.**

| Number of shortcuts | Traffic increase | Traffic increase per shortcut | Delay reduction | Delay reduction per shortcut |
|---|---|---|---|---|
| $N_s = 1$ | -0.30 % | -0.30 % | 30.86 % | 30.86 % |
| $N_s = 2$ | 0.24 % | 0.12 % | 43.99 % | 21.99 % |
| $N_s = 4$ | 1.53 % | 0.38 % | 56.57 % | 14.14 % |
| $N_s = 8$ | 2.32 % | 0.29 % | 64.70 % | 8.09 % |
| $N_s = 12$ | 4.35 % | 0.36 % | 69.10 % | 5.76 % |
| $N_s = 16$ | 5.16 % | 0.32 % | 70.96 % | 4.43 % |

### 3.5.13 Algorithm for topology maintenance

For creating and maintaining the ring topology, any suitable method can be used, including the one used in Chord. In contrast to Chord, Zone Indexing does not require a node to join at a specific position in the ring – a requirement generating O(log $N$) messages per joining node in Chord. Because of the lower demands, we propose a more lightweight method to join the ring, requiring only O(1) messages. Our simulations have further shown that the proposed algorithm is very robust under high churn rates.

To create a new ring topology, a node $v$ selects itself as both its predecessor and successor: $Pred(v) = Succ(v) = v$.

To join an existing ring, the joining node must know any node $w$ already in the ring, for instance, through a bootstrap mechanism or from previous sessions. The fact that a node can join the ring at any position allows the node to measure the delay to a set of nodes and select $w$ as the node with the lowest delay, thereby reducing the overall delay of the system. The joining node $v$ inserts itself between $w$ and $Succ(w)$. To accomplish this, node $v$ sends a Join message to node $w$. Upon receiving the Join message, node $w$ sets its successor $Succ(w) = v$ and answers with a Join Reply containing its previous successor $u$ and its current zone size $Z_w$. When $v$ receives the Join Reply, it updates its successor $Succ(v) = u$ and optionally uses the current zone size of $w$ as an initial value for its own zone size. The following index update informs $u$ about its new predecessor $v$ using the normal index update algorithm.

A node $v$ can leave the ring gracefully by sending its predecessor $u$ a Leave message containing $Succ(v)$. The node $u$ receiving a Leave message sets its successor to the node indicated in the message $Succ(u) = Succ(v)$. The new successor of $u$ is informed about the change in the following index updates.

### 3.5.14 Algorithm for topology maintenance in exceptional cases

If a node $v$ has received no index updates via its predecessor $w$ during a specified timeout, the node $w$ is assumed failed and it is added to node $v$'s list of failed nodes $F_n$. A node $w$ can later be removed from the list $F_n$ if $v$ receives any message from $w$. Node $v$ starts recovering by sending Join messages to its predecessors according to increasing distance, starting from the failed predecessor. The addresses of the preceding nodes are obtained from the neighbor cache. These Join messages are sent and handled like normal Join messages. However, the node $v$ does not update its successor to the node $u$ indicated in the Join Reply if it has declared node $u$ as a failed node.

In the exceptional case when a large number of consecutive nodes fail simultaneously, none of the known predecessors replies. Node $v$ can then send a Join message to any known node $u$, such as a shortcut node. In such a case, both the joining node $v$ and the known node $u$ can be followed by several nodes. In fact, one of them is probably followed by the rest of the ring. Therefore, the chains of the nodes succeeding $v$ and $u$ must be merged. More precisely, as one of the chains may contain the rest of the ring, the chains must be interleaved until one of the chains

ends. Interleaving implies inserting the nodes following *v* as every second node in the chain following *u*.

Interleaving is implemented in the following way. A node *v* receiving a Join Reply from a node *w* sends its successor (if it has one) a Rejoin message containing its successor *Succ*(*v*). The node receiving the Rejoin message sends a Join message to the indicated node, i.e. to *Succ*(*v*). The Join Reply triggered by this Join may cause another Rejoin message to be sent, and the process is repeated until a node without a successor is reached.

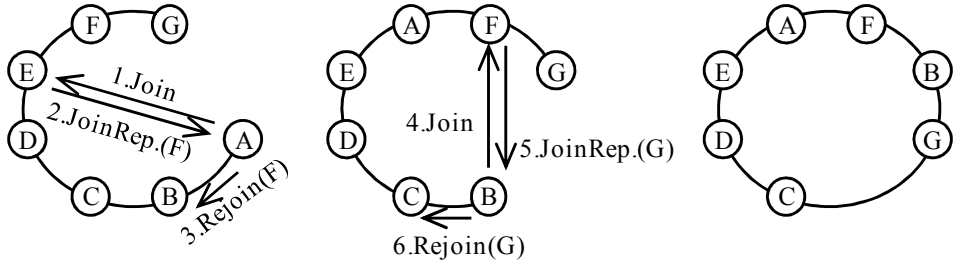


**Figure 3.27. Example of interleaving to repair the overlay.**

The procedure is illustrated in Figure 3.27 using an example. The attempts of node A to join its predecessor have failed. Node A, however, knows another node, E, to which it sends a Join message. Since both nodes A and E are succeeded by a chain of successors, these chains must be interleaved. As the node A has a successor (*Succ*(A) = B), it sends node B a Rejoin message indicating node F, i.e. the node in the received Join Reply. Node B then sends a Join message to F. Upon receiving the Join Reply, node B sends a Rejoin message to C, indicating node G. Node C consequently sends node G a Join message. The ring is corrected. The successor of node G, if included in the Join Reply, is dead and any further attempt of C's successor D to join G will fail and D will therefore keep its current successor.

The process of interleaving also gives a solution to a situation where a node *v* is not aware of any other nodes. In this situation, the only known node is the node *v* itself, and as a last resort the node sends a Join message to itself. This causes a sequence of events that wraps the remaining ring around node *v* like a snowball, interleaving the layers together and resulting in a repaired ring.

### 3.5.15 Performance under churn

To test the topology maintenance algorithm, we simulate nodes joining at exponentially distributed random intervals $1/\lambda = 1/f_{churn}$. The average interval is varied between $1/f_{churn} = 125$ s and $1/f_{churn} = 16000$ s. Nodes stay in the system an exponentially distributed random period of average length $1/\mu$. The parameter $\mu$ is selected according to Little's law $\mu = N/f_{churn}$ so that the average number of nodes $N = 1000$ is constant. As $N$ is constant, the leave interval equals the join interval. The index update frequency is $f_i = 1/500$ s$^{-1}$ and the search frequency is $f_s = 1/500$ s$^{-1}$ giving

a search/index ratio of $r = 1$. The number of shortcuts is $N_s = 10$ with $H_{max} = 5$. We examine the case where all nodes leave gracefully and the cases where 10% and 50% nodes fail without notification. A node declares its predecessor failed if no index updates are generated or relayed through the predecessor within a timeout $T_{timeout} = 1/f_i = 1000$ s.

The effect of churn on success ratio is shown in Figure 3.28. When nodes exit without notification, the success ratio starts dropping when $1/f_{churn} \leq T_{timeout}$. Therefore, the periodical index update frequency should be selected sufficiently high to match the expected churn rate. In order to ensure reliable operation under churn, $f_i$ should preferably be selected $f_i > 2f_{churn}$. When nodes exit gracefully, the drop in success ratio is lower. Additional simulations (not shown) showed that the search/index ratio $r$ does not affect the success ratio.

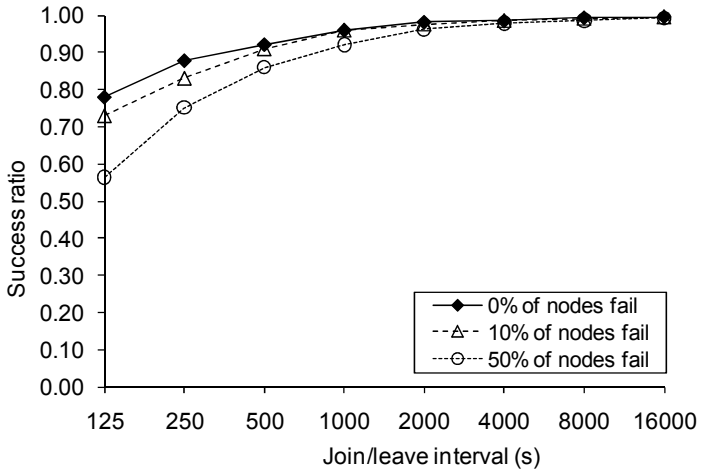


**Figure 3.28. Effect of churn on success ratio.**

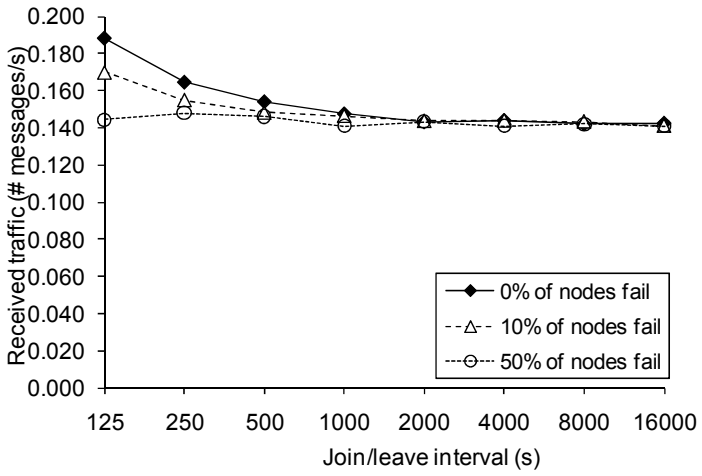


**Figure 3.29. Effect of churn on the frequency of received messages.**

Figure 3.29 shows the frequency of received messages under churn. The highest churn rate increases the message overhead up to 30% when graceful exits are used. When nodes leave ungracefully, the lower message overhead is partially related to the lower success ratio.

Testing with churn rates $1/f_{churn} < 250$ showed cases where the node joined itself as all other known nodes failed. Separately studying these cases, where a node wraps the ring around itself shows that this method succeeded to save the topology from partitioning in all tested cases. As any known node can be used to repair the ring, a moderate number of shortcuts and indexed nodes gives a good protection against partitioning.

### 3.5.16 Replicated resources

In current peer-to-peer networks, resources are typically well replicated. In zone indexing, the distribution costs and storage can be reduced when resources are replicated. It is often sufficient to find one of the replicated resources. In the case that a node $v$ receives an index update for a resource that it itself shares it can ignore the received index update and stop forwarding it. The succeeding nodes consequently only see a single copy of the resource. The number of copies can be limited, so that if the index already contains a given number of copies of a resource in a received update, the forwarding is suppressed. Another alternative is to combine the local resource descriptors with the received one and forward a single copy of the resource descriptor. This resource descriptor contains multiple addresses for the resource.

Similar methods can be used in searching. Copies of resources encountered by the search message can be ignored or combined with the existing result set. A resource requester that only needs a single resource can indicate that the search shall be interrupted when the first matching resource has been found. Alternatively, the search can be interrupted when a given number of copies have been found. If the resource requester later requires more matches, it can make a new search request that starts at the node which returned the previous search message.

The design goal of Zone Indexing is to provide deterministic searching. If it is necessary to trade determinism for better scalability, it is further possible to limit the maximum hop count for a search message. Such an approach corresponds to the TTL used in Gnutella and other common file sharing systems, and relies on well-replicated resources.

### 3.5.17 Comparison

Table 3.4 summarizes the properties of Zone Indexing with and without delay control using shortcuts. The zone size is assumed to be nearly optimal and the number of shortcuts fulfils Equation (3.21). The properties of Chord, based on [LCP+04], are included for comparison, although these algorithms have different goals.

**Table 3.4. Properties of the Zone Indexing architectures with and without shortcuts, and a comparison with Chord.**

| Architecture | Zone Indexing | Zone Indexing with Shortcuts | Chord |
|---|---|---|---|
| Index topology | Unidirectional ring | Unidirectional ring | Ring with fingers |
| Search topology | Unidirectional link from node to its border node | Unidirectional link from node to its border node, shortcuts | Ring with fingers |
| Index distribution method | Zone Indexing | Zone Indexing | Routed |
| Search distribution method | Forwarding to border nodes | Initial forwarding to shortcuts, forwarding to border nodes | Routed |
| Class | Hybrid proactive-reactive uniform | Hybrid proactive-reactive uniform | Structured single-key exact-match uniform |
| Links in network | $O(N)$ | $O(N + NN_s)$ | $O(N \log N)$ |
| Index entries per node | $O(\sqrt{N})$ | $O(\sqrt{N})$ | $O(1)$ |
| Index message scalability, $M_i$ | $O(\sqrt{N})$ | $O(\sqrt{N})$ | $O(k \log N)$ * |
| Search message scalability, $M_s$ | $O(\sqrt{N})$ | $O(\sqrt{N})$ | $O(\log N)$ |
| Search delay, $T_s$ | $O(\sqrt{N})$ | $O(\sqrt{N}/N_s)$ | $O(\log N)$ |

* Note that nodes in structured systems update each resource descriptor to a separate node, whereas nodes in the considered unstructured indexing systems update all resource descriptors to the same set of indexing node. This additional overhead of Chord is indicated with *k*.

## 3.6 Direct Index – a fully proactive architecture

Let us finally study fully proactive architectures. As stated by Equation (2.35), a fully proactive architecture is feasible in a marginal group of scenarios. One such scenario is studied in Section 4.6. Contrary to Zone Indexing, the studied architectures do not create and maintain a structured topology. Instead, they operate on arbitrary (random) topologies.

In a fully proactive architecture, the distribution of search requests is eliminated, but the efficiency of index distribution becomes important. A simple way to distribute index updates is to use flooding. For example, link-state routing protocols, including OSPF [Moy94], operate by flooding updates to all nodes proactively. Flooding is, however, inefficient with an overhead of $\Omega_i \approx D$. In this section we propose a more efficient method, called Direct Index, to distribute updates in an overlay

network. We utilize the fact that in a proactive architecture each indexed node is known. Our goal is to provide index distribution with an overhead approaching $\Omega_i \approx 1$.

Let us first review how a link-state routing protocol operates. Each node collects a map of the network in the form of a link table. For each link ($v$, $w$) starting from node $v$, node $v$ generates a link state advertisement that is distributed using flooding to all nodes in the network. All nodes store this information in the link table and consequently the link table contains an entry for both directions of every link. The link table, representing the network topology, is given as input to Dijkstra's shortest-path first algorithm [CLR+01], which generates the routing table. The routing table indicates for each destination $u$ the distance in hops to $u$ and the first link on the route to $u$.

### 3.6.1 Algorithm

In [Bei07b] we propose an algorithm called the *Direct Index* for index distribution. The algorithm is based on a link-state routing protocol. Each node builds a topological map of the network. While a link-state routing protocol operates at the network layer and uses flooding over physical links, the proposed Direct Index approach uses an overlay network. This allows us to reduce overhead by exchanging index updates directly between nodes over temporary links instead of forwarding the updates via intermediate nodes. Furthermore, duplicate messages are avoided as the distribution is controlled by a single node, the sender. For each known node, an exchange of index information is periodically invoked. Through the exchange, the node learns about the neighbors of the other node, and utilizes this information to construct a topological map of the group.

Each node $v$ maintains a table $P_v = \{p_1, p_2, \ldots,\}$ called the *peer table*. The entries $p_w = (N_w, E_w, T_w)$ of the peer tables represent each peer $w$ known by node $v$. The neighbor list $N_w$ indicates the neighbors of node $w$. The index entry $E_w$ includes a set of resource descriptors of the resources shared by node $w$ and $T_w$ is a timer used for triggering the following update exchange with node $w$.

As a node $v$ starts, it only knows its direct neighbors, which are preconfigured or obtained via some known node. Each of these neighboring nodes is initially added to the peer table with empty entries for $N_w$ and $E_w$. The timer $T_w$ is initialized to a random value between zero and *UpdateDelay*. The constant *UpdateDelay* is used to delay update exchanges in order to avoid a storm of messages on a topology change.

When the timer $T_w$ expires, an *update exchange* with node $w$ is triggered. An Update message with the fields ($N_v$, $E_v$, $F$) is sent over a temporary link (e.g. using UDP) directly to node $w$. The Update message contains a list $N_v$ of node $v$'s neighbors and the index entries $E_v$ describing resources that $v$ shares. The request flag $F$ is set to $F = 1$.

Upon receiving an Update message from node $v$, node $w$ updates the information for node $v$ in its peer table. The index entry $E_v$ and neighbor lists $N_v$ are copied from the received message and the timer $T_v$ is set to *UpdateInterval*. For each node $u \in N_v$ that has no corresponding entry in node $w$'s peer table, a new entry is created with empty $N_u$ and $E_u$, and

with the timer $T_u$ initialized to a random value between zero and *UpdateDelay*. If the request flag $F$ is set, node $w$ replies to node $v$ with a similar Update message containing the list of its neighbors and its index entries but with the request flag unset ($F = 0$). If the request flag is unset in a received Update message, an Acknowledgement message is sent to end the update exchange. When node $w$ receives an Acknowledgement message from node $v$, the timer $T_v$ in $P_w$ is reset to *UpdateInterval*.

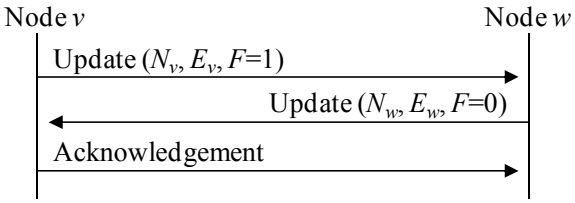


**Figure 3.30. Update exchange signaling in Direct Index.**

The update exchange is depicted in Figure 3.30. Through the update exchange node $v$ learns about the shared resources of $w$, and vice versa. More importantly, node $v$ learns about the neighbors $N_w$ of node $w$. On the $k$th round of update exchanges, node $v$ learns about nodes at a distance of $k$ hops. Consequently, it takes at most $d$ rounds of update exchanges to learn about all nodes in a network with diameter $d$.

When a node's local resources or local neighbors change, it may reschedule update exchanges with each known node $w$ by setting the timer $T_w$ to a random value between zero and *UpdateDelay*. Alternatively, it can wait for the following update round.

If an Update message sent to $w$ is not answered with an Update or Acknowledgement messages within a given timeout, it is considered lost. A lost message is resent using exponential backoff using the timer $T_w$. A node that has not been updated within several *UpdateInterval*s is declared unreachable and is removed from the peer table.

A node $v$ may optionally maintain a routing table $R_v$, in which the entry $R_v(w)$ indicates $dist(v, w)$. The routing table is recalculated with Dijkstra's algorithm after each change of $N_w$ in the peer table of $v$. The routing table is required in the access control extensions presented in Section 4.6.

### 3.6.2 Index compression

Maintaining full information about all resources in the network may require a substantial storage space. To reduce the storage size and bandwidth requirements, we propose compressing the index information using *Bloom filters* [Blo70].

A Bloom filter is a lossy but compact method to represent a set as a bit-string. The bit-string can be used to check whether or not an element belongs to the set. False positives are possible but not false negatives. The more elements that are added to the set, the larger the probability $p$ of false positives.

The Bloom filter is implemented as a bit-string of $m$ bits, initially zeroed. The filter uses $k$ hash functions. Each hash function maps an element to one of the $v$ bits. When an element is added to the filter, the $k$ bits indicated by the hash functions are set to one. To test whether an element is in the set, the element is hashed and the $k$ bits indicated by the hash functions are tested. If all are ones, a positive is obtained. If not all of the indicated bits are ones, a negative is obtained, and the element is definitely not in the set.

According to [FCA+02] the probability of false positives in a filter containing $n$ elements is minimized for

$$k = \ln 2 \cdot m/n\,, \tag{3.23}$$

in which case the probability is

$$p = \left(1/2\right)^k = 0.6185^{m/n}\,. \tag{3.24}$$

This allows us to determine the optimal number of hash functions as

$$k = \frac{\ln p}{\ln 0.5} \tag{3.25}$$

and the optimal number of bits as

$$m = \frac{nk \ln 0.5}{\ln 0.6185} = 1.4427\, nk\ . \tag{3.26}$$

Bloom filters can be used in a wide range of applications. A good overview of different networking applications is given in [BM02]. In our resource sharing application, the elements are the attributes of the resource descriptors. The hash functions then calculate the bit positions based on the attribute string. A single Bloom filter can describe the attributes of several resource descriptors. In the extreme case, a Bloom filter can describe all resources available at a node. In order to reduce the probability of false positives, the attributes of a resource provider may be divided between several Bloom filters.

Although providing significant reduction in index size, Bloom filters come with two disadvantages. Firstly, because of the false positives, the actual existence of a resource matching with the filter must be checked separately. Therefore searching involves an additional roundtrip to a set of nodes to check whether the resource actually is available. Secondly, the possibility to make complex searches is lost. The use of hash functions reduces the possible queries to exact-match single-key queries. However, because the whole filter is available at a single node, some types of multi-attribute queries can be performed efficiently, as presented in Section 4.6.6.

### 3.6.3 Performance

Our simulations reported in [Bei07b] compare three architectures:

1. a reactive system with flooding-based search distribution,
2. a proactive system with flooding-based index distribution, and
3. a proactive system with Direct Index distribution.

In all compared cases, a similar topology is used. The network has $N$ nodes connected with a random power-law topology. The default network size is $N$ = 100 nodes and the default average degree is $D$ = 10. Index updates are according to an exponential distribution with an average of one update per day, $f_i$ = 1/86400 $s^{-1}$. Extra index updates are triggered if no index updates is generated in two days, i.e. *UpdateInterval* = 172800 s, which practically increases $f_i$ further. For avoiding bursts of updates, *UpdateDelay* = 60 s. These parameters are chosen to reflect the intended use scenario, described in Section 4.6. Search requests are generated according to an exponential distribution with frequency $f_s$. The search frequency $f_s$ is adjusted to obtain a desired search/index ratio $r = f_s / f_i$. By default, $r$ = 1:1. In the simulations, one of the parameters $N$, $D$, and $r$ are varied while the others take their default value. Index flooding and Direct Index use index compression. All local resources are described using a Bloom filter, simulated so that it generates false positives with a probability of $p$ = 0.02.

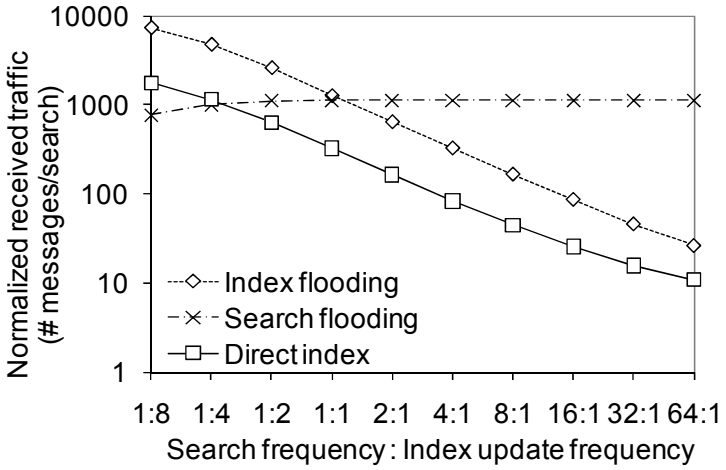


**Figure 3.31. Message receptions depending on the search/index ratio.**

Figure 3.31 shows the message receptions per search for the examined architectures. The traffic is normalized so that the total number of receptions is divided by the number of searches during the simulated time. The simulations confirm our assumption that the search/index ratio is the most important parameter in determining whether a proactive peer-to-peer system is feasible. When flooding is used for both search distribution and index distribution ($\Omega_s = \Omega_i$), a proactive system generates less traffic than a reactive system if $f_s > f_i$, as given by Equation (2.40). In the simulations $\Omega_s = \Omega_i \approx D\text{-}1$, and it is expected that a flooded message generates about $ND \approx 900$ receptions.

Direct Index shows a lower traffic than index flooding with $\Omega_i < D$. In the simulation $\Omega_i \approx 3$ because each update exchange requires three messages (Update, Update, and Acknowledgement). Using unacknowledged updates (as in flooded index updates), would yield a lower overhead – even as low as $\Omega_i \approx 1$.

Compressing the index with Bloom filters causes some additional roundtrips due to false positives. The proactive solutions therefore generate a small additional overhead because of the roundtrip (2 messages per query) for confirming the availability of the resource once a match has been found. False positives are generated at a probability $p$ per node, resulting in $pN$ extra roundtrips per query in addition to the roundtrip used for the actual resource. The total search message count $M_s = 2 + pN_s$ has a marginal effect on the total overhead unless $p$ is high. In searching the simulated 100 node network, on average three nodes are queried of which two are due to false positives.

For the Direct Index algorithm, the independence of node degree is beneficial in scenarios with a high average node degree, as shown in Figure 3.32. The message overhead of flooding increases linearly with increasing average degree as $\Omega_i \approx D\text{-}1$. The overhead of the Direct Index algorithm is independent of the degree.

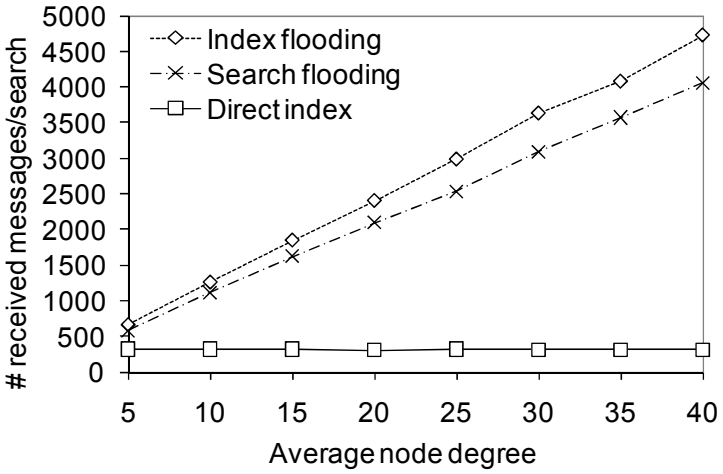


**Figure 3.32. Message receptions under increasing node degree.**

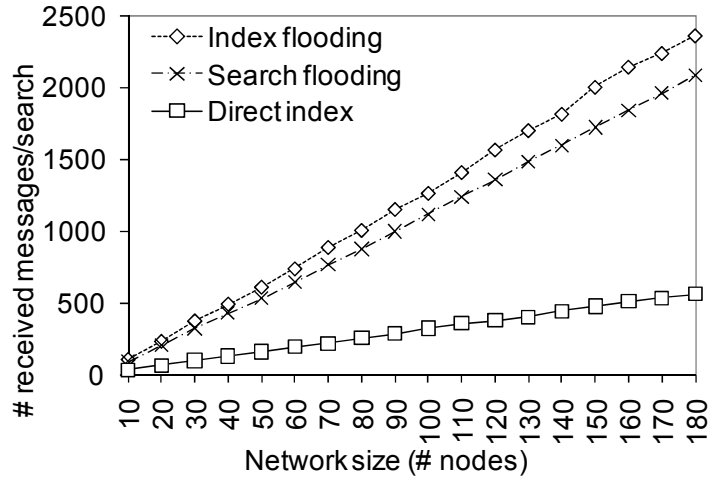


**Figure 3.33. Message receptions under increasing network size.**

As visible in Figure 3.33, the overhead of both flooding and Direct Index grows linearly with the network size. The simulations show that the Direct Index architecture always performs better than a flooding-based index distribution in terms of message overhead.

Figure 3.34 shows the search delay for the first matching resource, assuming a delay of 20 ms per overlay link. In the proactive architectures the index information is immediately available. Because Bloom filters are used, a single roundtrip is necessary to verify that the indicated resource actually exists. Reactive searching based on flooding has a delay that is proportional to the network diameter, which is shown to increase in the order of O(log *N*) for given power-law networks [CL02].

All architectures are deterministic, and the success ratio was measured to be $R_{success} > 99.75\%$ in all simulations. The failed searches are due to the dynamically added and deleted resources.

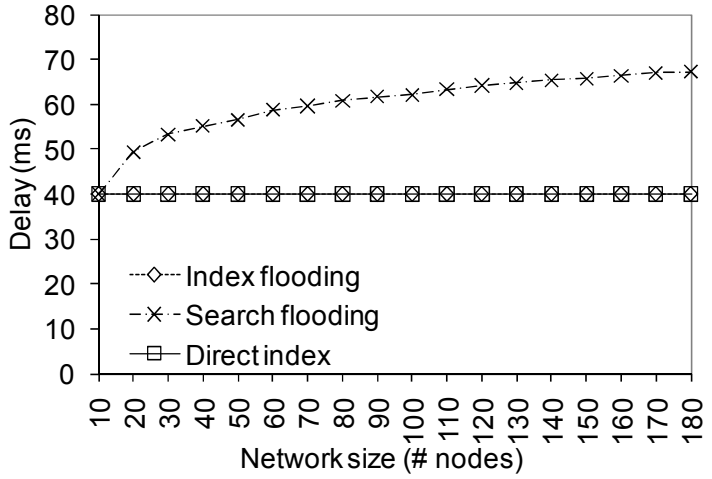


**Figure 3.34. Delay under increasing network size.**

### 3.6.4 Related architectures

Direct Index replaces flooding with direct updates over temporary links to reduce the cost of index distribution. Could the same approach be utilized in a reactive system to make search flooding more efficient? Unfortunately this is not equally feasible. Direct Index is inherently based on proactive information about known nodes in the network. Even if direct links could be used for distributing search requests, the maintenance of the topology information still needs O(*N*) messages on top of the O(*N*) messages used for searching. Therefore the usefulness of such a "Direct Search" approach is marginal.

### 3.6.5 Comparison

Table 3.5 compares the properties of Direct Index with index flooding and search flooding.

**Table 3.5. Properties of flooded search, flooded index and Direct Index.**

| Architecture | Search flooding | Index flooding | Direct Index |
|---|---|---|---|
| Index topology | - | Random power-law | Random power-law |
| Search topology | Random power-law | - | - |
| Index distribution method | - | Flooding | Link state with direct links |
| Search distribution method | Flooding | - | - |
| Class | Reactive uniform | Proactive uniform | Proactive uniform |
| Links in network | $O(DN)$ | $O(DN)$ | $O(DN)$ +temporary links |
| Index entries per node | $O(1)$ | $O(N)$ | $O(N)$ |
| Index message scalability, $M_i$ | - | $O(DN)$ | $O(N)$ |
| Search message scalability, $M_s$ | $O(DN)$ | $O(pN)$ $0 < p < 1$ | $O(pN)$ $0 < p < 1$ |
| Search delay, $T_s$ | $O(\log N)$ | $O(1)$ | $O(1)$ |

The largest limitation of proactive systems is the need to store index and state information about all other nodes. As all $N$ nodes store state of all $N$ nodes, the total state information in the network is in the order of $O(N^2)$. This restricts the scalability and is one of the reasons why the proactive peer-to-peer approach is uncommon in practical systems. Because of this, we see proactive architectures as more applicable to group-based resource sharing, as discussed in Section 4.6, than to global scenarios.

Based on our performance study, we provide the flowchart in Figure 3.35 for comparing the fully proactive solutions to an alternative solution with $\Omega_s$ in a network with parameters $r$, $D$ and $N$. The overhead of flooding is assumed to be $\Omega_i \approx D$ and of Direct Index $\Omega_i \approx 3$. As hybrid proactive-reactive solutions generally require a managed topology, these solutions may not always be available, as assumed in our intended scenario. When only fully proactive and fully reactive solutions are available, the choice of solution is determined by Equation (2.40). Furthermore, the Direct Index architecture requires temporary links, which may not always be available.

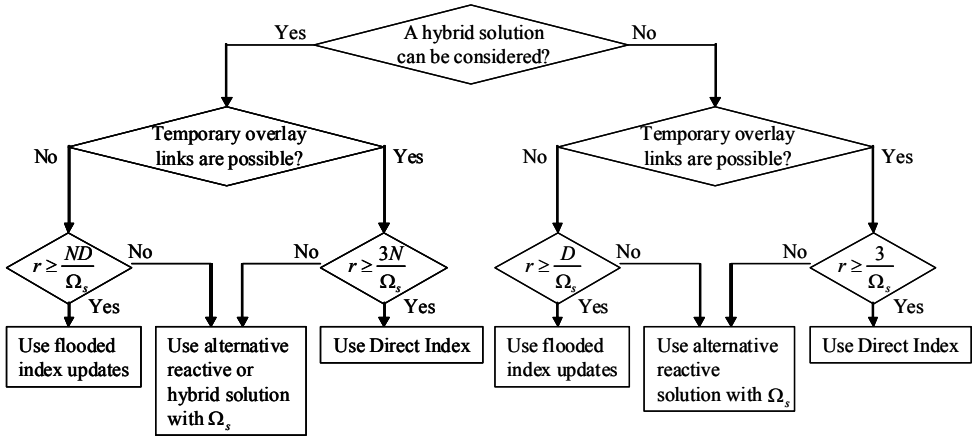


**Figure 3.35. Flowchart for considering proactive solutions.**

## 3.7 Summary

We presented three new types of architectures involving index distribution. For all architectures we required that it must be possible to locate all existing resources, and that the load is evenly distributed between the nodes. We also required a possibility to implement complex queries.

The IPIC architecture extends the PIC architecture by allowing the clusters to be arbitrarily interconnected. This architecture requires a new search algorithm and we presented the Stack Based Random Walk (SBRW), which visits each cluster exactly once. To reduce the delay of SBRW, we also proposed the Replicating Stack Based Random Walk (RSBRW) algorithm which uses multiple walkers in parallel at a slightly higher overhead. The IPIC architecture can replace PIC when full connectivity between clusters and exponential growth of state information are undesired.

We proposed the Zone Indexing architecture, which minimizes the traffic by making the balance between proactive and reactive operations optimal. While this is possible with existing architectures, such as PIC and PSC, our architecture is more applicable to practical scenarios by allowing nodes to join and leave without restructuring the overlay. It remains optimal under changing conditions, such as when the network size grows or the ratio between searches and updates changes. Optimality is maintained by continuously adjusting the zone size with the provided algorithm. We also presented an algorithm that reduces the search delay using shortcuts. Zone Indexing is a general purpose overlay useful when complex queries are required.

Finally, we proposed the Direct Index architecture, which is fully proactive with a low traffic overhead. It is particularly useful in networks with a high node degree and a topology that is determined by external influences (i.e. cannot be modified by the overlay). Social networks are examples of such networks.

The architectures are designed to have properties that are advantageous in mobile networks. The application of the presented architectures in mobile networks is described in the following chapter.

# Chapter 4

# Resource discovery in cellular networks

This chapter studies the use of resource discovery in cellular networks. We evaluate the feasibility of peer-to-peer systems in cellular networks both from a user perspective and from a technical standpoint. The chapter begins with an introduction to the use of resource discovery and peer-to-peer services in cellular networks and a presentation of related research. We present the results of four user surveys, which provide input for the technical design. We propose architectures for two different scenarios. In the first scenario, the resource discovery service is provided by the operator or a third-party service provider. In the second scenario, the system is completely maintained by the participating users. We propose signaling schemes based on the Session Initiation Protocol (SIP) [RSC+02]. Finally, we evaluate the technical feasibility of the proposed systems using prototypes.

## 4.1 Introduction

Today's mobile phones are in an ever increasing degree becoming more similar to personal computers. The memory and the processing power are increasing rapidly. Together with increasingly faster network connections, this allows complex communications software to be developed. Integrated peripherals like cameras, media players and GPS receivers create possibilities for new types of applications, which specifically can benefit from the mobile nature of the phone. At the same time, applications based on social networks, communities, sharing and user-generated media are becoming popular. Well-known examples include Wikipedia, MySpace, Flickr, YouTube, and Facebook. Still, most of these so called Web 2.0 applications are centralized and based on fixed servers. However, distributed operation is gradually becoming part of many traditionally centralized applications, which can be exemplified by Skype and Spotify. We expect these applications to be popular in mobile phones as well.

In this chapter, we examine architectures that enable resource sharing in a peer-to-peer fashion between users in a cellular network. The architectures form the basis of collaborative applications that support sharing of files, media, services and other types of resources between users. In particular we study resource discovery, i.e. searching, in mobile peer-to-peer systems.

The following are hypothetical examples of applications that utilize resource discovery in mobile networks:

1. A user takes photos with a camera phone while traveling. The newest pictures are immediately available for the user's friends and family without being uploaded to a centralized server first.
2. During an event (e.g. a sports event), several users are broadcasting real-time video transmission using their camera phones. Other users can locate these video streams using suitable keywords or by clicking on the location on a map.
3. While visiting a city, a tourist searches for nearby restaurants. As search results, he obtains the latest menus, current offers, pictures and user comments. The application presents the restaurants ordered according to distance from the current location. Users in the area can be located, for example, to form a discussion group.
4. Users in a tennis club share their pictures, video clips, score tables, and equipment reviews. Only invited users may access the resources.
5. While shopping, a user finds an interesting product for which he wants to obtain additional information and reviews from other users. The product is identified by extracting visual properties of the photo or by decoding the bar code. In addition to the specifications and description from the manufactures, the user finds comments, reviews, pictures and application suggestions from other users.
6. A map displays notes and photos related to the shown locations. The information is created and maintained by other users. Also parts of the map can be downloaded from other users to reduce the load on centralized servers.
7. A working group shares documents between group members. Instead of centrally collecting the documents on a file server, the files reside on the user's devices.

### 4.1.1 Considered networks

While more types of wireless networks, such as Wi-Fi and Bluetooth, are becoming supported by phones, cellular networks, including GSM and UMTS networks, still have a central role in providing data access for mobile users. Cellular networks are infrastructure-based networks maintained and operated by a commercial operator. The operator has a strong role in providing services and controlling the network use. The operator commonly aims to provide services either itself or by a contracted third-party service provider. Traditionally operators have preferred charging based on the service use, avoiding being used as a "bit-pipe" that only transports traffic between the phone and some external (e.g. in the public Internet) service provider. However, recently flat-rate schemes have become more common. In our survey, we found

that 22% of users have a flat-rate scheme. A later survey by Heikkinen & Nurminen [HN09] reports 33% of users having flat-rate, 32% usage-based pricing and 24% limit-based pricing.

Joint efforts by the Third Generation Partnership Project (3GPP) and IETF have been spent on providing the 3rd generation (3G) networks with an IP based core. The IP Multimedia Subsystem (IMS) is a platform offering common functionality including session control, charging, accounting and authorization for IP based services, such as VoIP (Voice over IP). The signaling in IMS is based on the Session Initiation Protocol (SIP), which has been extended for the needs of IMS. Basing a mobile peer-to-peer service on IMS allows reuse of the services provided by the infrastructure avoiding the development of new proprietary protocols. It is clear that for a mobile peer-to-peer service to succeed technically in an operator controlled environment, it must integrate well with the rest of the network.

In addition to cellular network access, modern phones may be connected to several other networks. Bluetooth connectivity is mainly used to connect the phone to nearby peripherals and users or to a computer. Wireless LANs (WLANs) allow the phone to be part of a corporate or private LAN, and thereby part of the public Internet without using the operator's services. These networks are generally privately owned and available for free to a limited number of users. Commercial and community-driven WLAN networks exist as well but these are still uncommon. Furthermore, WLAN can form the foundation of an ad hoc network, connecting several devices together without any fixed infrastructure. Because of the multitude of networks, the possibility to switch between networks or utilize several networks simultaneously is becoming interesting.

### 4.1.2 Technical constraints

Compared to the fixed Internet, the cellular environment adds technical constraints arising both from the terminal and the network. Compared to a fixed PC, the mobile terminal has significant limitations in its

- battery power,
- memory,
- processing power,
- programming environment and operating system support,
- persistent storage, and
- user interface, with small screen size and rudimentary input functionality.

The limitations of the network include

- low bandwidth, especially in the uplink direction,
- high delay,
- connectivity limited by NATs and firewalls,
- possible monetary costs, unless flat rate charging is used, and
- frequent disconnections due to mobility, i.e. a high churn rate.

These limitations must be observed in the design of resource discovery solutions. The software and the underlying algorithms must be

simple to reduce the required processing power and memory. The state information and number of active connections must be reduced. More importantly, the traffic over the wireless link must be reduced. According to [BFN06], a mobile Gnutella client constantly generates 3-4 kbps traffic with a minor dependence on the user's own activity. There are several reasons for reducing the wireless traffic:

- The available bandwidth is limited and shared between multiple users.
- Transmissions create interference.
- Transmitting and receiving consume power, which shortens the time between recharging.
- Power consumption also needs to be reduced for environmental reasons.

The battery capacity of mobile devices has not been able to keep up with the development of computational resources, falling far behind Moore's law [Rie95]. In terms of energy consumption, the transmission cost is high. According to [PC04], each bit sent over a WLAN link consumes 700 nJ, whereas a CPU cycle consumes 0.07 nJ. Transmission of a single bit therefore corresponds to 10000 CPU cycles.

Because of monetary costs, especially the traffic that gives the user no direct and visible benefit such as signaling and network maintenance traffic should be minimal. Power consumption must be reduced by lowering the computational requirements and the network activity. Altogether these factors motivate the use of algorithms that are more efficient than flooding, and to avoid algorithms with high structure maintenance costs, such as structured overlays. They also motivate performing some functions in elements in the fixed network.

### 4.1.3 Mobile peer-to-peer scenarios

Peer-to-peer services can be deployed in cellular networks in several scenarios. The choice of architecture largely depends on whether the operator or some other commercial provider is providing the service, or whether the users run the service among themselves. In the existing networks, the architectural choice often, but not always, follows the type of control. We separate three cases depending on the controlling body:

1. *Single controlling body*: In this case there is a central body, e.g. the operator or a service provider, which controls the service. This model often leads to a centralized architecture, such as in Google [Google] and early Napster [Napster]. However, even though there is a single controlling body, the architecture can be partly distributed, often with a few centralized elements, such as in Skype [Skype].
2. *Multiple controlling bodies*: This case involves several bodies, e.g. operators, which are equal peers. Each body controls a part of the service and a part of the network. Each user subscribes to one operator. This model arises when several systems with a single controlling body are interconnected. Although semi-centralized architectures are natural for this model, the technical solution can still be distributed.

3. *Decentralized control*: In this model each user is independent and there is no controlling body. Most of the current file sharing systems follow this model. The operator has no, or a minimal, role in controlling the network. The technical architecture is usually a search flooding architecture (e.g. Gnutella and Kademlia) or, for scalability, a semi-centralized architecture (e.g. Kazaa).

In this work, we study peer-to-peer services in cellular networks in two scenarios: commercial peer-to-peer services with multiple controlling bodies (operators) and a system with decentralized control involving only users. The single controlling body can be considered as a special instance of the case of multiple controlling bodies. We only consider data transfer over networks available at the time of the writing, such as GSM and UMTS, and networks currently being introduced, such as IMS. We mainly consider cellular phones, although similar systems can be used in laptops and other devices connected to the networks.

## 4.2 Related research

A large number of different peer-to-peer clients have been developed for the fixed network, including well-known clients like Gnutella, Kazaa, e-Donkey, Bit-Torrent, etc. Research on mobile peer-to-peer systems has mainly concentrated on using peer-to-peer technology in mobile ad hoc networks. Research on service discovery in ad hoc networks is presented in Chapter 5. Significantly less research has been spent on developing peer-to-peer solutions specifically intended for cellular networks. Mobile peer-to-peer is an upcoming research area, as the need for mobile collaborative applications increases.

The concept of mobile peer-to-peer has been explored using popular fixed peer-to-peer systems ported to the mobile network. Symella [KFM07] is an implementation of the popular Gnutella protocol on the Symbian platform. The implementation supports the downloading of files shared by fixed clients, but lacks the possibility to upload content, and is therefore not a complete implementation. The behavior of only downloading without uploading is known as free riding or leeching, and it is not appreciated by the file sharing community. SymTorrent [KEP07] is a BitTorrent [BitTorrent] client implemented on the Symbian platform. BitTorrent is currently one of the most popular peer-to-peer file sharing protocols and this popularity makes it attractive to implement a version for mobile devices. SymTorrent is a complete implementation, including both uploading and downloading. MobTorrent [ENK08] is a BitTorrent client implemented on the J2ME platform in order to measure and evaluate the capabilities of low-end devices and examine if they can join a large already existing peer-to-peer network. Currently MobTorrent supports downloading only. MobileMule [MMule] is a mobile control application for the eMule [eMule] client but it is not an actual peer-to-peer application. Fring [Fring] and iSkoot [iSkoot] bring the peer-to-peer based Skype application to the mobile phone. In [BH09] a mobile device can access the eDonkey network via a fixed peer that has been modified to support mobile devices and relay traffic.

JXTA for J2ME [AHP02] is a version of JXTA adapted for mobile terminals. JXTA [MM02] is a general purpose overlay platform based on XML messaging. Peers are divided into edge-peers and super-peers, and super-peers are further divided into rendezvous peers and relay peers. Edge peers have low capacity and are possibly behind firewalls, rendezvous peers coordinate the peers in the network, and relay peers provide firewall traversal. Peers can be divided into peer groups to provide logical clustering and message scoping.

Kato et al. [KIS+03] presents an XML-based mobile peer-to-peer protocol, where the control is located in central nodes. The architecture is semi-centralized. New types of architectures have mainly been examined theoretically and through simulation. Bakos et al. [BCF+03] compare topologies resembling wireless networks using simulation, and found that semi-random mesh and connected stars are suitable for homogeneous and heterogeneous networks, respectively. Marossy et al. [MCB+04] examine the use of the PIC architecture in GPRS networks and compare different cluster topologies. Bakos et al. [BFN06] examine peer-to-peer applications on mobile phones using four experimental systems: distributed computing, adaptation of fixed software to the mobile phone, using a PIC architecture on mobile phones, and searching in a social network. In addition to [BFN06], searching a social network has been proposed by the same authors in [BFN05a] and [BFN05b], but with an architecture differing from ours. Recently, Tiago et al. [TKK+08] published a solution with similar ideas as the ones in [Bei07b], but with a reactive approach based on flooding with user-controlled iterative deepening. The solution utilizes the social network formed by the address books of cellular phones but does not use explicitly defined groups.

Few user studies on peer-to-peer systems in cellular networks had been performed when we started our work. Kostamo et al. [KKK+07] study incentives for content distribution among mobile users using a questionnaire. An interesting conclusion is that mobile users are willing to send their self-generated content to a greater number of users than commercial content. Heikkinen et al. [HKV09] study the traffic and use of peer-to-peer applications using traffic traces and user logs. Heikkinen and Nurminen [HN09] present a questionnaire study of user attitudes towards peer-to-peer services.

## 4.3 Our contribution

This chapter proposes various resource discovery solutions for cellular networks. The solutions are primarily aimed at providing a platform for a peer-to-peer type of resource sharing between mobile users. Some of the publications that form the basis of this chapter are shared with Chapter 3, as we apply the presented architectures to cellular networks.

In order to make a user-centric approach, we first perform initial user studies based on questionnaires and interviews. We present and analyze the results in Section 4.4. The studies elucidate the thoughts, opinions, behavior and expectations concerning resource sharing applications in a cellular environment. This work directly uses the results of four separate surveys. The results of the survey made in 2005 were partly published in

[MBL+06b] and [MBL+07]. This survey was designed and performed by the present author and Matuszewski as joint work. Here, we also include some unpublished results from this survey. The other three surveys have been organized by the present author and carried out by students under the present author's supervision. The results are reported as student work in [Lev09], [Pal09] and [Lag09]. Our work analyzes the raw data independently from these. The central results and constraints from the user studies, such as the need for access control and groups, are utilized in the rest of the chapter.

We present two different approaches to providing peer-to-peer resource discovery in cellular networks. The first approach, the operator-controlled service, is presented in Section 4.5. We motivate the need for a two-layer hierarchical architecture. We propose a commercial resource sharing service involving multiple operators and we describe how the service is implemented in the IMS. This architecture was originally published in [BML+05] and refined in [MBL+06b] and [MGB+07]. The architecture and its application in IMS were developed jointly by the present author and Marcin Matuszewski. This work further extends the published scheme by specifying the functions of the fixed nodes in detail. We present unpublished work on the implementation of groups and access control, including definitions and implementation of access control classes. Although we take a different approach, we use as a background a limited set of ideas in the Master's thesis of Tuomo Soinio [Soi09] instructed by the present author. As unpublished work, we evaluate how different resource discovery architectures can be applied to the fixed network of the system. In particular, we propose utilizing our IPIC architecture, which allows operators to be connected according to business relations while still maintaining control over their network and customers.

In the second approach, presented in Section 4.6, there is no support from an operator or an external provider. The users run the system in a fully distributed fashion. We motivate the need for uniform architectures. We propose a new scheme for distributed group management. We propose extensions to flooding and the Direct Index algorithm in order to support groups and policies. The fundamental ideas have been published in [Bei07b]. We present an unpublished analysis of the feasibility of proactive solutions. We analyze the properties when index compression is used and suggest how queries are specified in this case. The fully distributed approach represents independent work of the present author.

In Section 4.7 we present signaling schemes based on extensions to SIP for resource discovery, as originally suggested in [BML+05]. We establish a set of requirements for the scheme. The first signaling scheme has been published in [MBL+06b]. The second scheme is published in [MGB+07] and given as input for standardization in [GMB+06]. We further present how SIP is used for establishing the access connection.

At the time of our research, there were no existing implementations of peer-to-peer software for mobile clients. Therefore, it was also unclear if mobile phones are able to run peer-to-peer software in a satisfactory way. In Section 4.8 we examine whether mobile peer-to-peer is a technically feasible concept with the current technology and in the

current networks. We formulate a set of postulates that we test using a prototype implementation involving multiple components. A mobile client has been developed by Juuso Lehtinen and Tuomo Hyyryläinen under instruction from the present author and Marcin Matuszewski. The client implementation has been presented in [Leh06]. A relay and a centralized index node are developed by the present author. A distributed index node is implemented by Victor Morales under the supervision of the present author. This implementation is presented in [Rey07]. We configure the components into three testbeds, with which we prove the validity of the concept both from a device and a network perspective. Testbed O1 has been demonstrated in [BML+05] and [MBL+06a]. We measure the performance and compare it to the user's expectations. Furthermore, we evaluate whether a SIP-based signaling scheme works in practice. Some of the results of Testbed O1 are reported in [MBL+06b]. Testbed O2 and O3 represent unpublished research, but with some results presented in [Rey07]. We further present a new evaluation of the influence of different topologies on the search delay.

In Section 4.9 we evaluate the fully distributed peer-to-peer service using a prototype implementation. We verify the feasibility of the concept and examine whether the restrictions in today's network prevent the practical applicability. The implementation is based on the concept defined in [Bei07b] with a protocol defined in [Bei07c]. The prototype is implemented by Veikko Pankakoski under the instruction of the present author and described in a report [Pan09].

## 4.4 Mobile peer-to-peer services from the user's perspective

The cellular environment has certain properties that make the use cases different from the fixed Internet. An evident but remarkably important one is the fact that the device is mobile. The user is able to request and consume information and services at any time and in any location. The use of a mobile peer-to-peer system is therefore often expected to be related to a given location or time, whereas quick and interactive access to information is needed. The phone is less suited for long downloads. In contrast to server-based solutions, pieces of content can be made available immediately after they are created without uploading to a server.

Today's mobile phones are powerful multimedia devices, allowing not only displaying different types of multimedia content, but also creating content by taking photos, recording video clips, recording voice and taking notes. The popularity of blogs and reality television also shows an increasing trend in the interest in documenting, disseminating and disclosing personal experiences as well as observing, viewing and becoming entertained by the experiences of others. Thus, we can assume that there are both providers and consumers of legal peer-to-peer content in the cellular network. The willingness to distribute illegal content, especially to masses of unknown recipients, is reduced due to operator control, charging and easy identification of users.

The mobility, the increasing bandwidth, the capability for creating media, the popularity of user-generated media and communities is a

powerful combination. Peer-to-peer technology suits well as a distribution method in this scenario. If only a part of the created content actually is accessed, it may not make sense to upload everything to a server. The peer-to-peer approach transfers only requested content, saving both bandwidth in the access network and storage space on the servers. Caching allows the peer-to-peer and server mode to be combined so that the content being downloaded through a peer-to-peer system is stored on a cache in the network, allowing popular content to be accessed efficiently. The best parts of both worlds can thus be combined.

Technologically the concept is promising, but are users interested in this type of application? In order to better understand the potential users, we performed a set of surveys, both based on questionnaires and interviews. The people answering the surveys were mostly technical students, representing young people with a technical understanding, that are assumed to be the first people to adopt such a service. However, the relatively small sample size and the homogenous background limit the applicability of the results to other than an initial background study. The following surveys are used in this work:

- *Survey 1*. This questionnaire performed in 2005 was answered by 98 respondents. Of the respondents 86% are Finnish and 80% are male. All respondents are students of Helsinki University of Technology. 64% are employed part- or full-time. 66% are between 21 and 25 years old and 24% between 26 and 30 years old. Part of the results have been published in [MBL+06b] and [MBL+07].
- *Survey 2*. This questionnaire performed in 2008 was answered by 125 respondents. Of the respondents 75% are Finnish and 78% are male. 78% of the respondents are studying, of which 59% study technology. 72% are employed part- or full-time. The majority (85%) are between 22 and 28 years old. A thorough report is given in [Lev09].
- *Survey 3*. This questionnaire concentrating on microblogging performed in 2009 was answered by 16 respondents. Of the respondents all are Finnish and 50% are male. All respondents are students of Helsinki University of Technology. The respondents are between 19 and 30 years old. A presentation of the survey can be found in [Pal09].
- *Survey 4*. This interview concentrating on social networks performed in 2009 involved 8 respondents. All respondents are Finnish and students of Helsinki University of Technology. Half of the respondents are male. The respondents are between 20 and 25 years old. From this interview we use only the suggestions for applications. The other results have been reported in [Lag09].

### 4.4.1 Interest in obtaining content

In both Survey 1 and 2, we first examine the interest in obtaining content on the mobile phone without considering the distribution technology. The interest is surveyed based on the type of content and the creator of the content. Most types of content were divided into two groups: the content created by people known to the user, and content created professionally.

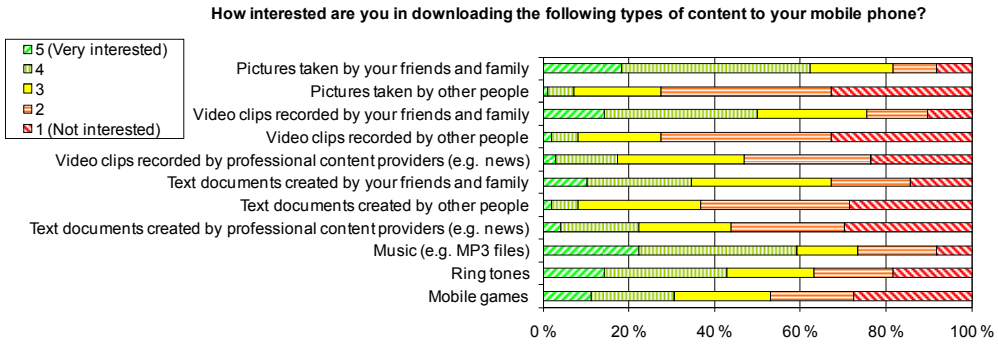


**Figure 4.1. Interest in downloading content to a mobile phone in Survey 1.**

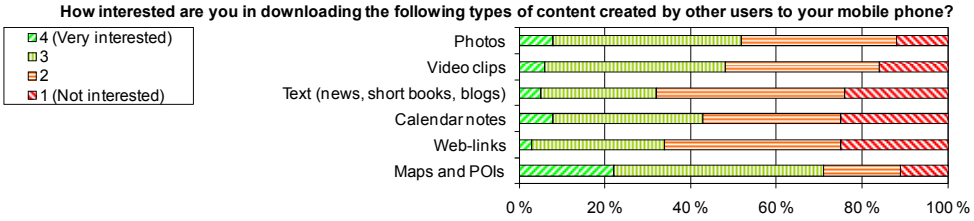


**Figure 4.2. Interest in downloading content to a mobile phone in Survey 2.**

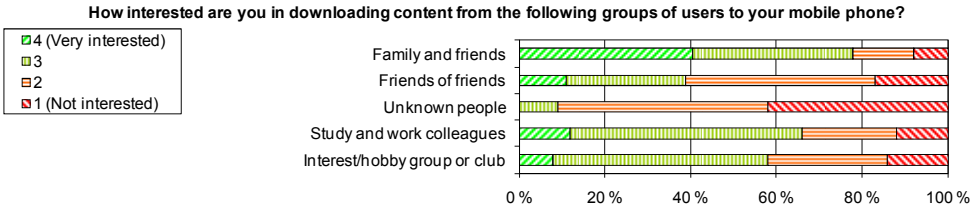


**Figure 4.3. Interest in downloading content from different groups of users to a mobile phone in Survey 2.**

According to Survey 1, shown in Figure 4.1, the most interesting type of content is pictures taken by friends. Video clips made by friends are almost equally interesting. These types of content show a high difference in interest depending on whether the creator is known: pictures and video clips made by other people are considered uninteresting. Text documents are generally less interesting than pictures and video clips, but also here the interest was higher for text documents made by friends. Commercially created content (e.g. news reports) is valued more than content generated by unknown users, but less than content generated by known users. The differences between the popularity of these sources are large: most users have a high interest in the content generated by their friends, a moderate interest in professionally generated content and rather low interest in content made by other users. Music is considered almost as interesting as pictures by known people. Ring tones and mobile games were interesting only to a smaller part of the respondents. For music, ring tones and mobile games the creator was not defined.

In Survey 2, the content and creators are separately surveyed. As shown in Figure 4.2, the interest in downloading different types of

content corresponds to the results of Survey 1, with photos and video clips considered as the most interesting types and text considered less interesting. Calendar notes and web-links, included as new content types, are also considered relatively interesting. Interestingly, maps and points of interest (POI) are considered as the most promising types of content. Figure 4.3 shows the importance of knowing the source of the content. We can see a much higher interest in downloading content from family and friends than from the friends of friends, which in turn is higher than the interest in downloading content from unknown people. The special groups consisting of study and work colleagues and interest/hobby related groups is relatively high, although not as high as for family and friends. These groups contain people of varying closeness to the user, although most can be considered as known.

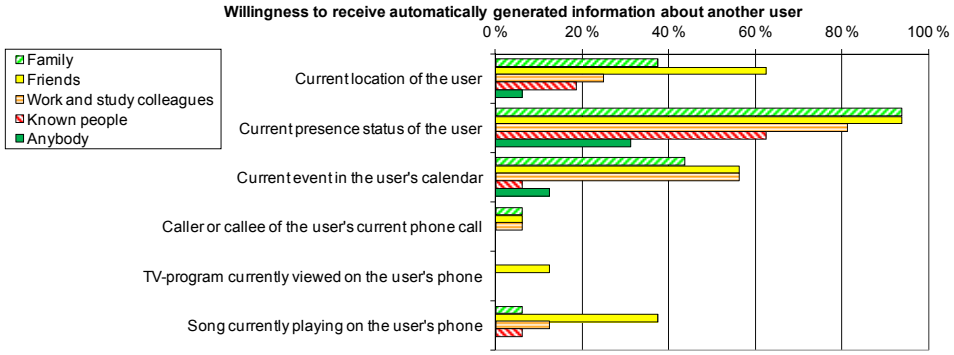


**Figure 4.4. Interest in downloading content to a mobile phone in Survey 3.**

Status information, such as the current availability of a user, is becoming part of the communication in several applications, including Skype and Live Messenger. In many systems, such as Facebook, the user can comment on what he/she currently is doing. Microblogging, represented by Twitter [Twitter], among others, is a service fundamentally aimed at distributing short status updates about thoughts and doings. The mobile phone provides interesting possibilities for such status information as it is always following the user. The mobile phone further makes it possible to automate the generation of certain types of status information. Automatically obtained status information can be presented in the phone book or in a separate screen showing status information about a few selected contacts.

In Survey 3, shown in Figure 4.4, we asked the respondents about the interest in obtaining status information about other users. The most requested information is the presence status, to which many users have been used to through many popular applications. The current location of another user is often mentioned in phone calls. Automatic presentation of the current location is desired by several respondents, especially regarding their friends. If the calendar contains information about a current event of the user, around half of the respondents would like to obtain this information. Other types of information is less popular. However, the currently playing song in a friend's phone is considered interesting. Interestingly, users are less willing to obtain automatically

generated status information from their family members than from their friends.

### 4.4.2 Willingness to share content

The willingness to share different types of content with different types of users according to Survey 1 is shown in Figure 4.5. As expected, a user's willingness to share a resource with another user depends on how closely related the users are. The willingness to share resources with family members and close friends is much higher than the willingness to share resources with other friends and work and study colleagues. Willingness to share resources with other known people (people in the address book of the phone) and to completely unknown people was very low. About 10% of the users did not want to share given resources with anybody.

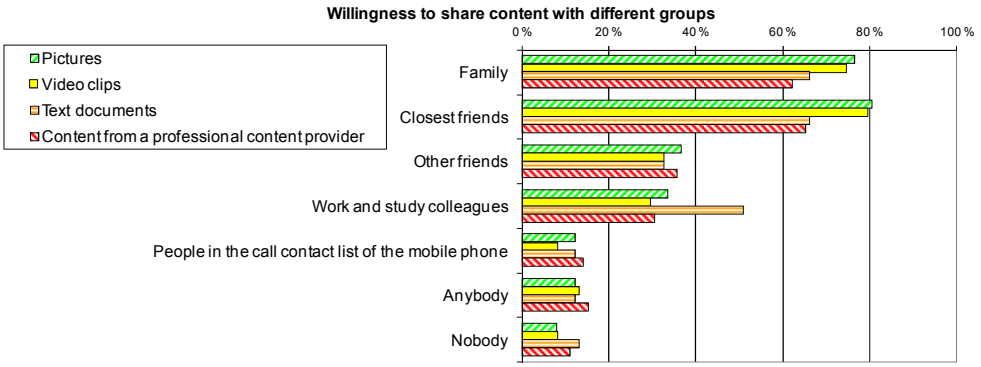


**Figure 4.5. Willingness to share content with different groups of users in Survey 1.**

The difference between different content types was low. Generally, users were more interested to share pictures than video clips. The interest to share text documents was lower than for video clips, with an exception for work and study colleagues, to which the interest to share text documents was high. Photos, video clips and many text documents are usually generated by the user and are personal in nature. Hence, the user wishes to control to whom the resource is distributed. The same tendency can, however, be seen for less personal types of content, such as content generated by a commercial provider. The reason may be the cost of resource sharing in the form of bandwidth use and monetary costs, but also because of avoiding problems with intellectual property rights.

The results of Survey 2 shown in Figure 4.6 confirm the above results. The willingness to share (in Figure 4.6) corresponds remarkably well with the interest to download (in Figure 4.3). The willingness to share is slightly lower than the interest to obtain the same information, this difference being larger for unknown people. The difference between content types, shown in Figure 4.7, is relatively small, although the order of content types matches the one of sharing.

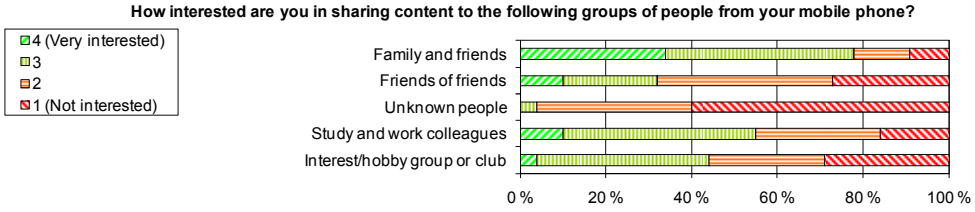


**Figure 4.6. Willingness to share content with different groups of users in Survey 2.**

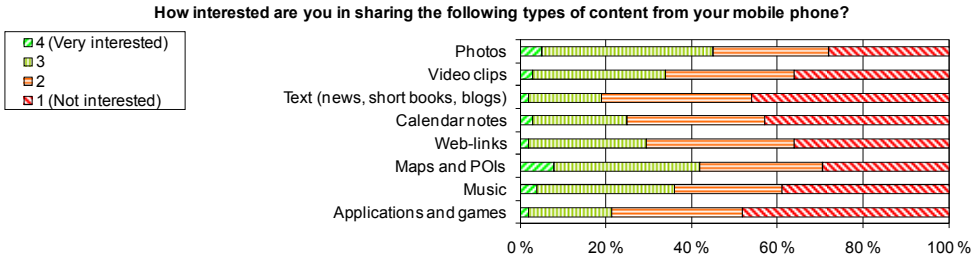


**Figure 4.7. Willingness to share different types of content in Survey 2.**

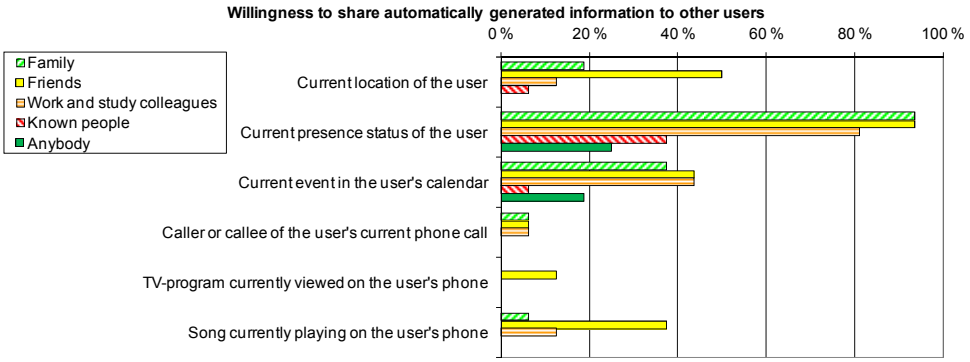


**Figure 4.8. Willingness to share different types of content in Survey 3.**

For the automatically generated status updates in Survey 3, the willingness to share presented in Figure 4.8 is slightly lower than the interest in obtaining the same type of information. Especially for location information and calendar information this asymmetry can be seen.

### 4.4.3 Access control and user groups

Survey 1 compared the user's preferences for different methods of controlling file access and assigning access rights in a file sharing service. Figure 4.9 Shows that most respondents prefer placing shared files in a separate directory. The differences between the suggested methods for assigning access rights to users are small, as shown in Figure 4.10.

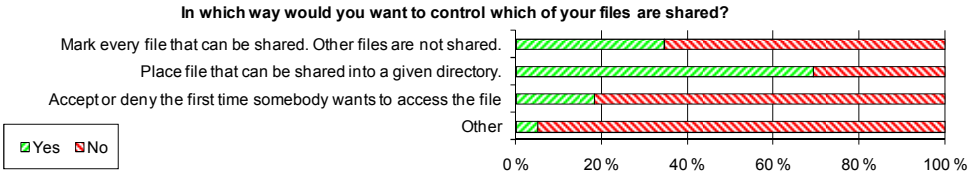


**Figure 4.9. Methods of controlling file access.**

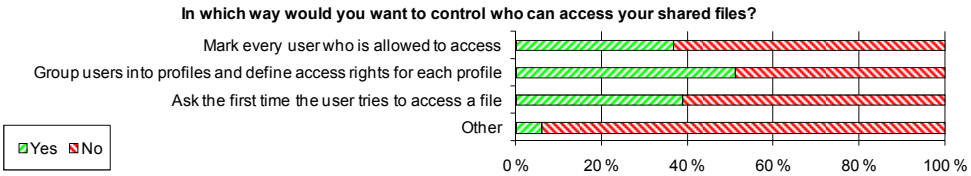


**Figure 4.10. Methods of assigning access rights.**

We asked the respondents to approximate the size of given groups of users. The group sizes of Survey 1 and 2 are presented in Figure 4.11. In both Survey 1 and 2, the median number of family members is 5 and the median number of work/study colleagues is 20. In Survey 1, the median number of close friends is 7 while the median number of other friends is 20. In Survey 2, the types of friends are not separated, whereas the median still is 20. The variance in group size is fairly large. The total of the median sizes in Survey 1 is 52 contacts per user and 45 contacts in Survey 2.

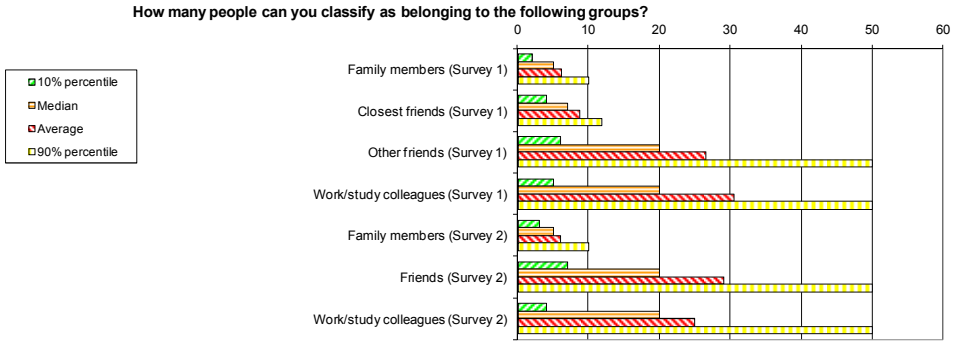


**Figure 4.11. Group sizes.**

We can assume the existence of several types of contacts in addition to the given ones, of which many are related to interests and hobbies. According to Survey 2, 80% of the respondents would participate in 1-5 groups while 4% would participate in 6-15 groups for sharing content. Only 16% of the respondents would not utilize groups. The median size of a group is 20 persons, although the suggested size was millions of users in some cases. The theme of the most important group is hobbies (30%), sports (27%), music (21%), friends and family (13%) and other (9%).

According to Survey 1, the median number (approximated) of contacts in the respondents' phone books is 113. This fits well with the measured median value of 111 contacts reported by Verkasalo [Ver07].

### 4.4.4 Interesting applications

Asked in Survey 1 to evaluate a few given mobile applications, the respondents were mostly interested in an application allowing searching for pictures related to a given keyword (e.g. a public event or incident). The application would be used at least monthly by half of the respondents. Also the downloading of pictures from an event commonly experienced by friends was found interesting, though the willingness to pay for the service was lower. Ringtone downloading is much less interesting. The respondents proposed several applications for requesting information, including traveling, entertainment, sports, education and financial information. In particular, information related to the current location, such as nearby restaurants, was suggested. Furthermore, applications related to social networking, dating, and sharing contact information were proposed.

In Survey 4 the respondents were asked about ideas for applications based on social networks. Several respondents proposed combining social networks with the phone book so the online status and pictures are shown in the phone book. Several also wished that the location of a selected group of friends could be presented. Participants showed general interest in a suggested application showing the profiles of surrounding people, for example, in a party. While a few thought this was frightening, most thought that this would work as a starting point to get familiarized as some background information was available.

### 4.4.5 Constraints

Figure 4.12 shows how various constraints affect the willingness to download and share content in a content sharing service. The respondents of Survey 2 rated the effect on a scale between one (does not reduce the willingness) and five (prevents the use). All constraints were considered more limiting in sharing than in downloading. The battery consumption and the resulting reduction of the operation time of the phone was considered as the least restricting constraint. The most serious constraint reducing the willingness is the cost. Most respondents would limit the use because of the cost involved and some (over 20%) would completely stop using the service. In particular, a cost resulting from sharing was not accepted. The slowdown of the phone has a relatively low influence, although larger than the reduction in operating time. Users are not prepared to wait much longer for a transfer to finish than they are used to on PCs. One of the most important constraints is the lack of access control. Half of the respondents would not at all use a content sharing service that lacks access control. Less than 10% are unaffected by this constraint. Several respondents commented that usability issues, such as screen size, restricted the ability to handle large data amounts. Also privacy concerns were raised.

Survey 1 also asked the respondents about acceptable search delays. As displayed in Figure 4.13, a search delay of $T_s < 2$ s is required by 30% of the users. A search delay of $T_s < 30$ s satisfies 70% of the respondents while a search delay of $T_s < 60$ s satisfies 38% of the respondents.

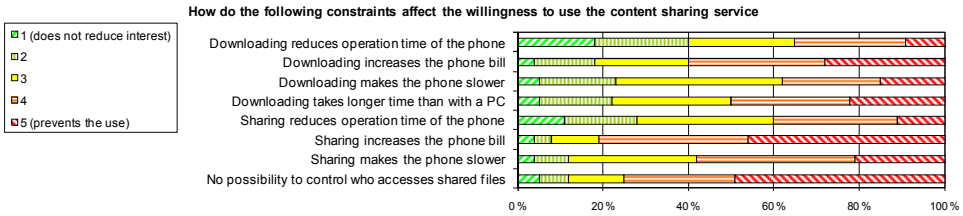


**Figure 4.12. Effect of various constraints on service use.**

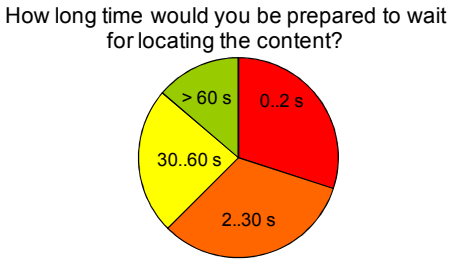


**Figure 4.13. Acceptable search delay.**

### 4.4.6 Cost

In current cellular networks, the user usually pays both for uploading and for downloading. Thus, the user must pay for other people accessing the shared content. As discussed, this is a major obstacle for peer-to-peer applications in cellular networks. The current trend is, however, shifting toward flat rate service. The user pays for a given access rate but the actual transfer is free. Some operators prohibit the use of peer-to-peer software in their networks to prevent massive use, and in particular illegal use. While peer-to-peer systems easily fill up all available capacity, it is often a driver for the user to obtain a fast connection.

Interestingly, Survey 1 showed that the user prefers service based charging (pay per downloaded item) to a fixed monthly sum (including everything) in almost all of the proposed usage scenarios. Roughly one fourth of the respondents would only use the proposed services if they were free. About 41% of the respondents would not pay for content created by other users. Equally many respondents would be prepared to pay, but less than for professionally created content. About 16% of the respondents give equal value to user-created content as to professional content. Very few users (2%) would pay more for user-created content.

In traditional peer-to-peer networks, most users only download content without sharing anything. Measurements in [For07] show that 53% of the nodes stay online less than 10 minutes and 62% stay online less than 15 minutes. This behavior, called free-riding or leeching, is

expected to be more common in the mobile scenario, where the cost of uploading data is higher. The uploading user consumes battery power and must often pay for the upload traffic, for which he personally does not benefit. However, the user might have a higher incentive to provide resources for publishing content created by himself than to provide extra copies for public and even illegally distributed material.

### 4.4.7 Considerations

For the respondents, it can be difficult to evaluate a service from which no personal experience has been gained. Therefore, the survey results mainly show the initial opinions and directions for development. The actual attractiveness of a mobile peer-to-peer service is visible only when it is publicly available.

According to Survey 1, most respondents would use a resource sharing service at least sometimes, and around 10% of the respondents would use it often. Only about 15% of the respondents are not interested at all. The interest is higher for respondents with much experience of fixed peer-to-peer software and among younger respondents.

Asking the respondents about their reasons for using mobile peer-to-peer gives about the same number of responses for the following given reasons: a possibility to purchase the content cheaper from a peer-to-peer based service, a more convenient way of obtaining content, a possibility to get hard-to-find content, and a possibility to get very recently created content. Very few respondents saw the possibility to get new friends as a reason. A large range of other reasons were suggested, a large part of which relate to sharing content within a community and obtaining illegal content.

The surveys show the importance of knowing the people with whom you exchange information. The closer the relationship, the higher the willingness to share resources with the other user. But a close relationship is also important in accessing resources; users are more interested in obtaining resources created by known people than those created by unknown people. The exception is commercially provided resources, which may have a higher quality and a more general audience. To support the sharing of user-generated content, the sharing application should therefore allow forming user groups and control the access to resources within the group. Alternatively, the contacts can be modeled as a social network, whereas the access rights are determined by the contacts and distance. Additionally, it may be feasible to integrate access to commercial content in the same application.

## 4.5 Architectures for operator-controlled peer-to-peer services

In this section, we consider the technical implementation of a resource sharing service in a cellular network. We assume that the service is provided either by operators or separate service providers. Henceforth, we use the term operator for both cases. Several operators may combine their services, forming a multi-operator resource sharing system. Combining the services increases the amount of available resources and, according to Metcalfe's law [Kir98], the value and attractiveness of the

system. We assume that overlay links can be established between a pair of operators only if they have a formal agreement. The peer-to-peer service forms a generic platform on top of which different applications can be built. In order to support a broad range of possible applications, we require the system to support complex searches while providing deterministic search results. As structured systems currently do not support complex queries adequately, we concentrate on unstructured systems.

### 4.5.1 Two-layer hierarchical architecture

The bottle-neck in a cellular network is the limited bandwidth and high cost, both for the user and the operator, of the wireless link. The primary technical requirement is therefore to minimize the traffic over the wireless link, which also reduces the consumption of battery power and other resources.

Direct communication between nodes in a cellular network is not possible. In GPRS networks, all communication goes through the Gateway GPRS Support Node (GGSN), making a true peer-to-peer architecture in the lower layers impossible [MCB+04]. Consequently, a message sent between two cellular devices traverses the air interface twice. Moreover, firewalls and NATs contribute to making direct communication between cellular devices difficult or impossible.

The choice of architecture has a major influence on the traffic. In a completely distributed system without indexing all queries must be distributed to all terminals. As terminals flood the query to each other, every hop traverses two wireless links. Due to the inefficiency of flooding, queries traverse the wireless links multiple times. Utilizing indexing, the total traffic can be minimized. However, the fundamental problem is still the cost of communicating between terminals.

From the network perspective, traffic is minimized using centralization. Centralized nodes can be maintained either by the operator or by other users. Although this choice is irrelevant from the technical viewpoint, it influences the reliability and possibility to control the network. Given the operator's need for control, the approach of placing the centralized elements in the operator's network or a third-party service provider network is more realistic. Centralized elements among the mobile nodes create uneven distribution of load and battery consumption and it is difficult to give good reasons why a user should receive more traffic than other users. Centralized elements in the fixed network reduce the traffic from the perspective of both the network and the terminal. A server maintained by an operator can have a significantly higher bandwidth and processing power, and a constant power supply. As we consider multi-operator scenarios with multiple points of control, we need to introduce several centralized points – at least one per operator. All cellular devices should be loaded equally, while also the load between operators should be distributed.

These considerations lead to the hierarchical architecture shown in Figure 4.14, in which the upper layer consists of servers in the operator network and the lower layer of mobile terminals. The architecture is suitable for cellular networks, since terminals are usually behind NATs

and firewalls, which complicates direct application layer communication between terminals. The connections between the centralized elements are at this stage undefined: practically any type of peer-to-peer architecture can be used within the operator's network and between different operator's networks.

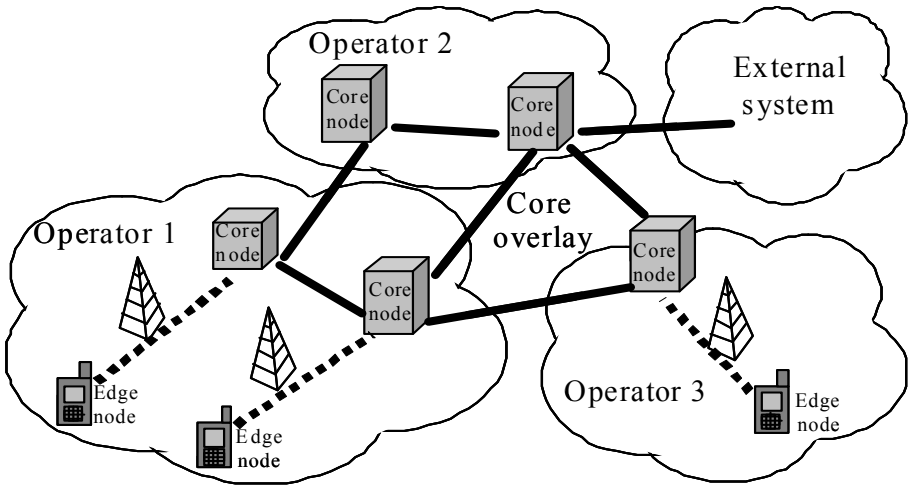


**Figure 4.14. Architecture for hierarchical operator-controlled peer-to-peer service.**

For clarity, we call the centralized elements in the fixed network *core nodes*. The overlay network between the core nodes is called the *core overlay*. Optionally, there can be several core overlays, for instance, for different applications or types of operator agreements. Links in the core overlay between elements of the same operator are called *interior core links*. Correspondingly, an *exterior core link* connects the elements of two different operators. Exterior core links are formed based on formal agreements between operators. The mobile nodes are called *edge nodes*. There are no overlay links directly between the edge nodes. The edge nodes form the lower hierarchical level and the core nodes form the upper hierarchical level. The core nodes and edge nodes differ in several significant ways as indicated in Table 4.1.

**Table 4.1. Key characteristics of edge and core nodes.**

| | Core node | Edge node |
|---|---|---|
| Responsible party | Operator | User |
| Storage capacity | High | Low |
| Processing power | High | Low |
| Power supply | Unlimited | Battery |
| Bandwidth | High | Low |
| Connectivity restrictions | None | NATs/Firewalls |
| Stability | High | Low |

An edge node is configured with the address of one or several core nodes in its network. Alternatively, there can be a bootstrap server in the network providing configuration information. An edge node selects one core node, to which it connects. A connection procedure similar to the one proposed in Section 3.4.6 can be used to allocate the edge node to the least loaded core node. Optionally, an edge node can be simultaneously connected to several core nodes. The edge node does not need to know the architecture or topology of the core overlay.

A mobile peer-to-peer system can be connected to an external peer-to-peer system, for example, to Kazaa. This allows the mobile user to access resources in the external system and also to provide resources to users in the external system. In this scenario, special care has to be taken to avoid loops. A query sent to the external system may be received later by the same or another core node in the mobile peer-to-peer system. To prevent loops there must be a search identifier common to all systems. Unless there is a global search identifier, a query received from the external system must not be forwarded within the mobile peer-to-peer system.

Compared to traditional peer-to-peer sharing, the cellular infrastructure allows for more control. Because of the centralized control, the operator may limit illicit material and reduce the risk of security attacks. As all users can be identified, the incentive for illegal sharing is reduced and evidence for possible legal action is available. However, since the content itself is not in the operator's network, the control must be based on the metadata describing the content. Analyzing the content distributed through the operator's network is typically a heavy process.

As the traffic passes through the fixed network, it is possible to improve performance by caching. We can distinguish between two types of caching: the caching of search requests and the caching of resources. The former implements an implicit form of indexing. Resource caching allows popular content to be replicated on a central server. This allows the system to adjust toward a client-server operation, combining the best parts of both worlds.

### 4.5.2 Implementation in IMS

In Publications [BML+05] and [MBL+06b], we present a commercial peer-to-peer service for the IMS network. As IMS utilizes SIP signaling, we propose using SIP also for P2P control messages, as described more comprehensively in Section 4.7. Services in SIP are generally hosted and executed by SIP Application Servers (SIP-AS) defined in 3GPP Release 5. Therefore we implement the core node as an application server called the P2P Application Server (P2P-AS). Utilizing an application server makes the P2P service similar to other IMS services, allowing the charging and security provided by the IMS infrastructure to be reused. As depicted in Figure 4.15, the P2P-AS is connected to the S-CSCF (Serving Call/Session Control Function) through the ISC interface based on SIP signaling. The S-CSCF is connected to a P-CSCF (Proxy Call/Session Control Function) through the Mw interface, also using SIP signaling. The S-CSCF and P-CSCF are fundamentally SIP proxies enhanced with IMS-specific functions. The P2P-AS can store persistent information

about the users in the HSS (Home Subscriber Server) between active sessions. The ISIM provides user authentication. Supplementary functions, such as prioritizing of resources, load balancing and caching can be implemented in the P2P-AS.

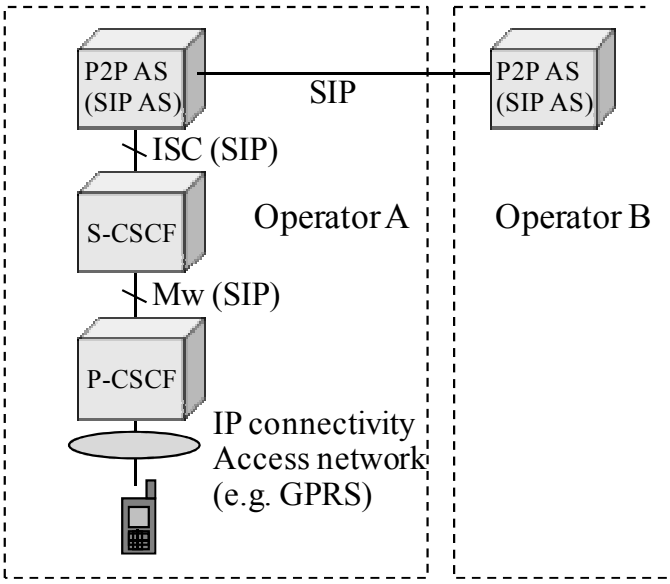


**Figure 4.15. Peer-to-peer resource sharing service in the IMS.**

The overlay network is formed when several P2P-AS are connected together. We specify the following main functions of the P2P-AS:

1. *Edge connection maintenance*. This function involves handling connection requests from edge nodes, and optionally allocating the edge node to another P2P-AS better capable of serving the connecting edge node. Before establishing the overlay connection, the user is authenticated and authorized. The function further handles error conditions, mobility, connection teardown and ungraceful exits.
2. *Edge index maintenance*. This function involves maintaining a remote index of resources shared by the connected edge nodes. The P2P-AS may require the index to be refreshed periodically and may further limit the index size.
3. *Core overlay maintenance*. This function maintains the core overlay as specified by the used architecture. The operator may specify additional policies, for example, regarding the number of peer connections for load balancing. The core overlay includes both interior and exterior core overlay links. When a core link specified by the architecture is missing, a new core link is establishing by sending connection requests to another P2P-AS. Incoming connection requests are compared to the architecture requirements and the policy, and the other party is authenticated and authorized. The connection quality and traffic are monitored.
4. *Core index maintenance*. In applicable architectures, this function maintains a remote index of resources reachable through other P2P-ASs. Correspondingly, the P2P-AS's own index information is distributed to other P2P-ASs.

5. *Search distribution*. This function involves forwarding search requests received from edge nodes and other P2P-ASs according to the search algorithm. For matches, search replies are generated. Depending on policies, these replies may be sent directly to the requester or accumulated in intermediary P2P-ASs.
6. *Usage recording*. This function creates usage records for the purpose of charging and billing. The operator may also compensate the user for providing resources or redundant copies in order to motivate sharing.

In addition to these, the P2P-AS may perform optional functions:

7. *Caching*. This function stores a copy of the transferred content in a content cache. The content cache is a logical element that is inserted as a proxy on the transfer path. The function may maintain statistical information about the popularity of items for efficiency reasons.
8. *External connectivity*. This function maintains a connection to an external P2P network. The P2P-AS translates between the protocols on each side. Search requests are forwarded from and to this network.
9. *Commercial providers*. This function connects the peer-to-peer network to a provider of commercial content. The search requests are compared to the content provided by the commercial provider. The function may also provide methods to charge for accessing commercial content.
10. *Content control*. This function inspects the index information and search requests in order to impede the sharing of illegal content, or the sharing of which is restricted by intellectual property rights (IPR) or policies. The function detects malicious use of the network, such as spamming and providing false resource descriptors. It may also search for viruses and other malicious software. An advanced function can also compare the index with a database of IPR. It may also be feasible to inspect the actual content as it is transported through the network, although this is complicated and resource demanding.

The P2P-AS collects application level usage records that form the basis for charging. This allows flexible pricing models based on the service instead of data- or time-based charging. Pricing models based on the actual transferred content is assumed to be more understandable by the user than a model where the user is charged for all traffic, including overlay maintenance, signaling and search traffic. The user is then able to better estimate the cost of resource sharing. A scheme that charges for signaling and maintenance traffic creates an incentive to close the application between resource downloading sessions, making resources unavailable. On the other hand, free uploads motivate users to share and the higher amount of available resources pays back as higher service use. Pricing models that reward users for sharing resources are also possible. In general, it is important to distinguish between signaling (overlay maintenance, search and index distribution) and resource access traffic.

### 4.5.3 Group management and access control

In our user study we identified the need for access control: the resource provider may require the availability to be limited to known people, and the resource requester may prefer to limit the search scope to include only known people. One way to implement access control is by using groups. A user can be a member in one or several groups. The shared resources can be marked with the groups to whose users it is shared. Search requests can be marked with a set of groups, which limits the results to resources in the given groups. Groups are identified by their names. Soinio [Soi09] identifies four kinds of groups:

1. *Local groups*. Personal groups defined by users with membership not visible to other users.
2. *Symmetric personal groups*. Personal groups defined by users with membership visible to (at least) all members. Membership is symmetric, i.e. the added person must accept the inclusion to the group.
3. *Asymmetric personal group*. Personal groups defined by users (normally) with global visibility.
4. *Global groups*. System groups typically defined by a system administrator with global visibility.

#### Access control classes

We denote the set of groups available in the system as $G = \{g_1, g_2, \ldots, g_n\}$. The group membership of a user $u$ is $M_u$, where $M_u \subseteq$ G. The groups to which a resource $r$ is shared is $G_r$. The groups to which a search request $s$ is directed is $G_s$. Let us now define different levels of access control. These are defined based on the search outcome independently of how and where the access control is implemented.

**Definition 4.1.** A resource discovery system provides *indicative access control* if every resource $r$ that is shared to a set of groups $G_r$ is excluded from the search reply when $G_s \cap G_r = \varnothing$.

**Definition 4.2.** A resource discovery system provides *read access control* if every resource $r$ that is shared to a set of groups $G_r$ is excluded from the search reply of a user $u$ when $M_u \cap G_s \cap G_r = \varnothing$.

**Definition 4.3.** A resource discovery system provides *write access control* if every resource $r$ that is shared by user $v$ to a set of groups $G_r$ is excluded from the search reply when $M_v \cap G_s \cap G_r = \varnothing$.

**Definition 4.4.** A resource discovery system provides *read-write access control* if every resource $r$ that is shared by user $v$ to a set of groups $G_r$ is excluded from the search reply of a user $u$ when $M_u \cap M_v \cap G_s \cap G_r = \varnothing$.

Indicative access control can only be regarded as a guide for easier location of the desired resources, as the system does not check whether the requester $u$ is a member in any of the groups $G_r$. Indicative access

control works as an access control system only if the authorization is checked in the access phase separately. Read access control, on the other hand, guarantees that only resources shared in groups that the requester is a member of are located. Write access control ensures that the resource provider is a member of the groups to which the resource is published.

Note that the resource discovery system is not responsible for controlling the access to the resource itself. Thus, if the resource has been found with means outside the resource discovery system, the resource could still be accessed. However, this situation can easily be corrected if the resource provider generates a secret identification code, a *token*, that is included in the resource descriptor. The resource provider grants access to the resource only if the token is provided in the resource access message. A hash identifying the file can be an adequate token.

### Group management

Group management means the control of the group membership $M_u$ of the users $u$. Group management functions include adding members to the group, removing members from the group, and defining rights and roles for members. Also the definition of the scope and properties of the group can be regarded as group management functions. We propose three different models for managing groups:

1. *Role based group management*. In this model, the right to manage the group is assigned to members with given roles. For example, a group administrator may be responsible for managing the group. The administrator is typically the user that created the groups, and the current administrator can entrust the administrator role to other users.
2. *Collaborative group management*. Group management decisions are collaboratively made, e.g. using voting. The aim is to make decisions in a democratic way. One way is to require acceptance of a given percentage or number of users for a decision, such as accepting a new member.
3. *Distributed group management*. All users have an equal right to make decisions. This model can be implemented so that all users manage their own view of the group and the group is implicitly built from the sum of the views of all participants. Such a model approximates a social network.

Depending on the model, group management functions include functions for applying for membership in a group, inviting a user to a group, collecting votes from users, etc. A (distributed) group membership database is used to store the group membership $M_u$ of all users $u$. The database either maps the user into a set of groups that the user is a member of, or alternatively maps the group name into a set of users that are members in this group.

### Implementation in a operator-controlled service

A centralized implementation of the system allows all models to be implemented in a straightforward manner. Users are assumed to trust the core node and that the core node implements access control correctly.

Most currently available resource sharing systems that support access control are centralized.

Where the architecture includes several interconnected core nodes, the implementation becomes challenging, both in terms of distributed group management and distributed search and indexing. The implementation varies depending on whether the core nodes trust each other. We assume that a relationship of trust is formed as a consequence of the agreements between peering operators. This assumption makes the implementation simpler. The lack of trust most likely leads to the use of cryptographic methods.

In a simple implementation index updates and search requests are distributed normally. The access control is performed locally at each edge node by comparing the group list $G_r$ of the resource descriptor $r$ with the group list $G_s$ in the search request $s$. Only indicative access control is achieved in this way, and the requesting user can obtain access to any desired resource. Indicative access control is feasible if the resource access is authorized separately. To provide read access control, the system must ensure that the search request is directed to only groups of which the resource requester is a member. This is done by querying the group membership database. To provide write access control, the system must check the group membership database when resources are published.

The group membership database can be centrally maintained or replicated to all core nodes. Alternatively, each core node hosts a given set of groups and maintains the fraction of the database describing the membership of the hosted groups. In this case, the group name indicates the core node hosting the group, for instance, in the format group@operator.com. The group membership database can also be distributed using other means, such as DHTs, but this adds an additional overlay to the system.

Ideally, the search request should be distributed only to the edge nodes with group members in order to reduce the traffic. Such an implementation is, however, challenging, especially when flooding is used in the core overlay. Nodes must know which other nodes have members belonging to a given group. In implementing this knowledge, the system easily reduces to a centralized system or to a global index system. We return to the problem of flooding according to group membership in Section 4.6.4, where a slightly different approach is used.

The work has been continued by Soinio, which has produced a comprehensive study of access control in peer-to-peer systems [Soi09]. Soinio proposes three solutions for access control: distributed access control lists, service passwords and local delegations.

### 4.5.4 Architectures for the core overlay network

The core overlay connects the core nodes together. Practically any architecture can be used in the core overlay. In this section we evaluate the suitability of different architectures for the core overlay in a commercial resource sharing service. In all architectures, we assume that users are connected as edge nodes to the core nodes.

### Desired properties

So far the choice of architecture has been driven by performance issues, such as the need to reduce the message overhead, processing load and delay. The architectural choice for a commercial resource sharing service is influenced by several factors in addition to the performance. First, as each operator may have several core nodes, the architecture should be able to distinguish between interior core links and exterior core links. Exterior links can be assumed to have a higher cost, a lower bandwidth and an additional processing overhead because of firewalls and policies. It is therefore desirable to reduce the traffic on the exterior core links.

Second, the number of peer operators should be limited. Each peering relationship is assumed to be based on a formal contract with the traffic measured and controlled. Also the cost of the exterior link motivates limiting the number of peer operators. In particular, it is unreasonable to require all operators peer with all other operators, especially in a large scenario.

Third, the operators generally desire to keep certain information private, such as information about the internal topology, the customers, and the resources shared by the customers. As a consequence, the indices of an operator's customers should not be distributed to the networks of other operators. We call this property index confidentiality. Index confidentiality also implies that the indices of one operator do not consume storage resources in another operator's network. An operator does not need to increase its capacity when the competitor's index grows or if more operators join the system. Likewise, the operator should be able to choose the topology of the interior core links independently of other operators. We call this property topology autonomy. Finally, the topology should not be revealed to other operators. This property is called internal topology confidentiality.

### Flooding

In the simplest case, the core overlay has an arbitrary topology, whereas the architecture is semi-centralized. In publications [BML+05] and [MBL+06b] we present a peer-to-peer service based on this architecture. Searching is performed using flooding or random walk. The search algorithm does not distinguish between interior and exterior core links, and a query may traverse operator boundaries multiple times, which is ineffective as the cost of exterior links is high. The semi-centralized architecture is only feasible if an operator has one or a few super-peers. When the number of super-peers is high, the inefficiency of flooding becomes apparent and other architectures become more feasible. A search request must be forwarded to all core nodes. Therefore, adding core nodes increases the index capacity but not the search capacity. Index confidentiality is provided as no indexing is used within the core overlay. The internal topology of an operator is not revealed. Only nodes with exterior links are visible to the outside.

### Clustered architectures

In clustered architectures, such as PIC, PSC and IPIC, it is feasible that each operator maintains a cluster. The operator can then control the topology and the distribution method used within the cluster. The operator also controls the addition and removal of core nodes in the cluster. We do not find it feasible that several operators share a cluster, as this breaks the autonomy regarding control and topology choice. Instead, it may be advisable to divide an operator's network into several clusters if the network size grows very large. As we found in Section 3.4.10, the clustered architectures perform best in terms of load distribution when the sizes of all clusters are similar. As each node requires an exterior link, the number of nodes in the cluster is revealed to all operators. However, the topology and the distribution algorithm used within the cluster are not revealed.

In PSC, all exterior core links are index links and all interior core links are search links. A PSC architecture does not provide index confidentiality as the index is distributed to one node in each cluster. The operator can increase the storage capacity by adding new core nodes to the cluster. However, it is difficult to add capacity for handling search requests. Search capacity can only be increased by adding clusters.

Because of the lack of index confidentiality, we consider architectures based on PIC and IPIC more feasible for a commercial scenario. All exterior core links are search links and all interior core links are index links. PIC provides index confidentiality since the index of the operator's customers is distributed to the nodes in the operator's own network but not to any competing networks. PIC allows the operator to easily add capacity for handling search requests by adding more core nodes, which unloads the existing nodes. The index capacity is more difficult to enlarge as each node must store the index of the whole cluster. Index capacity is added by splitting a cluster into several clusters.

The problem with the PSC and PIC architectures is the requirement of full connectivity between clusters. Exterior links are set up based on agreement. The establishment and management of these links is a heavy process. We therefore consider the requirement of full connectivity between clusters as incompatible with scenarios involving several operators, especially when the number of operators is high. For these scenarios we proposed a new architecture, IPIC, for the upper hierarchical layer. While reducing exterior links, it suffers from a slight performance degradation, either as an increased search delay or higher bandwidth use. Compared to flooding, IPIC provides a significant improvement in performance and a controlled flow of information in the network. Like the other clustered architectures, the topology and distribution method used within the cluster remain confidential. However, the number of nodes in the cluster is revealed to the peer operators.

As PIC, PSC, and IPIC demand a maintained topology, a method to construct and maintain the topology is required. Generally, construction requires the following steps:

1. Dividing nodes into clusters.
2. Interconnecting all nodes within the cluster with index links (in PIC and IPIC) or search links (in PSC).
3. Interconnecting all clusters so that each node has a search link (in PIC and IPIC) or index link (in PSC) to at least on node in each cluster.

In our case, the first two steps are manual: the operator divides the nodes into one or several clusters and arranges the index or search distribution between the nodes using any method. For the third step, we can use the method outlined in Section 3.4.6 that allocates new links to the node with the lowest indegree. A node $v$ needs to know at least one node $w_C$, the ingress node, for each peer cluster $C$. The ingress node is a node that externally represents the cluster, and is known by the neighboring operators. The same procedure can be used to connect edge nodes to core nodes.

### Ring architectures

A ring topology ensures distributing a message to each node only once. The ring structure is easy to create and maintain, using e.g. the topology maintenance algorithm in Section 3.5. Exterior traffic is reduced if all nodes of an operator's network are consecutive in the ring. An operator then has only two exterior links and the internal nodes are not revealed to other operators. In a simple solution, a search message is forwarded between the edge nodes and no index distribution is used. For two reasons the solution is, however, not scalable: (1) the search capacity cannot be increased by adding nodes as all queries are forwarded to all nodes and (2) the search delay increases linearly with the network size.

A more advanced version of the ring is the Zone Indexing algorithm. In a normal implementation of Zone Indexing, the index distribution is not limited to the operator's own network. Moreover, the internal topology is disclosed to the successors through the index updates. However, this disclosure is only limited to a small number of operators.

The Zone Indexing algorithm can be modified so that index updates are dropped at operator borders. This corresponds to zones that include only the operator's own nodes. Unfortunately, this affects the performance negatively. The zone is very small for the last nodes of the operator's part of the ring – the last node has a zone of size one. This may be a high cost for providing index and topology confidentiality.

Adding nodes to a Zone Indexing network increases the search and index capacity of the whole system. However, the benefit of the increase is not limited to the operator that adds the new nodes.

### Tree architectures

Since the topology of the upper hierarchical layer is fairly static, more rigid topologies can be used. A tree has the desired property that there is a single path between every pair of nodes. Broadcasting on a tree is simple: a node forwards a message on all links of the three, except on the one over which the message was received. There are no loops in a tree.

Broadcasting is also optimal: each node is reached only once. In practice, it is only feasible to forward search messages on a tree.

A tree can be manually formed. Such a tree could consist of a upper-layer tree connecting one node from each operator's network. From this node a sub-tree connects all nodes within the operator's network.

Spanning trees can also be automatically built of an arbitrary topology, from which an algorithm selects a subset of overlay links as belonging to the tree. Links that do not belong to the tree are disabled. Automatically formed spanning trees are used by Ethernet bridges, where the tree is automatically generated with the Spanning Tree Protocol (STP) [IEEE1990] or Rapid Spanning Tree Protocol (RSTP) [IEEE2004]. Similar algorithms can be used for connecting the nodes in the core overlay. Automatically formed trees can recover from node or link failures. Two critical issues must be considered in generating a tree for forwarding search messages. First, the diameter, i.e. the maximum distance between two nodes, affects the search delay and should therefore be minimized. The diameter in a tree with $N$ nodes can be anything between 2 and $N$ - 1. Second, load balancing is difficult. Although each node receives a message only once, the times the message is forwarded depends on the degree. As leaf nodes have a degree of one and do not forward a message, perfect load balancing cannot be achieved. Both these properties are determined by the topology on which the tree is based as well as the choice of root node. Adjusting the topology for the purpose of shaping the tree is as heavy a work as configuring the tree manually. The automatically generated tree does not separate between interior and exterior links, and therefore the internal topology is revealed.

### Multicast

Searching in peer-to-peer networks is based on distributing a message to a set of nodes. This is also the aim of multicast algorithms such as the Distance Vector Multicast Routing Protocol (DVMRP) [WPD88] and Protocol Independent Multicast Dense Mode (PIM-DM) [ANS05]. For each sender a separate tree is generated and the path length from each sender is minimized. The load is well distributed. However, using multicasting in the overlay would require the overhead of a separate routing protocol operating at the overlay level. At the network level, multicast provides an interesting alternative. However, network level multicast is currently not widespread and it is questionable if there will be multicasting between operators in the near future.

### Comparison

Table 4.2 and Table 4.3 summarize the properties of the discussed overlay architecture. The first five properties are the policy properties defined as in the beginning of this section. Search overhead is considered as low when it is under O($N$), medium for O($N$), and high for O($kN$) with a constant $k$>1. Search delay is considered as low for O(1), high for O($N$) and medium when it is between these extremes. Search and index capacity expandability refer to the operator's ability to add nodes in order

to handle more search requests or store more index information, respectively.

None of the architectures are perfect. While search flooding provides all desired policy properties, the low performance limits the usefulness of the solution, especially in terms of search capacity expandability. The search ring shares the same properties, although the feasibility suffers more from the high delay than from a high overhead. PSC and Zone Indexing, while being efficient, are less suitable for a competitive environment as they do not provide index confidentiality. PIC is the most efficient, but requires full connectivity between operators. When full connectivity is impossible, IPIC is a more suitable architecture despite having slightly lower performance. We argue that this overhead is acceptable considering the other alternatives. IPIC also allows index capacity to be expanded by splitting a cluster, which can be implemented without constructing new relations with all other operators. Spanning trees, while being interesting options for further development, provide a limited search capacity expandability as no indexing is used. Automatically generated spanning trees fail to consider policy requirements.

In practice, the clustering architectures can be seen as tree-layer hierarchical networks, where the uppermost layer is the inter-operator topology, the following layer is the intra-operator topology and the lowest layer the edge nodes. Also spanning trees with separate sub-trees for interior core links show this property.

**Table 4.2. Properties of the core overlay architectures from operator's viewpoint.**

| Architecture | Flooding | PIC | PSC | IPIC |
|---|---|---|---|---|
| Separation between interior and exterior links | No | Yes | Yes | Yes |
| Index confidentiality | Yes | Yes | No | Yes |
| Topology autonomy | Yes | Yes | Yes | Yes |
| Internal topology confidentiality | Yes | Partial | Partial | Partial |
| Limited number of peer operators | Yes | No | No | Yes |
| Search overhead | High | Low | Low | Low / Medium |
| Search delay | Medium | Low | Low / Medium | Medium |
| Search capacity expandability | No | Yes | Limited | Yes |
| Index capacity expandablity | N/A | Limited | Yes | Yes |

**Table 4.3. Properties of the core overlay architectures from operator's viewpoint.**

| Architecture | Search ring | Zone Indexing | Spanning tree 1 * | Spanning tree 2 ** |
|---|---|---|---|---|
| Separation between interior and exterior links | Yes | Yes | Yes | No |
| Index confidentiality | Yes | No | Yes | Yes |
| Topology autonomy | No | No | Yes | No |
| Internal topology confidentiality | Yes | Partial | Yes | No |
| Limited number of peer operators | Yes | Yes | Yes | Yes |
| Search overhead | Medium | Low | Medium | Medium |
| Search delay | High | Medium | Medium | Medium |
| Search capacity expandability | No | Partial | No | No |
| Index capacity expandablity | N/A | Partial | N/A | N/A |

* Manually configured spanning tree with separate sub-tree for interior core links.

** Spanning tree build from an arbitrary (random) overlay topology.

## 4.6 Peer-to-peer with decentralized control

A decentralized peer-to-peer system allows mobile users to share resources without the control of an operator or service provider. The overlay can be formed by the users without any external support and service subscription. This scenario is interesting if the operator or a third-party service provider do not intend to provide the desired service. It can further be used for forming overlays specific to certain applications, collaborative projects or user groups.

As discussed in Section 4.4.6, the cost of uploading discourages most users from sharing. Unless there is a high will to share all costs, decentralized peer-to-peer systems practically require flat-rate charging to maintain an incentive to share. Fortunately, it looks like flat-rate charging will become the prevalent charging scheme in a few years. Furthermore, the cellular environment calls for a uniform load distribution, which excludes the use of super-peers and centralized servers among the nodes. Centralized control must also be avoided because of the possibly high unavailability and churn. In this section we study architectural solutions for such a decentralized peer-to-peer network. Since decentralized peer-to-peer networks are unlikely to scale globally, we are interested in utilizing groups to limit the distribution.

### 4.6.1 Scenario

In order to test the concept of fully distributed resource discovery in peer-to-peer systems, we apply it to a particular area, where we see it as a feasible solution. The application area we consider is the sharing of

resources within relatively small-scale groups of users in a mobile network. We consider a scenario based on the following assumptions:

1. *Social network*. The overlay topology is based on the social contacts of the user. The contacts are a selected subset of the contacts in the address book. As each contact is a neighbor in the overlay, the average degree $D$ of the nodes is high.
2. *Stable topology.* As the topology is based on the social network, it is relatively stable. This is in contrast to traditional peer-to-peer systems, where the topology is based on the devices and the stability is determined by the churn rate of the devices. A node that is offline still exists in a social topology but the offline status is separately considered. We assume that a fairly small percentage of the nodes are offline at a given moment.
3. *Groups*. As identified in Section 4.4, users prefer to share their resources within a group. In a mobile network, we assume that the shared content is mostly generated by the user and therefore is rather personal in nature. Furthermore, providing access to the shared resource incurs expenses, loss of battery power, and consumption of available bandwidth. Legal restrictions, such as the fair use concept, may reduce the willingness or possibility to share content with unknown users. Therefore, the objective is that the access to a resource can be limited to certain groups. Section 4.4 also shows that users are more interested in resources provided by known people. As groups are formed based on the common interests and social relationships, users are more likely to find relevant interest. Based on Figure 4.11, the median size of a group based on personal contacts is between 5 and 20, but groups based on interests and hobbies can be assumed larger.
4. *Frequent searches, infrequent updates.* During a session of active use, a user performs search requests in bursts, for example, for finding photos with various motives. The frequency $f_s$ is assumed to be in the order of a few sessions a day, each session involving several searches. New pieces of content are published relatively infrequently, such as when photos are published. The frequency $f_{i,modification}$ is expected to be less than one modification per day.

### 4.6.2 Social network model

We propose a method for creating groups based on the contacts of a user. These contacts are available in the phone book of the user. Although the phone book does not cover nearly all persons known by a user, it represents the set of users that are frequently contacted. Kuitto [Kui02] reports significant overlapping between the contacts in a user's phone book with the user's real social network, and that only in a few cases important contacts are not in the phone book. We model the social network formed by the phone book contacts as a graph $G = (V, E)$ where $V$ is the set of users and $E$ is the set of contacts between users. The graph is directed, since a contact from user $v$ to $u$ does not imply a contact from user $u$ to $v$.

Many phones already allow categorizing the contacts into different groups or profiles. In current phones, the incentive for a user to categorize contacts is rather small, since the added value is often limited to assigning a common ring tone to the members of a category. Lugano [Lug08] points out that many important features in the user interface, including the phone books, have not improved since old phones: phone books do not integrate a user profile nor social networking features and mechanisms supporting sharing, searching and filtering of data with other users. We propose to attach a set of textual group names, called *categories*, to contacts in the phone book. Contacts can be categorized into family, friends, work/study colleagues. Additional categories can be related to, for instance, hobbies, activities and communities. We denote the set of categories of a contact $e \in E$ with $C_e$. The categorization is used as a basis for our definition of groups.

**Definition 4.5.** The *category graph* of category $c$ is $G_c = (V, E_c) \subseteq G$ where $E_c = \{e \in E \mid c \in C_e\}$.

**Definition 4.6.** A *group* $c$ of a user $v \in V$ includes every user $u \in V$ to which there is a path from $v$ to $u$ in $G_c$.

According to this definition, the group membership depends on the user observing the group. In a directed graph $G$ the groups are asymmetric, whereas they are symmetric in an undirected graph. Observe that the network may contain several separate groups of a given category, i.e. different users may see different groups with the same name. It is therefore important to separate the concepts group and category. For the purpose of the following discussion we define a few more concepts.

**Definition 4.7.** The *social distance* $dist_c(u, v)$ between two users $u \in V$ and $v \in V$ in a category $c$ is the minimum number of edges needed to connect $u$ and $v$ in $G_c$.

**Definition 4.8.** The *connectivity* $d_c(u)$ of a user $u \in V$ in a category $c$ is the degree of $u$ in $G_c$.

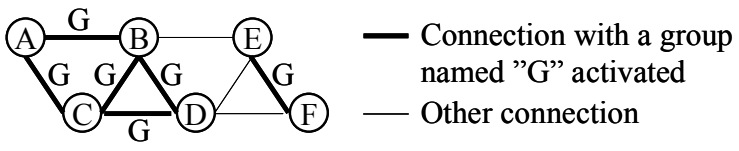


**Figure 4.16. Social network with groups of category G.**

The example in Figure 4.16 shows a social network with the undirected category graph of category G marked with thick lines and the symbol G. Nodes A, B, C, and D are members of a group named G. Nodes E and F are members of a different group with the same name G. The social distances $dist_G(A, D) = 2$ while $dist_G(A, E) = \infty$. The connectivity of B in G is $d_G(B) = 3$.

### 4.6.3 Group management and policies

In the given scenario there is no centralized control and no node that can store a group membership database. Furthermore, part of the users may be offline. These factors make role based or collaborative group management difficult to implement in an efficient manner. The former requires a trusted node serving the membership database. The latter requires querying all (or a given percentage) of the group members for a collaborative decision. Instead, we propose a new method for distributed group management. This method has been published in [Bei07b].

#### Distributed group management

Groups are formed based on the categories the user specifies for each contact, for instance, using the phone book. A user $u$ specifies a set of categories $C_{uv}$ for the connection (contact) to a user $v$. While two users can specify different sets of categories for the connection between them, only the set of common categories are activated. The groups are thus symmetric. For instance, if user A marks user B as member of groups $C_{AB} = \{G_1, G_2, G_3\}$ and user B marks user A as member of groups $C_{BA} = \{G_2, G_3, G_4\}$, then the groups activated on connection $e$ between A and B are the groups common to both users, $C_e = C_{AB} \cap C_{BA} = \{G_2, G_3\}$. To simplify coordination, categories added on a connection are in a practical implementation displayed to the other user and the other user is asked to join the proposed category. Thus, from a user perspective, adding a category implies inviting another user to a common group.

In an implementation, the determination of common groups operates with an exchange of Hello messages. Each time the list of categories $C_{uv}$ that the node $u$ specifies for a contact $v$ is modified, the contact $v$ is marked unverified in $u$'s application. For each unverified contact $v$, the application of $u$ periodically tries to send a Hello message as long as the contact remains unverified. The Hello message contains the list of categories $C_{uv}$ proposed by $u$. The receiving node $v$ replies with a Hello Response message, including its corresponding list of categories $C_{vu}$ proposed for contact $u$. When either a Hello or Hello Response message is received, the categories $C_e = C_{uv} \cap C_{vu}$ common to both parties are determined and the contact is marked as verified.

For groups that are not common to both users, the other user is invited. Thus, for each group $c$ in $C_{uv} \setminus C_{vu.}$ the application displays the group name and offers user $v$ to join the group. If user $v$ accepts the invitation, the category $c$ is added to $C_{vu}$ and the contact is marked unverified in $v$'s application, thereby triggering another Hello message.

Removal of a category $c$ from $C_{vu}$ requires no special considerations: the contact $u$ is marked unverified and the set of common groups is updated on the triggered Hello exchange.

#### Access control and policies

The purpose of groups is to provide access control. The access to a shared resource can be limited to users of a given group $c$. For example, a work-related file might only be available to colleagues and a photo only to

family members. The resource shared by user $u$ is then available to a user $v$ only if $dist_c(u, v)$ is finite.

Based on our studies in Section 4.4, we assume that the willingness to share a resource declines as the social distance between two users increases. For example, a user may want to distribute photos to her friend, but is less willing to distribute them to a friend's friend. We also assume that the willingness to access a resource declines as the social distance increases. For example, a photo taken by a user at an event may be interesting to the user's friends, as they know many people in common. It may also be interesting to the friend's friends as they might know some common people and the event and the location probably are known to them. For people more remotely connected, the interest may be marginal.

To account for this, we introduce two policies, called the *distribution horizon* and the *interest horizon*. A resource shared by user $u$ with a distribution horizon $dist_{dh}$ is only accessible to a user $v$ if $dist_c(u, v) \leq dist_{dh}$. Likewise, a search by user $u$ having an interest horizon of $dist_{ih}$ only locates the resources of a user $v$ if $dist_c(u, v) \leq dist_{ih}$. Both horizons limit the maximum distance between the resource provider and the resource requester, but each horizon is defined by a different user. Combining both horizons, a resource by user $u$ shared to group $c$ is accessible to user $v$ if $dist_c(u, v) \leq \min(dist_{dh}, dist_{ih})$.

We also propose a policy based on the *minimal connectivity* of a user to the group. A resource shared with minimal connectivity $d_{min}$ to group $c$ is only available to a user $u$ if $d_c(u) \geq d_{min}$. For example, when the minimal connectivity is two, a resource is visible only to group members with at least two connections to other group members. In this case, it is not sufficient to be accepted to the group by only a single member. In practice, this implements a simple form of collaborative group management.

Policies control sharing and searching. Each resource $r$ is marked with the sharing policy $P_r = (G_r, dist_{dh,r}, d_{min,r})$ including the set of groups $G_r$ to which they are shared, the distribution horizon $dist_{dh,r}$ and the minimal connectivity $d_{min,r}$. In practice, it is unlikely that $dist_{dh,r}$ and $d_{min,r}$ are defined on a per-resource basis. The search $s$ may be limited to resources in a given set of groups $G_s$ and within an interest horizon $dist_{ih,s}$.

### 4.6.4 Implementing resource discovery in a mobile social network

In a social network, the topology is relatively static. A user is expected to add and modify contacts at a rate of a few modifications per week or month. The user's devices, however, may show a high churn rate. Verkasalo [Ver07] reports an average of 1.2 power-off switches per day. Despite the churn, the device is assumed to be online most of the time, and offline periods are of a more temporary nature. We propose the rather radical and unusual approach to consider the node as part of the topology even though the device is offline. The implementation must therefore consider the existence of such offline nodes.

The topology of a social network is typically a random power-law graph [WS98]. Replacing the topology with a structured or loosely

structured network, or forming another layer of overlay on top of the social network, would break the connection between the real network and the implemented one. It would cause signaling between nodes that are not connected in the social network, which would cause costs (monetary or non-monetary) that are unrelated to the user's contacts. Furthermore, maintenance of a structured or loosely structured topology is difficult, as the churn may be high and the overhead of maintenance should be kept low. Therefore, we build the technical solution directly on top of the topology of the social network.

In a random topology, distribution is typically performed with flooding or random walks. We exclude random walks because of long delay and non-determinism. Adapting a distribution method for the given scenario implies addressing the following requirements:

1. The distribution must be limited to group members only.
2. The system must support implementation of policies.
3. Offline nodes in the topology must be supported.

### Implementation based on flooding

Flooding can easily be adapted to observe groups. Both index updates and search requests can be distributed using flooding. Thus, we can use flooding to build either a fully proactive or a fully reactive solution. The message contains a distribution list $D$ specifying the groups to which it will be delivered. In search distribution, $D = G_s$ is the groups examined in a search $s$ and in index distribution $D = G_r$ is the groups to which the resource $r$ is distributed. In the flooding process, the message is forwarded on a link $e$ with the groups $C_e$ activated only if $D \cap C_e \neq \varnothing$. When the message is forwarded on the connection $e$, the distribution list is updated: $D \leftarrow D \cap C_e$. The algorithm limits the distribution to members in groups common to the sending node $u$, i.e. to the set of nodes $\{v \mid dist_c(u, v) < \infty, c \in D\}$.

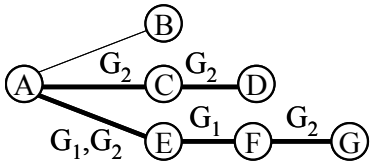


**Figure 4.17. Flooding to group members.**

To illustrate flooding with groups, let us look at the example in Figure 4.17. Nodes A, E, and F form the group $G_1$. Nodes A, C, D, and E form the group $G_2$. Nodes F and G form a different group named $G_2$. Node A publishes a resource $r$ with $G_r = \{G_1, G_2\}$. The group list in the entry sent to E contains $D_{AE} = D_A \cap C_{AE} = \{G_1, G_2\} \cap \{G_1, G_2\} = \{G_1, G_2\}$. The index entry sent to C contains $D_{AC} = D_A \cap C_{AC} = \{G_1, G_2\} \cap \{G_2\} = \{G_2\}$. The entry is not forwarded to B because $D_{AB} = \{G_1, G_2\} \cap \varnothing = \varnothing$. Node C forwards the entry to node D: $D_{CD} = D_{AC} \cap C_{CD} = \{G_2\} \cap \{G_2\} = \{G_2\}$. When E forwards the entry to node F, group $G_2$ is removed: $D_{EF} = D_{AE} \cap C_{EF} = \{G_1, G_2\} \cap \{G_1\} = \{G_1\}$. Node F does not forward the entry to node G: $D_{FG} = D_{EF} \cap C_{FG} = \{G_1\} \cap \{G_2\} = \varnothing$. Note

that the solution provides read-write access control as A cannot publish resources to a group that A is not a member of, and other nodes cannot locate resources in groups they are not members of.

The distribution horizon and index horizon policies are implemented by counting remaining hops and forwarded hops. The minimal connectivity policy cannot be implemented for flooding.

While requirements 1 and 2 are reliably handled, requirement 3 is challenging. The number of redundant links must be sufficiently high to allow offline nodes to be bypassed. In practice, this requires a high connectivity and, thus, a high average degree for all nodes in the group. If flooding is used for index updates, the periodical index update frequency $f_{i,refresh}$ must be sufficiently high to enable later updates to nodes currently offline or unreachable. This increases the overhead. While a high average degree is required for reliable operation in a topology based on a social network, it is at the same time the main restriction. Recall that the overhead of flooding is proportional to the average degree.

### Implementation based on the Direct Index algorithm

As an alternative to flooding, we extend the Direct Index algorithm defined in Section 3.6 to limit distribution to group members only and to provide access control. The extended algorithm is published as part of [Bei07b]. The modification requires adding group membership information to the peer table and the messages. Recall, that the peer table of a node $v$ has an entry $p_w = (N_w, E_w, T_w)$ for each known peer $w$, where the neighbor list $N_w = \{u_1, u_2, \ldots, u_n\}$ indicates the neighbors of node $w$. Now for each neighbor $u$ we store a list of groups common between $v$ and $u$. Furthermore, in the Update message $(N_v, E_v, F)$, the neighbor list $N_v$ is extended with a list of groups for each neighbor.

In order to provide access control, a routing table must be calculated based on the neighbor table. A separate routing table $R_{u,c}$ is calculated for each group $c$ known by node $u$. The routing table entry $R_{u,c}(v)$ indicates the distance $dist_c(u, v)$ in group $c$. More importantly, it indicates whether nodes $u$ and $v$ are in a common group $c$. The routing table $R_{u,c}$ is recalculated (e.g. using Dijkstra's algorithm) when any neighbor information for a group $c$ is modified.

Before node $u$ sends an Update message to node $v$, it checks the routing table. The message is sent only if $R_{u,c}(v) < \infty$ for any group $c$. Otherwise, the entry $p_v$ for node $v$ is removed from the peer table. The update message sent from $u$ to $v$ includes only the resources for which $R_{u,c}(v) < \infty$ for any $c \in D$, where $D$ is the distribution list of the resource. Thus, before granting access to a resource, the node checks that there is a path of connections with a common group activated. When an Update message is received, it is processed only if $R_{u,c}(v) < \infty$ for any group $c$.

The distribution horizon and interest horizon policies are implemented using the routing table. An Update message sent by $u$ to $v$ only includes resources for which the distance $R_{u,c}(v) \leq dist_{dh}$ for any $c \in D$. When node $u$ receives an Update message from $v$, it only stores the index of resources for which the distance $R_{u,c}(v) \leq dist_{ih}$ for the group $c$. The minimal connectivity policy is also implemented using the neighbor

table. An Update message sent from $u$ to $v$ includes only resources distributed to group $c$ for which $|W| \geq d_{min}$, where $W = \{w \mid v \in N_w\}$. The algorithms for handling timer expirations and sending updates are summarized in Figure 4.18 and Figure 4.19, respectively. The modified algorithms for handling received messages are summarized in Figure 4.20 and Figure 4.21.

```
1:  on expiration of timer T_w:
2:  SendUpdate(w)
```

**Figure 4.18. Pseudo-code for node $v$ when timer $T_w$ expires.**

```
3:  SendUpdate(w):
4:  C_vw ← { c | R_u,c(w) ≠ ∞ }
5:  if C_vw = ∅ then
6:     remove entry P_w
7:  else
8:     E_v ← { r ∈ ℒ_v | G_r ∩ C_vw ≠ ∅ }
9:     N_v ← { (v, N_m ∩ C_vw) | N_m ∩ C_vw ≠ ∅ }
10:    send update (N_v, E_v, 1) to w
11:    reschedule T_w with an exponential backoff
12: end if
```

**Figure 4.19. Pseudo-code for node $v$ sending an update to node $w$.**

```
13: on received update (N_w, E_w, F) from w:
14: if R_u,c(w) = ∞ ∀ c then
15:    send error message to w
16: else
17:    P_w ← (N_w, E_w, 0)
18:    for each (u, G_u) in N_w do
19:       if P_u is undefined then
20:          P_u ← (∅, ∅, 0)
21:          reschedule T_u after 0… UpdateInterval seconds
22:       end if
23:    end for
24:    if F = 1 then
25:       SendUpdate(w)
26:    else
27:       reschedule T_w after UpdateInterval seconds
28:       send acknowledgement to w
29:    end if
30:    for each known category c do
31:       generate R_u,c(w) to all w in N_w using Dijkstra's shortest-path-first
32:    end for
33: end if
```

**Figure 4.20. Pseudo-code for node $v$ handling an update received from node $w$.**

```
34: on received acknowledgement from w:
35: reschedule T_w after UpdateInterval seconds
```

**Figure 4.21. Pseudo-code for $v$ handling an acknowledgement received from $w$.**

The Direct Index algorithm is not required to perform separate actions in order to ensure that updates are delivered to currently offline nodes. If a node is offline, it will not reply to an Update message, and a new update will be rescheduled with an exponential backoff. The update exchange is performed normally when the node is again online later. As extra protection and to handle possible error conditions, a slow periodical update interval $f_{i,refresh}$ in the order of one per several days can be used. Neighbors are included in the neighbor lists of the Update messages even though they are offline; thus, the topology represents the entire social network. Only if a node does not reply in an excessive time (in the order of several days) it is removed from the neighbor table.

#### Considerations regarding the proactive architectures

We propose, as an optimization, that nodes store the indices of other group members between active sessions. This is possible because of the stable topology. The index of all other nodes is already available at startup, and the update traffic can be reduced. The consequence is that resources of members currently off-line seem to be available, and therefore the actual availability of the resource must be checked separately. Checking is, however, anyway required if the index is compressed by a Bloom filter because of potential false positives. Checking has a low overhead involving a single roundtrip and can be combined with obtaining a detailed description of the matching resources.

The access control of the resource discovery system only hides resources that should not be accessed. Full access control can be obtained by including a random number in the resource descriptor. This number must be included in the resource access signaling. Access is granted only if the number in the access signaling matches with the one in the original resource descriptor.

### 4.6.5 Feasibility of proactive architectures

We can take either a proactive or a reactive approach for the group-based mobile resource discovery system. Using an appropriate signaling protocol, such as SIP, direct messaging between users is possible at the application layer, which enables the use of temporary links required for Direct Index. With the help of Figure 3.35, we can decide whether a proactive solution meets our needs. We assume that a hybrid proactive-reactive solution cannot be used because we need to follow the topology of the social network. In this situation, Equation (2.40) states that a proactive solution is optimal when $r > \Omega_i / \Omega_s$.

We first examine the ratio between $\Omega_i$ and $\Omega_s$. As the alternative is reactive flooding, we use the value $\Omega_s \approx D$ - 1, determined experimentally in Section 2.4.5. The first considered proactive approach uses flooding to distribute the index information. This gives $\Omega_i \approx D$ - 1 for index updates. If the resource is distributed to all contacts of a user, the degree is high. Based on Survey 1 described in Section 4.4.3, the approximated median number of contacts in the phone books is $D$ = 113, while Verkasalo [Ver07] reports a median value of $D$ = 111 contacts. When groups are used, only a part of the user's contacts belong to a given group. We

therefore assume an average degree in the order of $D = 10$. Because of the high degree, flooding is highly inefficient in social networks, both for search and index distribution. Instead, we propose using the Direct Index algorithm described in Section 3.6. This has a low overhead ($\Omega_i \approx 3$ in simulations) that is independent of the degree. Using $\Omega_i = 3$ and $\Omega_s = 10$, we can conclude that Direct Index is optimal when $r > 0.3$ in our scenario.

The search/index ratio is defined as $r = f_s / f_i$. The search frequency $f_s$ is entirely determined by the users and cannot be affected by the design of the system. However, the index update frequency $f_i$ can be adjusted for the given scenario. In Section 3.1, we divided index updates into components: $f_i = f_{i,entry} + f_{i,exit} + f_{i,modification} + f_{i,refresh}$. By saving the remote index between active sessions, we can eliminate $f_{i,entry}$ and $f_{i,exit.}$. The modification frequency $f_{i,modification}$ completely depends on the user. The periodical update frequency $f_{i,refresh}$ depends on the index distribution algorithm. Index flooding requires a periodical update frequency comparable to the churn frequency since it has to be frequent enough to allow nodes to receive updates that they missed while being offline. Direct Index requires a periodical update frequency comparable to the frequency of changes in the social network. Because Direct Index detects offline nodes and changes in the topology, the periodical refresh rate is merely an extra protection. Based on this reasoning we can assume that $r > 1$ for a wide range of applications. Clearly, proactive solutions should be considered for social networks.

### 4.6.6 Index compression

The proactive operation gives quick searches, but a major challenge is the index update traffic and the amount of index information to be stored at each node. A node must, for each group, store the index of all members within the interest horizon. We therefore propose utilizing Bloom filters to compress the index information. The material in this section is published as a part of [Bei07b].

#### Traffic estimate

We dimension the Bloom filter to represent 100 resources on average, each resource described using 7 elements (keywords) on average. Thus $n = 700$ elements are stored per node. Nodes with more than 100 resources need to create several index entries to maintain a sufficiently low probability of false positives. Because of the low cost of a false positive, we allow false positives at a probability of $p \leq 0.02$. Using (3.25) we can calculate the optimal number of hash functions $k = 6$. With (3.26) we calculate the optimal number of bits $v = 6059$. Adding a header of about 20 bytes (in binary format) to the entry, the size of an index entry is $S_i = 777$ bytes.

For an average group size of $N = 50$ nodes, the index takes about $NS_i = 38$ kB of storage space per group. Such a group contains up to 5000 resources. The requirement is small compared to the memory sizes that are common for modern phones.

Transporting index updates over SIP gives the advantage of good interoperability with the IMS and easier NAT traversal. The disadvantage is the requirement of a centralized SIP server. A textual representation of the Bloom filter using Base64 encoding [Jos06] adds about 40% overhead, increasing the size of the index entry to $S_{i,Base64}$ = 1060 bytes. Adding the header of a SIP MESSAGE request (approximately 330 bytes), a header of the index entry (approximately 50 bytes), and the UDP and IP headers (28 bytes), the size of the index update datagram is $S_{i,IP}$ = 1468 bytes. The index update is sufficiently small for transport in a single UPD datagram.

With the Direct Index algorithm, a node receives an index update only once per update round. A complete update of all nodes in the group requires $NS_i$ = 72 kB, corresponding to an average bandwidth of 6.8 bit/s per group if a complete update is done daily. In practice updates are expected to be done less frequently. With flooded index updates in a topology with average degree of $D$ = 10, a complete update of all nodes requires about $DNS_i$ = 720 kB. This corresponds to a bandwidth of 68 bit/s per group when updates are performed daily.

### Searching with compressed filters

To locate a resource, a Bloom filter called a search filter is used. The search filter is generated by applying the hash functions to the keywords of the query. The search filter is compared with each Bloom filter in the index. If the query matches, i.e. for every bit in the search filter the corresponding bit in the Bloom filter is set, a message containing the original query is sent to the responsible node, which performs a local check. The query contains the original list of query strings. Because of false positives, the node may receive a query that does not match with any resource. It then responds with an empty result list. The search results are presented to the user according to increasing social distance, allowing the user to quickly locate the most relevant sources.

The disadvantage of Bloom filters is that the information lost in compression makes complex queries impossible. However, since the whole Bloom filter is available at a single node, combinations of attributes can be performed using logical operations. The logical AND operation, meaning that all keywords must match, is implemented as a binary OR operation between the search filters of both keywords. For example, if the search filter of “golden” contains the bits (12, 15) and the search filter of “gate” contains the bits (6, 15), then the query “golden” AND “gate” contains the bits (6, 12, 15). The logical OR operation requires specifying the query as a set of search filters. The query matches if any of these search filters matches with the Bloom filter. For a query “golden” OR “gate”, the query contains two separate bit-fields: (12, 15) and (6, 15). Nodes matching with either of these bit-fields are examined. In a query combining different logical operators, such as “golden” AND (“gate” OR “bridge”), the distributivity property [RW95] can be utilized to rewrite the query. In this case the query is rewritten as (“golden” AND “gate”) OR (“golden” AND “bridge”), which can be specified with two bit-vectors: (6, 12, 15) and (3, 12, 15, 19), assuming that “bridge” represents the bits (3, 19).

Other types of complex queries must be forwarded to all nodes in the group, whereas every node performs the matching locally. In a proactive architecture, the node is aware of all nodes belonging to the same groups. Therefore, the node can send its query directly to these nodes, which saves bandwidth compared to flooding. To further save bandwidth, the resource requester can send the queries sequentially with increasing distance until the resource is found. This method increases the search delay, but allows the query to be aborted once a required number of resources are found, which is not possible with traditional flooding.

## 4.7 SIP signaling schemes for resource discovery

We can separate between four generic situations where control messages are sent in today's peer-to-peer systems:

1. Publishing the shared resource (index distribution).
2. Locating the shared resource (search distribution).
3. Initiating and controlling access to the shared resource.
4. Maintaining the overlay structure.

Index distribution and search distribution are one-to-many signaling in most architectures. Control of resource access is one-to-one communication even though a resource can be accessed from several locations simultaneously. The actual resource access (e.g. the content transfer) is classified as user traffic and not as control traffic. Messaging related to overlay maintenance is specific to the overlay structure, and has a minor role in unstructured or loosely structured systems.

All commonly used file sharing systems use proprietary protocols for index distribution and search distribution. For several peer-to-peer systems there are several compatible implementations available and these protocols have become more or less de facto standards. No considerable efforts on interoperation between different systems have been made. However, some applications can be simultaneously connected to multiple resource discovery systems. The file transfer is usually implemented with the Hypertext Transfer Protocol (HTTP). The File Transfer Protocol (FTP), which at first sight would seem the natural choice, has not been used in any file sharing systems. HTTP and FTP combine the access-related signaling and the actual content transfer in the same protocol.

### 4.7.1 Resource discovery with SIP

The Session Initiation Protocol (SIP) [RSC+02] is becoming the standard protocol for initiating and controlling calls in IP telephony. SIP is also chosen as the signaling protocol for the IP Multimedia Subsystem (IMS) [PMK+04]. SIP implements application-layer routing using proxies, which can forward, redirect and fork (forward to multiple recipients) a call. Several extensions have been defined, allowing SIP to be used for advanced session control and value-added services including presence services [Ros04] and instant messaging [CRS+02]. For searching, SIP relies both on its own location servers and on the Domain Name System (DNS). The user registers its current location through the registrar to the location server. The location server typically serves all users of a given

domain. Upon receiving an incoming call, a proxy retrieves the destination's current location from the location server. Before that, the proxy has been located by looking up the domain name of the user in the DNS. Like in peer-to-peer systems, the access (media transfer) takes place directly between endpoints once the destination is found. SIP provides signaling for controlling the access (i.e. the call) but the content transfer is performed with a separate protocol (e.g. the Real Time Protocol).

Apart from the help from DNS, SIP can be seen as a centralized resource discovery system, where the resource is the user. The REGISTER message performs the index update and the INVITE performs the search. The use of SIP in generic resource discovery is a relatively new idea. A conceptual SIP-based peer-to-peer application, called SIPShare, has been presented by Earthlink [SIPshare]. The application is based on a search flooding architecture and demonstrates the use of SIP for peer-to-peer signaling. It is, however, not designed with mobile networks in mind.

A different approach for combining SIP with peer-to-peer technology is P2P-SIP [P2PSIP]. In P2P-SIP, the server-based architecture of a SIP network is replaced by a structured overlay network. P2P-SIP thus borrows ideas from Skype [Skype] but bases the signaling on the open SIP protocol. The purpose is, like in standard SIP, to locate users by mapping the URI of a user to the user's current IP address. Therefore, a simple exact-match mapping is adequate. The P2P-SIP approach is different from the topic of this work, where we use extended SIP to support complex searches and to locate more generic types of resources.

Using SIP signaling for generic resource discovery becomes especially interesting in IMS-networks. We identify the following advantages of the approach:

1. SIP provides an established method for user-to-user signaling. As SIP is an integral part of the IMS, SIP signaling messages are routed correctly between mobile terminals, whereas firewalls and NATs would complicate the use of proprietary signaling.
2. SIP supports connectionless signaling, which is especially important in architectures using temporary overlay links, e.g. the Direct Index architecture.
3. SIP allows re-using the IMS infrastructure, avoiding the need for separate charging, security and management mechanisms for the peer-to-peer service.

The major drawback of using SIP in a mobile environment is the overhead of text-based messages. This problem can be, to a certain degree, solved using compression, e.g. using SigComp [PBC+03].

In this section, we examine how SIP can be used as a generic resource discovery system. To accomplish this goal, we must

1. enhance the SIP signaling to support generic types of resources, and
2. replace the use of DNS in forwarding between proxies with a peer-to-peer system.

We take two different approaches to designing a SIP-based signaling scheme, as presented in the following subsections. The following requirements are set as the basis for the design:

1. The signaling must maintain compatibility with both SIP and with IMS.
2. The signaling must be applicable to different peer-to-peer architectures, including centralized, semi-centralized and distributed architectures.
3. It must be possible to obtain search results incrementally as matches are found and it must be possible to abort the search.
4. The state information required to be maintained by network elements must be minimized.

### 4.7.2 Signaling scheme based on INVITE

Our first signaling scheme is proposed in [BML+05] and more comprehensively published in [MBL+06b]. To maintain the similarities to the operation of conventional SIP, we propose using an INVITE request to locate and initiate access to resources. A generic resource is located in a similar manner as a user is located in SIP. The main difference is that a user is unique, even though the user may have several active terminals, while there may be several resources matching to a generic search. The searching user must be presented with the alternatives and allowed to select one or several resources to access. Allowing the user to choose the destination from several alternatives is uncommon in normal SIP, but is possible to implement with the "303 Multiple choices" response. Another difference is that general resource discovery requires the ability to use several attributes in the search. Normal SIP specifies the destination in the "To" field. While a single line is enough in SIP to describe a user, the description of a generic resource involves several attributes. We therefore prefer transporting the query in the body. In our architecture, the INVITE request carries a body in XML format defining the search criteria.

SIP uses a central location server to store the locations of the users. Typically there is one server per domain. The user updates its location by sending a REGISTER request. A similar server can be used for generic resources, acting as an index. In our target implementation, index distribution is performed by sending a REGISTER request containing a body with the resource descriptor in XML format. A resource is removed by unregistering it.

To search for a given resource, an INVITE request is sent to the index node, which performs a local search. The replies are transmitted in a "303 Multiple choices" redirection responses or in the 1xx group of provisional responses. Provisional responses are useful as they allow the list of results to be complemented incrementally, while a redirection (3xx) response can only be sent once. The "200 OK" response cannot be used as it establishes a session with the super-node, which is not desired.

The signaling scheme is presented in Figure 4.22. The user agent (UA) binds its URI to a given IP and port number by registering with the registrar, which is implemented in the same device as the proxy. Then the

UA registers its shared resources to the P2P-AS. Searching is performed with the INVITE message to which the P2P-AS replies by giving the descriptions and addresses of matching resources. The user selects one of these resources to access, whereas the UA sends an INVITE directly to the address obtained from the search. The resource access is performed with a protocol not related to SIP, for example, with HTTP.

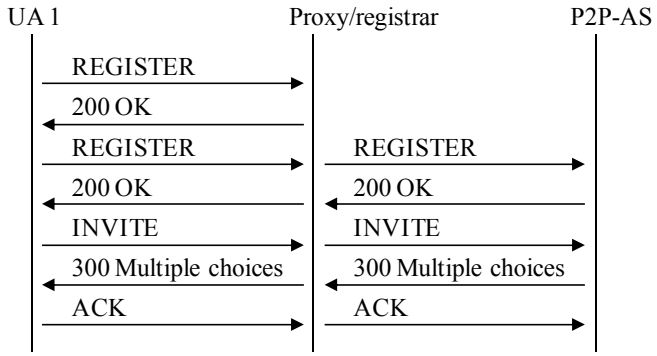


**Figure 4.22. Signaling in target scheme.**

We define a compact XML-based scheme for describing resources. The XML body includes commands for incrementally adding, removing and updating resource descriptions in the index. A similar XML body is used to describe the query in the INVITE request. The search results are in an XML format in the body of the response.

Implementing architectures other than centralized ones require the forwarding of search requests. Normally SIP relies on DNS to locate the destination proxy of a session, whereas the proxy is indicated by the domain name in the user's address. For general resource discovery no such information is available. The INVITE request must be forwarded on all search links specified by the overlay topology. Consequently, each indexing node must know the neighboring indexing nodes.

The SIP standard allows an INVITE request to be forwarded and forked to several destinations. This makes flooding possible to implement. Forwarding can be performed in a stateful or stateless manner. It is essential for the correct operation of flooding to avoid loops and to detect multiple receptions of the same message. The Via field in the INVITE is used to detect loops. If a node receives a request with its own address in the Via field, the request is dropped. The Via field consequently works like a trace. As stated earlier, traces avoid loops but do not avoid multiple forwarding. A more reliable method is to use state information, whereas the node remembers the Call-ID and CSeq headers of the forwarded messages. The Call-ID specifies a unique identify of the session and the CSeq indicates the request within the session. If a node receives a message with a Call-ID and CSeq combination seen before, it does not forward the message. Instead it replies with a "482 Loop Detected" response.

Iterative forwarding can be implemented using redirection responses. Each indexing node indicates a set of alternative indexing nodes to the

resource requester, which can choose whether to continue the search by repeating the INVITE to the indicated nodes. An iterative approach is not suitable to be performed by the mobile device as it creates excessive traffic on the wireless interface.

Architectures, such as global indices and Zone Indexing, that involve index distribution to multiple nodes require the REGISTER messages to be forwarded. REGISTER messages must be forwarded to all neighboring nodes to which an index link exists. SIP does not natively support forwarding of a registration. A registration can be forwarded recursively using the loop detection methods used for the INVITE request. However, this approach may meet practical difficulties as is deviates from standard operation significantly.

### 4.7.3 Signaling scheme based on SUBSCRIBE/NOTIFY

An INVITE request is intended to resolve the location of a destination and to set up a session with this location. Our experiments with signaling showed that for some applications, including resource discovery, there is a need to query the location of a destination without establishing a session with it. The work on this issue lead us to the proposal of a new mechanism for generic resource searching using SIP that was given as input for standardization [GMB+06]. The approach has also been presented in [MGB+07]. The signaling scheme is based on a new "resource" event package [GM06] allowing a terminal to subscribe to resource information. Similarly to SIPshare [SIPshare] the scheme is based on the PUBLISH and SUBSCRIBE requests, but utilizes existing standardized components.

We extend the use of the PUBLISH request [Roa02] to publish information about a generic shared resource. The resource is described in XML format in a *resource document* included in the body. The resource document is defined by a *resource event package* [GM06]. The P2P-AS replies with a "200 OK" response including a SIP-ETag field providing an identifier for the published resources. A publication must be refreshed periodically. To refer to the existing publications, these renewals include the identifier of the resource in the SIP-If-Match field. The resource can later be modified or deleted by resending a PUBLISH request referring to the identifier but with a new version number in the resource document.

Searching is implemented by sending a SUBSCRIBE request for the resource event package. The SUBSCRIBE request includes a filter body [KLL+06] that defines the query. The P2P-AS replies with a "200 OK" response. Immediately after sending the reply, the P2P-AS reports potential matching resources using a NOTIFY request. The body contains a resource event package describing the matches, if any. The SUBSCRIBE request creates a soft state in the P2P-AS for the duration indicated in the SUBSCRIBE request. The P2P-AS may later, as it learns about more resources, send further NOTIFY requests. Each NOTIFY is answered with a "200 OK" response. When several matching results are found, the results can be divided between several NOTIFY requests, which can be spaced in time to allow the user to receive a gradually growing list of matches. The subscription state is removed by sending a SUBSCRIBE with a zero value in the Expires header field. If, instead, the

subscription state times out, the P2P-AS sends a NOTIFY indicating "terminated" in the Subscription-State header field. The signaling is illustrated using an example in Figure 4.23.

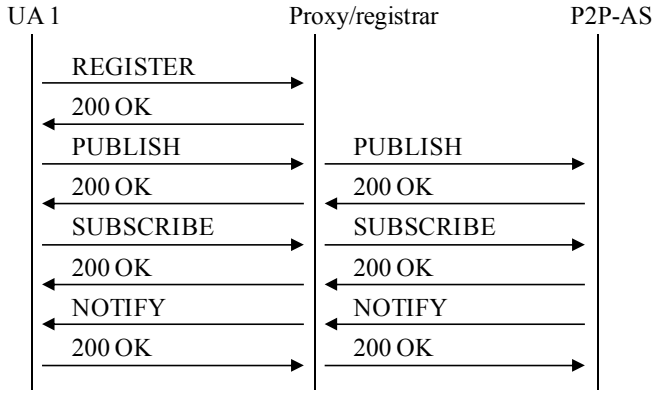


**Figure 4.23. Signaling using the resource event package.**

A P2P-AS can forward PUBLISH requests and SUBSCRIBE requests to other P2P-ASs. Loop detection and detection of duplicate receptions has not been considered in the published work. However, these can be addressed by an identifier tag unique to the transaction. The Call-ID could be used, but learning from our practical implementation (see Section 4.8.3) we suggest defining a new field for this specific purpose.

Compared to the scheme based on INVITEs, this signaling scheme represents a cleaner approach utilizing existing standardized components. However, the large signaling overhead due to the inefficiency of XML and the generality of the event packages is disadvantageous in a mobile network. Thus, the cost of generality is a high overhead.

### 4.7.4 Signaling in resource access

The signaling in the resource access setup is identical to both the INVITE and the SUBSCRIBE/NOTIFY schemes. At this stage, the resource requester has obtained the public SIP URI of the resource provider from the resource descriptor. As depicted in Figure 4.24, the resource requester sends an invite to the resource provider to establish the resource access session. Using an INVITE request retains the compatibility as it is the same message as used in normal voice sessions in SIP. The INVITE is sent directly from the requester to the provider without traversing the P2P-AS. The body transports a Session Description Protocol (SDP) [HJ98] offer, which describes the session in terms of capabilities, addresses, ports, communication protocols and other parameters. The SDP can be adapted for generic access protocols, e.g. for file transfers, whereas the identifier (e.g. file name or hash) of the resource is included. The resource provider can queue the access if it currently is serving the maximum number of other users. It then replies with a provisional response "128 Queued". When the provider is ready to provide access to the resource, a "200 OK" response containing a SDP body is sent to the requester. The session is terminated by sending a BYE request.

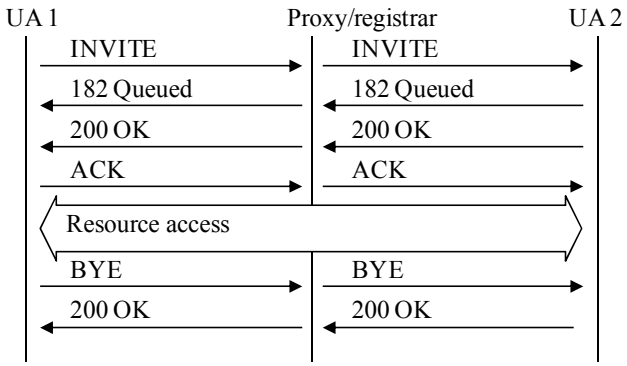


**Figure 4.24. Signaling in resource access.**

Each type of resource requires a different access method and a different set of parameters in the SDP message to describe the access. For file transfers, the Message Session Relay Protocol (MSRP) is a more suitable candidate than HTTP and FTP. MSRP [CMJ07] is a protocol intended for the transport of messages and files within a SIP session.

Communication between two mobile nodes is difficult in many networks including most cellular networks because of Network Address Translators (NATs) and firewalls. A NAT is more restrictive for the resource access than for signaling as the resource access is performed directly between endpoints whereas signaling traverses the SIP proxy. If only one endpoint is behind a NAT, this endpoint must be the initiator of the session to the other endpoint. If both endpoints are behind a NAT, the access must be assisted with a relay, either in the form of a separate server in the network or as an additional function provided by an endpoint that is not behind a NAT. A standard method for relaying is provided by Traversal Using Relay NAT (TURN) [RMM08]. As relaying loads the relay server and the network, it should be used as the last resort. The Interactive Connectivity Establishment (ICE) method allows searching for the optimal transport for a given case. Relaying is supported by MSRP through extensions [JMR07].

## 4.8 Technical feasibility of peer-to-peer in cellular networks

We perform an experimental study in order to evaluate the technical feasibility of peer-to-peer systems in today's cellular networks and with today's terminals, and to find possible limitations. We want to determine numerical estimates to:

- The memory consumption of a simple resource discovery application running in the mobile phone.
- The message sizes of a SIP-based resource discovery protocol.
- The search delays in various network configurations.

Several factors can hinder, delay or add complexity to the deployment of peer-to-peer services in cellular networks. We therefore examine the validity of the following postulates:

- *Postulate 4.1:* A resource discovery application can be implemented with the APIs and the user interface available on today's mobile phones.
- *Postulate 4.2:* Today's mobile phones have enough memory and CPU power to run a resource discovery application smoothly.
- *Postulate 4.3:* A SIP based signaling protocol for resource discovery can be implemented and works with the available SIP stacks and proxies without modification to them.
- *Postulate 4.4:* The network bandwidths of today's networks are adequate for a hierarchical resource discovery architecture.
- *Postulate 4.5:* Signaling can bypass the NATs and firewalls of today's networks.
- *Postulate 4.6:* Access connections can bypass the NATs and firewalls of today's networks.

### 4.8.1 Prototypes

We examine the technical feasibility using three testbeds consisting of prototypes of various elements:

- *Testbed O1*. The network consists of the PartySIP SIP proxy/registrar [Partysip], a Client Application running on a Nokia 6680 phone, a Centralized Index Node, and a TCP Relay. The wireless connection is a commercial 3G/WCDMA network (Sonera). The SIP proxy/registrar implements a standard SIP network. It can also be seen as a simple model of a CSCF in an IMS network. A single centralized index node is used, whereas a centralized peer-to-peer architecture is implemented.
- *Testbed O2*. The network consists of the Repro SIP proxy/registrar [reSIP], a Client Application running on a Nokia 6680 phone, a Distributed Index Node, and a TCP Relay. The wireless connection is a commercial 3G/WCDMA network (Sonera). This testbed is identical to Testbed O1, except for the changed proxy and a new index node. The Distributed Index Node replaces the Centralized Index Node, whereas the architecture is semi-centralized.
- *Testbed O3*. The network consists of the Octopus IMS network [Octopus], a Client Application running on a Nokia 6680 phone, a Distributed Index Node, and a TCP Relay. This testbed is identical to Testbed O2 with the SIP proxy/registrar replaced by a complete IMS network. The IMS network (Octopus [Octopus]) is a commercial but experimental IMS network used by universities and companies to test and evaluate IMS applications.

The *Client Application* is an edge node implemented on the Nokia Series 60 Symbian platform. The language chosen for the implementation is C++ because of the more complete interfaces for mobile programming and the availability of an operational SIP stack. The application is divided into three separate processes: the Core module, the Transfer module and the Graphical User Interface (GUI). The core module consists of the Registrar and the Finder. The prototype supports a file sharing service, which is represented by the Transfer module. File transfer is implemented

by a simple TCP-based protocol that adds support required for the TCP Relay. The modular structure allows later extensions to support access to other types of resources. New services, such as streaming and chatting, can be added by introducing new modules that define the shared resource, the access protocol and the GUI. The structure further allows different parts to be run independently. For example, the GUI can be stopped while ongoing file transfers continue running and shared resources still are available. Inter-process communication between modules is implemented using the standard client-server architecture of Symbian. The details of the implementation are given in [BML+05], in [MBL+06b], in [MGB+07], and in [Leh08]. The implementation was presented in the demo session of CCNC 2007 [MBL+06a].

The *Centralized Index Node* is implemented in the Python language [Python]. It includes a simple custom-built SIP stack allowing better possibilities to extend the SIP signaling. XML bodies are parsed using MiniDOM [Minidom]. The centralized index node served as a first implementation of an index node in order to assist the terminal client implementation and provide guidelines and interface models for the distributed index node. The resource database is implemented with Python data structures.

The *Distributed Index Node* is able to communicate with other index nodes, enabling a distributed multi-operator platform for resource sharing between mobile users. This index node is implemented in C++ on the Linux platform. The reSIPprocate [reSIP] SIP stack is used for SIP signaling, the TinyXML parser [TinyXML] for handling XML data, and MySQL [MySQL] as database for storing the index of shared resources. The details of the implementation are given in [Rey07].

The *TCP Relay* is used to relay file transfer connections between two client applications in order to bypass NAT and firewall restrictions. It is an element in the fixed network that connects two incoming TCP connections together. To be able to pair connections, a header is sent before the data transmission. The header indicates the direction (upload or download) and the hash of the resource. The relay can handle several simultaneous connections and is able to detect timeouts.

### 4.8.2 Feasibility in mobile device

Using the Client Application, illustrated in Figure 4.25, we examine the feasibility of the mobile resource sharing service from the perspective of the mobile device. We examine whether the current mobile phones have the capacity for running peer-to-peer applications. As the application was implemented successfully, it serves as a proof of concept, showing that client applications for resource sharing services can successfully be created on today's devices. We observe that the processing power and battery consumption is adequate for normal use. From the implementation perspective, no serious limitations are observed. The client is successfully implemented using the current APIs and SIP stack. Some issues related to the SIP stack, however, require modification of the intended signaling scheme as described in Section 4.7. Furthermore, the user interface is a serious challenge in the mobile application. Given the

small screen size, it is difficult to present long lists of matching resources and comprehensive resource information.

The memory usage of the Client Application varies between 200 kB and 350 kB. Additionally, the SIP stack and the SIP profile manager consume 170 kB of memory. Thus, about half a megabyte of RAM is adequate for running a simple file sharing application. Support for other types of services and a more complex user interface increases the memory requirements. It is highly likely that the SIP stack will be an integral part of the operating system in IMS enabled phones.

Studies [YG02] on Gnutella reveal that a user on average shares 340 files and that a file on average is described with $S_i$ = 72 bytes of metadata. Thus, the index of a node is on average 24 kB. One should, however, recognize that the use in a mobile environment is probably different and that the number of files can be much higher in case of, for example, photo sharing. Files are stored on flash memory, which currently is available in sizes of several gigabytes at affordable prices. As it is likely that future phones are equipped with more memory, we do not consider memory consumption as a limiting factor. Instead, the availability of memory allows a phone to cache indices of other users, as in the Direct Index architecture.


**Figure 4.25. The Client Application.**

### 4.8.3 Feasibility of SIP signaling

Testbed O1 evaluates the fundamental feasibility of using SIP signaling and the feasibility to implement SIP signaling using the available SIP stack of the mobile phone. Testbed O2 further evaluates the possibility to forward and fork requests as well as detect loops. Testbed O3 verifies the compatibility with a real IMS network. In all testbeds, the index node is implemented as a user agent registered to a SIP proxy. This is one of the several optional ways to implement a SIP Application Server. The intention of this section is to implement the signaling scheme based on the INVITE request as described in Section 4.7.2.

The implementation of signaling in Testbed O1 showed the constraints of extending SIP signaling for generic resource discovery. The SIP stack of the mobile phone is mainly intended for normal voice calls. As the stack is given and cannot be modified, we needed to make

some compromises, resulting in an implemented signaling scheme that differs from the target scheme.

First, the REGISTER was replaced with a MESSAGE method in resource registration as the proxy did not allow registrations to be forwarded. The operational principle remains the same. The MESSAGE method is certainly not a perfect choice as it is intended to transport user traffic and not signaling. However, as we could not define a new method, we had to stick to the available ones. We agree that the REGISTER message is in the first place not the best method for index distribution, as it also may face problems in implementing architectures involving forwarding of index information.

Second, we noticed that the SIP stack does not support the "302 Multiple choices" response and a replacement response code had to be chosen. As the response must have a code higher than 300 in order not to establish a session, we chose the "602 Not Acceptable" for lack of a better message. The signaling is summarized in Figure 4.26.

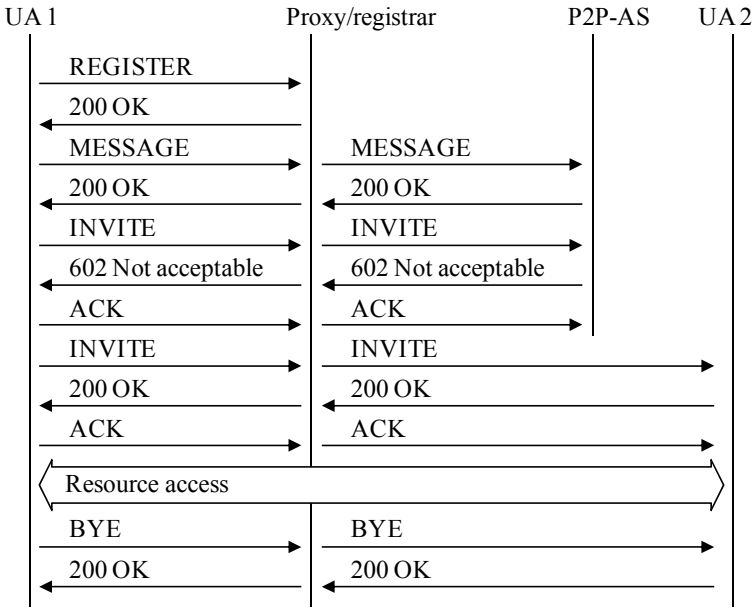


**Figure 4.26. Signaling in the implemented scheme.**

Testbed O2 revealed a disadvantage of implementing the index node as a user agent instead of a proxy server. In our signaling scheme the Call-ID header is used for loop detection and therefore all INVITE requests of the same resource discovery must have the same Call-ID. However, the used SIP stack considers that a call ends when a user agent is reached. When an INVITE is forwarded by a user agent, the forwarded INVITE must have a different Call-ID as it belongs to a separate call. Also, each forked INVITE must have a different Call-ID. This renders the use of Call-ID for loop detection impossible. To solve this problem, a custom header called Message-ID is introduced. The original Call-ID is copied to the Message-ID by the first indexing node after which it is repeated in all forwarded INVITE requests. Unfortunately, the PartySIP

proxy drops custom headers from forwarded messages. Therefore we replaced it with the repro proxy server.

Even though the signaling differs from the target scheme, the experiment shows that it is possible to implement a SIP based signaling scheme in a mobile scenario. An ideal signaling scheme requires standardized extensions that are supported by the SIP stacks and proxies. Furthermore, the P2P-AS should preferably be implemented as a proxy server instead of a user agent.

### 4.8.4 Network performance

So far in this work, we have measured traffic in terms of messages. This allows us to examine the behavior of an algorithm without considering a specific implementation, since the message size depends on how the used protocol encodes information. In this section we consider an implementation using SIP-based messages. From our prototype we obtain an estimate of the size of SIP messages, allowing us to estimate the required bandwidth in a SIP-based protocol. This determines the minimum required bandwidth on the wireless interface. The bandwidth further affects battery consumption in the terminal and the network traffic in the operator's network. In a pricing model based on transferred data, the message size also directly affects the cost of the service.

#### Centralized architecture

Using the centralized architecture of Testbed O1 we measure the message size and network delay. Sizes of SIP messages are extracted from packet traces using Etherreal (now Wireshark) and tcpdump. The results are presented in Table 4.4. The sizes are averages of multiple packets and rounded to the nearest ten-byte boundary. A typical search operation generates between 1120 and 2180 bytes, with a large variation due to the variable size of the search criteria, the metadata of the matching resources and the number of matching resources. A typical update operation generates between 700 and 1630 bytes, depending on the level of detail in the metadata and the number of advertised resources. Initiation of an access session generates 1220 bytes on average. We claim that the difference in the amount of bytes generated in different operations is relatively small, especially considering the variation in message size, and the error in using only a message count in algorithm evaluation is therefore low.

Likewise, our purpose has been to study delays independently of a specific implementation by measuring them as a hop count instead of as time. We now apply the delays obtained from actual wireless and wired networks. In [MBL+06b], we measured the delay of sending messages of varying sizes between the terminal and the index node in both directions. Applying these measured delays to a linear equation gives a delay of $(70 + 0.136b)$ ms for the uplink and $(100 + 0.155b)$ ms for the downlink where $b$ is the message size in bytes. The delay of the downlink is thus higher than the delay of the uplink even though the uplink has a lower bandwidth.

**Table 4.4. Sizes of SIP messages in prototype.**

| Action | Message | Type | Size (bytes) | Delay (ms) |
|---|---|---|---|---|
| Registration | REGISTER | Request | 370 | 120 |
| | 200 OK | Reply | 300 | 147 |
| Publication | MESSAGE | Request | 450…1380 | 131…258 |
| | 200 OK | Reply | 250 | 139 |
| Search | INVITE | Request | 430…480 | 128…135 |
| | 606 Not Acceptable | Reply | 370…1380 | 157…314 |
| | ACK | Request | 320 | 114 |
| Session setup | INVITE | Request | 540 | 143 |
| | 200 OK | Reply | 290 | 160 |
| | ACK | Request | 390 | 123 |
| De-registration | REGISTER | Request | 380 | 122 |
| | 200 OK | Reply | 250 | 139 |

Based on these equations, we calculate the delays for transmitting the messages between the application and the index node. Table 4.4 is complemented with these calculated delays, assuming requests are sent uplink and replies downlink. The delay of a search operation is thus between 399 and 563 ms, while the delay of a resource publication is between 270 and 397 ms. These assume a centralized architecture. The search delay is lower than the delay of the full search operation, as the ACK request is sent after receiving the results. The search delay is between 285 and 449 ms. According to our earlier user studies, all users are satisfied with such a delay. The initiation of an access session takes 426 ms for a single wireless link, but because the other terminal is assumed to be wireless as well, the total delay is 852 ms. Note that other delays, such as processing delay and queuing are not included in these measurements.

### Architectures with multiple core nodes

In Testbed O2 we replace the centralized architecture with four test topologies depicted in Figure 4.27. We select the topologies so that the results can be used to estimate the delays of more complex topologies. We measure the search load in an unloaded system: the time from sending the query to the reception of the response. The Distributed Index Node developed in [Rey07] is tested. The cumulative distribution of search delays for each of these topologies is presented in Figure 4.28. The figure shows the percentage of searches with a delay less or equal to the time at the x-axis.

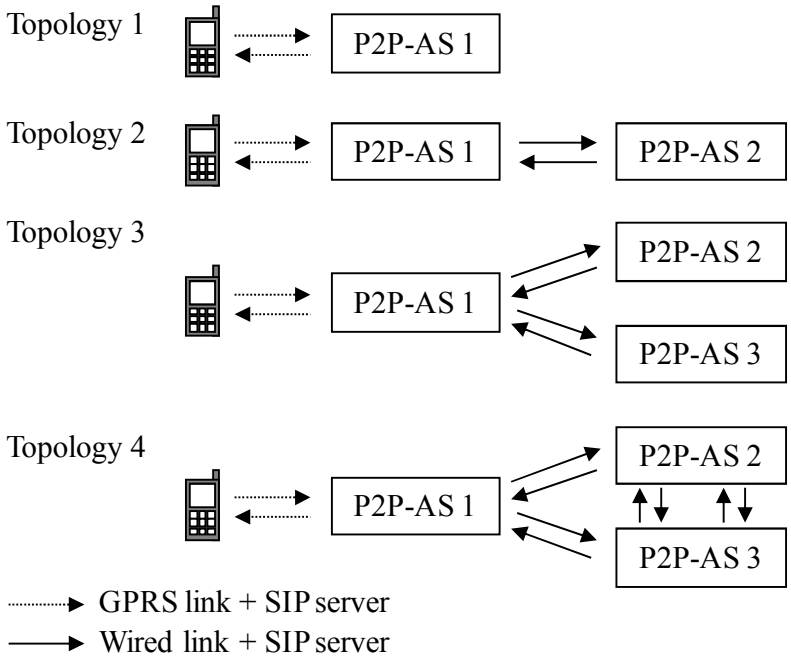


**Figure 4.27. Topologies in Testbed O1 for testing of search delays.**

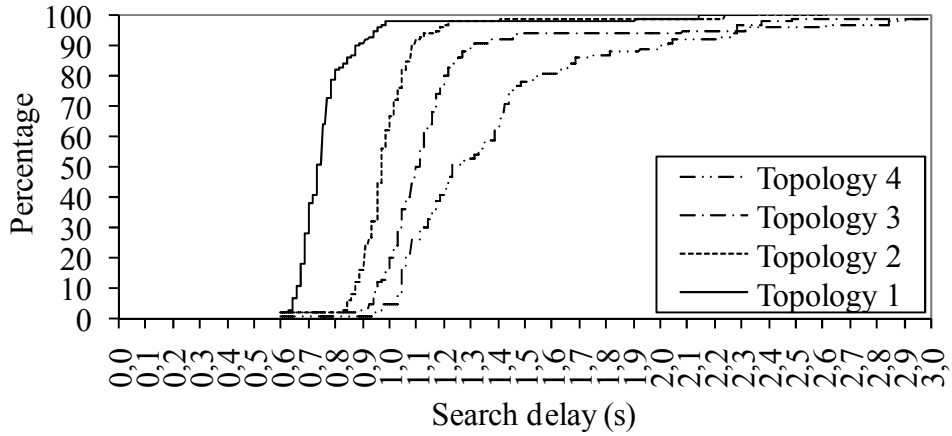


**Figure 4.28. Cumulative distribution of search delays in topologies 1 – 4.**

**Table 4.5. Search delays in topologies 1 - 4.**

| Action | Average delay (ms) | Median delay (ms) | Std.dev. (ms) |
|---|---|---|---|
| Topology 1 | $T_{t1}$ = 833 | 781 | 295 |
| Topology 2 | $T_{t2}$ = 1055 | 1000 | 194 |
| Topology 3 | $T_{t3}$ = 1140 | 1140 | 629 |
| Topology 4 | $T_{t4}$ = 1481 | 1421 | 616 |

Topology 1 represents a centralized architecture, in which the delay is caused by the air interface, the transport in the fixed network, the processing in the SIP proxy and the query handling in the index node. We

can thus express the delay as a sum $T_{t1} = T_{air1} + T_{proxy1} + T_{indexnode} + T_{proxy2} + T_{air2} + 4T_{fixed}$, where

- $T_{air1}$ is the delay of sending the request over the air interface,
- $T_{proxy1}$ is the delay of processing the request in the SIP proxy,
- $T_{indexnode}$ is the delay of processing the request in the index node, including the database lookup using MySQL,
- $T_{proxy2}$ is the delay of processing the reply in the SIP proxy,
- $T_{air2}$ is the delay of processing the reply over the air interface, and
- $T_{fixed}$ is the delay of sending a request or reply between two elements in the fixed network.

The delay $T_{indexnode}$ depends on the capacity of the index node and the database as well as the current load level of these. Likewise the capacity and load level of the proxy affect $T_{proxy1}$ and $T_{proxy2}$. The delay from the fixed network is small compared to the other delays.

The average, median and the standard deviation of the delays in the tested topologies are listed in Table 4.5. The 5% and 95% percentiles are 641 ms and 938 ms, respectively. Comparing them to the search delay obtained from Testbed O1 shows a difference of between 356 and 489 ms. This difference is likely to result from the processing in the index node, including a database lookup using MySQL. The messages and message formats used in both testbeds are the same.

In topology 2 the query is forwarded to another P2P-AS, adding the delay of the transport in the fixed network and the query handling in an intermediary super-peer. This increases the average delay with $T_{t2} - T_{t1} =$ 222 ms. For a chain of $n$ P2P-ASs, the median delay would be

$$T_{n-hop} = T_{t1} + (n-1)(T_{t2} - T_{t1}) \ . \qquad (4.1)$$

The results can be generalized to other architectures than the given semi-centralized architectures. Equation 4.1 gives the delay of an $n$-node zone in Zone Indexing, an $n$-node random walk, an IPIC network with $n$ clusters, and an $n$-node PSC cluster with ring topology, all provided that the reply is forwarded on the same path as the search request. Requiring the search delay to be below two seconds, which according to Section 4.4.5 satisfies all users, implies a maximum depth of 6 index nodes.

In topology 3, the results from two P2P-ASs must are combined and consequently the slowest of the branches determines the delay. Thus, even though the branches are traversed in parallel, each additional branch adds a delay. The difference in average delay to topology 2 is $T_{t3} - T_{t2} =$ 85 ms. In topologies with more branches, each branch could be estimated to add an identical average delay, but this implies an excessive simplification. In case the results are reported incrementally, the delay of the first reply is the delay of the fastest of the branches. Several one-hop branches are used in PIC and in fully connected architectures.

Topology 4 adds to topology 3 the additional query handling delay of a loop, which is detected and signaled. The difference in average delay between topology 4 and topology 3 is $T_{t4} - T_{t3} = 341$ ms. We can see that the use of parallel searches increases the standard deviation.

#### Resource access

The bandwidth of the access connection is mainly limited by the air interfaces. When the file is downloaded from another terminal with no other interfering transfers, the available bandwidth $B_{available}$ is determined as $B_{available} = \min(B_{uplink}, B_{downlink})$ by the bandwidth $B_{downlink}$ of the local downlink and the bandwidth $B_{uplink}$ of the remote uplink. Usually $B_{uplink} < B_{downlink}$, whereas the upstream bandwidth of the remote terminal limits the achievable bandwidth. The prototype implementation was tested in a 3G/WCDMA network with a nominal bandwidth of $B_{downlink,nom}$ = 384 kbit/s downstream and $B_{uplink,nom}$ = 128 kbit/s upstream. We observed a fixed transmission delay of 60 ms upstream and 100 ms downstream with an actual obtained bandwidth of $B_{available}$ = 92 kbit/s in the transfer. The measured file transfer time for a 100 kB file (e.g. picture) was 9 s and for a 5 MB file (e.g. MP3 song) 450 s. The dependence on the limited upstream bandwidth can be reduced by caching in the network or simultaneous downloading from several sources. The ultimate limitation of the transfer speed is determined by the local downstream bandwidth. For resources other than files, the required bandwidth varies widely with the type.

#### NAT and firewall traversal

NATs and firewalls are deployed in several networks to increase the available address space and to separate the customer network from the operator network. A firewall can especially in cellular networks be motivated by the need to prevent malicious traffic to the terminal, which would cause extraneous costs to the user. Firewalls also give the operator added control over the traffic. A NAT/firewall prevents incoming TCP connections, which inhibits direct connections between terminals, as required in fully decentralized architectures and in the access connection.

During our experiments, none of the Finnish cellular operators provided direct IP connectivity between terminals. The firewall of the used network prevented incoming TCP connections. UDP ports were kept open for incoming packets a limited time after an outgoing packet. We were able to pass this restriction by sending periodical keep-alive packets to the SIP server in order to keep the UDP connection open. This creates extra traffic. SIP-based initiation of the access connection succeeded via the SIP server as each terminal actively maintained a connection to the server. However, a TCP connection for a file transfer could not be established between terminals. The transfer must therefore be assisted by a TCP relay in the fixed network.

### 4.8.5 Summary of evaluation

Based on our testbeds, we summarize in Table 4.6 the validity of the postulates for successful deployment. We conclude that a resource sharing service is possible to implement today provided that certain constraints are observed.

**Table 4.6. Validity of postulates.**

| Action | Validity | Motivation |
|---|---|---|
| Postulate 4.1 | Valid | We successfully implemented a resource discovery application with the APIs and the user interface available on a Nokia smart phone. |
| Postulate 4.2 | Valid | Our implementation runs smoothly and consumes less than 520 kB of RAM in normal operation. |
| Postulate 4.3 | Valid with constraints | Despite departing from the original specification, we successfully implemented SIP-based signaling without changes to the SIP-stack or the proxy. |
| Postulate 4.4 | Valid | In the used network, the network-related delay of all operations are less than 1 s. |
| Postulate 4.5 | Valid with constraints | We successfully used SIP-based signaling but this required an actively maintained connection to the server. |
| Postulate 4.6 | Valid with constraints | Direct access connections could not be set up between terminals but access can be supported by a relay in the fixed network. |

Since our implementation, other implementations have appeared, including Symella [KFM07], SymTorrent [KEP07], and MobTorrent [ENK08]. These applications behave much like their counterparts in the fixed network, with a slightly reduced functionality (especially the upload restrictions of Symella and MobTorrent). The fact that these applications appear shows that mobile peer-to-peer is technically feasible and an upcoming type of application. Some features of the mobile platform still restrict implementing the full set of capabilities found in fixed peer-to-peer applications. Ekler et al. report [ENK08] that the limitation of the number of concurrent sockets, a single concurrent connection attempt, long timeouts and inability of random access to files in the J2ME platform limited the performance of MobTorrent.

## 4.9 Technical feasibility of decentralized group-based peer-to-peer

In this section, we study the decentralized group-based peer-to-peer application described in Section 4.6. The purpose is to verify the feasibility of the group-based access control and the Direct Index algorithm, both published in [Bei07b], using a prototype. In particular, we test the validity of the following postulates:

- *Postulate 4.7:* The application can be implemented with the APIs and the user interface available on today's mobile phones.
- *Postulate 4.8:* Today's mobile phones have enough memory and CPU power to run the application smoothly.
- *Postulate 4.9:* The Direct Index algorithm operates correctly.
- *Postulate 4.10:* Groups can be managed in a distributed way.

- *Postulate 4.11* The access to resources can be restricted so that only users of a certain group can access it.
- *Postulate 4.12:* Signaling can bypass the NATs and firewalls of today's networks.

### 4.9.1 Prototype

The proposed concept based on the Direct Index algorithm was implemented in Java 2 Micro Edition (J2ME) using the Mobile Information Device Profile (MIDP). Both Series 60 and Sun's Java libraries are used. The prototype covers the main functionalities. The user can define groups by indicating a set of categories for each contact. Access to resources can be limited to specified groups. Searching is based on a keyword. For each matching resource the IP address of the resource's location is obtained. The actual download is not implemented as it is not part of the resource discovery problem. Bloom filters are used to compress the index, which is exchanged directly with each known peer. Policies such as interest horizon, distribution horizon and minimum connectivity are excluded from the implementation. The details of the implementation are reported in [Pan09].

A binary protocol [Bei07c] is used instead of a SIP based protocol in the prototype in order to simplify implementation and reduce bandwidth. The protocol defines six types of messages: a Hello request, a Hello reply, an Update request, an Update reply, an Acknowledgement, and an Error message. All messages share a common header specifying the message type, the message length and the sender's and the receiver's IP addresses. In the prototype, users are identified by their IP address. This makes it in practice necessary to allocate a fixed IP address to the user.

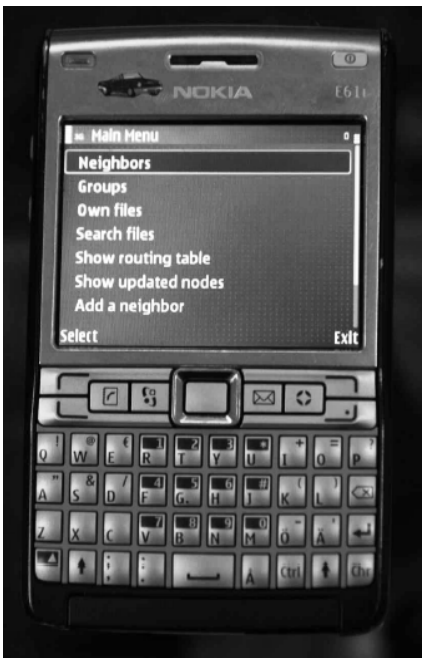


**Figure 4.29. Direct Index prototype application on Nokia E61i [Pan09].**

The software is dived into five packages: A Logic package implementing the Direct Index algorithm and the program control, a Packet package for generating and parsing messages of the binary protocol, a Network package controlling sending and receiving messages, a Client package controlling the user interface and a Utility package for various support functions such as configuration, Bloom filters and logging. The central data structure is the peer table containing the following entries for each peer: an IP address, a group list, an update timer, a neighbor list with an IP address and group list for each neighbor, and an index list with an IP address, ID and sequence number for each resource descriptor. The application is shown in Figure 4.29 on a Nokia E61i.

### 4.9.2 Feasibility tests

The prototype is tested in four environments:

- *Testbed D1*. The network consists of 7 emulated mobile phones. This verifies the correct operation of the Direct Index algorithm, the group management functionality and the access control.
- *Testbed D2*. The network consists of two phones (Nokia E61i and Nokia 5800 Xpress Music) connected with a WLAN network. This verifies that the application works in real telephones.
- *Testbed D3*. The network consists of two phones (Nokia E61i and Nokia 5800 Xpress Music) connected with a cellular network. This verifies that the application works in a commercial cellular network (Elisa).
- *Testbed D4*. The network consists of two phones (Nokia E61i and Nokia 5800 Xpress Music) connected with a cellular network. This verifies that the application works in a prototype IMS network (Octopus).

The test with Testbed D1 shows that the group management functions work. New contacts can be created and groups can be added to and removed from a contact. The prototype is simplified in that the other party automatically accepts all invitation requests, whereas the group is activated on the connection. The tests show that the Direct Index concept works. The local resources are correctly distributed to all nodes in a group, and each node obtains a correct routing table of each group in which it as a member. Nodes can search for resources by examining their remote index. Two isolated islands can be connected and the groups with the same names can be correctly united and synchronized. Synchronizing the removal of resources required special consideration but finally it was successfully implemented. The tests also verify correct operation of the access control. Resources can be shared to and removed from specified groups. Resources were not found by a user having no groups common with the sharing user, or if the groups with a common name are isolated.

Testbed D2 required replacing some libraries but after that the application successfully worked in Series 60 phones over WLAN. Testbeds D3 and D4, however, failed because of messages dropped by firewalls or NATs. These tests show that direct communication between mobile phones must be enabled in operator networks in order for the

application to work. Alternatively support from a relaying node in the fixed node is needed. This requires adding support for the relay in the application. A more generic approach, which we described in Section 4.6, is to use SIP signaling, whereas the SIP proxy operates as a relay. The connection to this proxy must be maintained by sending UDP traffic periodically.

### 4.9.3 Summary of evaluation

Based on our implementation, we summarize in Table 4.7 the validity of the postulates. The main obstacle for the proposed application is the NATs/firewalls preventing traffic between cellular terminals. Apart from this network-related issue, the prototype demonstrates that the Direct Index algorithm works in practical implementations.

**Table 4.7. Validity of postulates.**

| Action | Validity | Motivation |
|---|---|---|
| Postulate 4.7 | Valid | We successfully implemented the application using J2ME with Java and Series 60 libraries and the user interface functions of Java. |
| Postulate 4.8 | Valid | Our implementation runs smoothly on various mobile phones. |
| Postulate 4.9 | Valid | The local index of each node was distributed correctly to all nodes in the common groups. |
| Postulate 4.10 | Valid | The user can successfully define the groups of a contact and the groups common with the contacts are activated. |
| Postulate 4.11 | Valid | A resource shared to a given group is visible to all members of that group but not to other nodes. |
| Postulate 4.12 | Invalid, requires changes | Messages could not be sent directly between terminals in current cellular networks due to NATs/firewalls. The solution only works in WLAN networks. To bypass the restriction, the protocol must be relayed via a fixed node. |

## 4.10 Summary

This chapter examined the use of overlays for resource discovery in cellular networks. The chapter focused on technologies enabling resource sharing between mobile users in a peer-to-peer fashion. We first performed a set of user studies to obtain knowledge about user expectations and opinions that can guide the technical development. These studies particularly showed the need for observing the underlying social relations and providing access control. This need may not only be due to the personal nature of the shared content, but also because of the unwillingness to allow the device capacity and bandwidth to be used by strangers.

We presented a scenario where the service is provided by the operator or a third-party service provider. In this case, it is motivated to use a centralized approach, where interconnected servers in the fixed network support the overlay by taking over most of the processing and storage load. We proposed utilizing the IP Multimedia Subsystem (IMS) by implementing the resource discovery as an Application Server (AS). We identified the functions performed by the AS, and recognized different types of access control and group management that can be provided by the network. We identified a few properties that are desired in a commercial resource discovery service. These include separation between interior and exterior links, the ability to establish overlay connections based on inter-operator agreements, the ability to use any internal topology, index confidentiality and internal topology confidentiality. We evaluated various architectures against these criteria. We concluded that IPIC shows several good properties for the use in a commercial peer-to-peer service.

In a network without centralized operator support, it becomes important to take user relations into account, since the users must provide a share of their device and network capacity to other users. As a social network represents a fairly comprehensive model of user relations, it provides a good foundation for a resource sharing system. We provided a theoretic model of a social network and formed methods for distributed group management, access control and policies on top of it. We presented two proactive resource discovery schemes implementing the above methods: index flooding and Direct Index. We discussed the feasibility of a proactive scheme and described index compression based on Bloom filters to reduce the storage and transport requirements.

We studied the use of the Session Initiation Protocol (SIP) for signaling in resource discovery. Our motivation is that SIP is an established protocol utilized in the IMS with existing support functions like charging, security, NAT/firewall traversal and management. We defined two schemes for SIP-based resource discovery: the first based on INVITE and the second based on SUBSCRIBE/NOTIFY.

We evaluated the technical feasibility of peer-to-peer systems in cellular networks using prototypes arranged into a set of testbeds. We showed that neither today's devices nor the network pose serious restrictions. The major challenge is to handle NAT/firewall traversal. Our prototypes showed that a SIP-based signaling scheme works after certain modifications. Furthermore, the decentralized group-based peer-to-peer approach worked as expected.

# Chapter 5

# Resource discovery in mobile ad hoc networks

This chapter studies resource discovery in an ad hoc network formed between consumer devices. As the capabilities of the nodes are expected to vary significantly in such a network, we aim at allocating the load to the nodes with most capacity. To reach this goal, we form an overlay network according to the relative node capacities. We take two approaches for the construction of the overlay: the first approach is based on local decisions while the second approach forms clusters that are interconnected. Since resource discovery is closely related to routing in ad hoc networks, we also discuss issues related to routing. The aim is to enable a combination of proactive and reactive operations. The chapter begins with an introduction to the topic and an overview of the related research.

## 5.1 Introduction

Ad hoc networks are wireless networks that are established between a set of nodes without the support of any fixed infrastructure. Initially ad hoc networks were developed with military and rescue operations in mind. However, now the technology has been proposed for a wide range of applications ranging from meetings and conferences to personal area networks, vehicular networks, and control networks for automation. An ad hoc network can be a freestanding network or connected to another network. The network can also be connected as a leaf to a LAN, extending the range of the access point. Closely related to ad hoc networks are sensor networks, transporting measurements from a large set of wireless sensors to a sink. Usually the network has a temporary nature, such as for the duration of a meeting or conference.

A pure ad hoc network has only wireless links, but practical networks are likely to incorporate fixed devices and gateways to external networks like the Internet as well. Ad hoc networks can be implemented with various underlying physical network technologies, such as IEEE 802.11

[IEEE2007] wireless LANs (WLANs) and Bluetooth [Bluetooth]. The nodes can be consumer devices, such as laptops, PDAs, mobile phones, and lightweight laptops (netbooks). Nodes typically have a limited processing and battery power. Especially among different consumer devices, the capacity varies widely. The network is characterized by a dynamic topology with unreliable links with low bandwidth. All nodes act as routers, which gives a high number of possible routes. The address space is flat, which implies that a separate route entry to each node is needed. Altogether, these properties make routing in ad hoc networks challenging.

As traditional routing protocols, such as OSPF and RIP, perform poorly in ad hoc networks, numerous routing protocols have been developed specifically for these networks. The protocols can be divided into proactive and reactive protocols [Fee99]. A proactive protocol continuously maintains a route to each node in the network. Well-known proactive protocols include Optimized Link State Routing (OLSR) [CJ03] and Destination Sequenced Distance-Vector (DSDV) [PB94]. A reactive protocol aims to reduce traffic by creating a route only when a packet is sent to a given destination. Dynamic Source Routing (DSR) [JHM07] and Ad hoc On-demand Distance Vector (AODV) [PBD03] are examples of reactive routing protocols. In general, proactively maintained routing information ages quickly when mobility is high or when only a fraction of the nodes communicate. On the other hand, reactive protocols have an unnecessarily high route request overhead in a stable network or when a large number of nodes communicate. Thus, both approaches are suitable for a specific set of scenarios. To combine the advantages of these approaches, protocols based on both proactive and reactive routing have been proposed, including the Zone Routing Protocol (ZRP) [PH99].

Ad hoc networks share several properties with peer-to-peer networks, including the flat and changing topology, the low reliability of nodes, the distributed control, the security problems, and the need for co-operation between nodes [SGF02]. The techniques for routing and service discovery in these networks are therefore similar, with a large dependency on flooding. Yet, the techniques differ due to the fact that the topology in ad hoc networks is determined by the physical locations whereas peer-to-peer networks are free to form any kind of overlay topology [SGF02].

Whereas the purpose of resource discovery solutions in fixed networks usually is to locate application layer resources, ad hoc networks additionally need resource discovery for locating network layer resources. Network layer resources include storage, printing, domain name service (DNS), telephony servers, relays, and gateways to external networks. Because of the lack of centralized control and fixed servers in combination with the varying availability of nodes, the network services must often be located on a per-session basis. In ad hoc networks resource discovery is preferably implemented at the network layer.

To locate a small amount of fairly static resources, manual configuration suffices. However, in most practical applications an automatic way to discover resources fulfilling a given criteria is needed. Such a system automatically selects the best resource for a given purpose

in a dynamic environment where new resources are added and existing resources may fail or be unavailable at times. As the resources to locate mainly are services, we usually talk about *service discovery* instead of resource discovery, and *service directories* instead of indices. Service discovery allows devices to automatically locate network services based on their attributes. Typically, the service is provided by ordinary nodes or the service is distributed between these. Service discovery therefore provides the means for advertising a service available to other devices. Locating a gateway for communication with infrastructure-based networks can also be considered as service discovery.

## 5.2 Related research

Mobile ad hoc networks have gained large interest in the research community. In the early years, most research efforts were spent on developing and improving routing protocols, resulting in a multitude of routing approaches. A good survey over routing protocols for ad hoc networks can be found in [AWD04]. As the research scope has widened, research topics have included developing support for service discovery [MBB06], peer-to-peer networking [DB04] [OSM+05], security [YLY+04], delay-tolerant networking [JFP04] and specific applications for ad hoc networks. Several experiments with implemented ad-hoc networks have been performed [KM07].

### 5.2.1 Dominating sets and virtual backbones

Most ad hoc routing protocols, both proactive and reactive, are to some extent based on flooding. A proactive routing protocol floods the network in order to distribute updated routing information. A reactive routing protocol uses flooding to distribute a route request for locating the destination, often using an expanding ring search. Flooding in ad hoc networks differs from flooding in a fixed network in that the packet must not be sent to each neighbor separately. Instead, packets are transmitted on a broadcast address, which allows reception by all nodes within the sender's radio coverage. Consequentially, a node receives a flooded packet multiple times, as repeated by each of its neighbors.

A *dominating set* (DS) is a subset $D$ of nodes in a graph $G = (V, E)$ chosen so that every node either belongs to $D$ or has a neighbor that belongs to $D$. A *connected dominating set* (CDS) is a DS that is connected, i.e. there is a path from every node in the CDS to all the other nodes in the CDS. In a dense network it is possible to significantly reduce the flooding overhead using a connected dominating set. Flooded packets are forwarded only by nodes belonging to the CDS. Since every node either is in the CDS or has a neighbor in the CDS, all nodes will receive the flooded packet. The CDS operates as a *virtual backbone* spanning the network, and can be used to efficiently distribute routing information [DBB+97]. To reduce the overhead, the CDS can be minimized. Thus, the purpose of many solutions for efficient flooding is to find a *minimum connected dominating set* (MCDS). Finding the MCDS has been proved NP-complete [GJ79, LK01]. Therefore, various heuristic algorithms have

been developed, including [GK98] and [WAF04]. An MCDS can be approximated using a spanning tree, whereas the purpose is to maximize the number of leaves. The leaf nodes do not participate in flooding. As global information is not available, heuristic algorithms such as the self-pruning and dominant pruning in [LK01] have been proposed. An approximated MCDS is also formed indirectly through certain clustering algorithms.

### 5.2.2 Clustering

Kwon and Gerla [KG02] define clustering as the grouping of nodes into a manageable set. Roles are assigned to certain nodes, such as cluster-heads, gateways and ordinary nodes. Normally, only cluster-heads and gateways participate in forwarding, thus, these nodes constitute a virtual backbone. One good overview and categorization of different clustering algorithms is given in [YC05]. In [KG02] clustering algorithms are classified depending on whether the clusters are overlapping or disjoint. The classification also separates between algorithms generating two-hop clusters, whereas the distance between two nodes in a cluster is at most two hops, and multi-hop clusters, whereas the cluster size can be larger.

A large set of schemes, including [GT95] and [Bas99], create two-hop overlapping clusters. In such a cluster each node is at most two-hops from other nodes in the cluster and there is a cluster-head in the center of the cluster that is able to contact all nodes over a single hop. Some nodes belong to two or several clusters, acting as gateways between these clusters. The cluster-heads form a dominating set, while the gateways and cluster-heads together form a CDS. The aim is to minimize the number of gateways and cluster-heads, and consequently to approximate an MCDS.

Besides minimizing the CDS, clustering schemes have been developed with other goals. Some schemes, including Adaptive Multi-hop Clustering (AMC) [OIK03] and Degree-Load-Balancing Clustering (DLBC) [AP00], aim to balance load by controlling the size of clusters. Too large clusters may overload the cluster-head while a too small cluster size results in too many clusters and thereby inefficient routes. In mobility-aware clustering schemes, such as MOBIC [BKL01] and the Distributed Dynamic Clustering Algorithm (DDCA) [MZ01], the aim is to put nodes with similar mobility patterns in the same clusters. Clustering is based on the relative speed differences between nodes. The links within the cluster remain stable as nodes move with the same speed. In the On-Demand Weighted Clustering Algorithm [CST00], the cluster-head selection is based on the weighted combination of degree-difference, distance to neighbors, average speed and cluster-head serving time. Some schemes aim at reducing the maintenance costs. The motivation is that the need to maintain the cluster structure may reduce the benefit of clustering. Of these, Passive Clustering [KG02] is able to eliminate the active messaging by transporting clustering information using two bits in the normal user traffic.

While clustering improves routing efficiency, it comes at a cost. Firstly, explicit message exchange between node pairs is required to maintain the structure. Structure maintenance may be costly in a dynamic network. Secondly, some clustering schemes suffer from the so called

ripple effect of re-clustering: re-election of cluster-head due to some local event in one cluster may cause the restructuring of the whole network. Thirdly, most schemes are based on distinct phases for constructing and maintaining the clustering structure. This often requires that the network is relatively static during construction. Finally, several rounds of computation may be required for cluster formation. Different algorithms are affected to various extents by these limitations. [YC05]

### 5.2.3 Service discovery

Several service discovery protocols have gained the status of industry standard in wired networks. Jini [Sun99] is based on a centralized service directory, the Simple Service Discovery Protocol (SSDP) used by Universal Plug and Play (UPnP) [Upnp08] is based on multicasting, and the Service Location Protocol (SLP) [GPV+99] operates both with and without service directories. The service discovery protocols developed for fixed networks are not suitable for ad hoc networks as they are based on different assumptions, most critically regarding mobility [MBB06].

In ad hoc networks, both routing and service discovery can be implemented with similar techniques. These problems, however, have a major difference: In routing, the identity of a node is known; on the other hand, a service may be provided by several devices and the user wishes to contact the device that best fits some given criteria. [MBB06]

Several solutions for service discovery have been proposed specifically for ad hoc networks. Good overviews over service discovery protocols can be found in [SBW07] and [MBB06]. Mian et al. [MBB06] divide service discovery architectures into directory-based, directory-less and hybrid architectures. The operation principle of most *directory-less solutions* is that the service provider proactively broadcasts service advertisements to the network using flooding with a limited TTL. Nodes maintain a cache of the advertisement they are interested in. If, upon a resource request, no matching resource is found in the cache, a node can also reactively broadcast a search request to the network. The service discovery can also operate completely reactively. Examples of directory-less architectures are GSD [CJY+02] and Konark [HDV+03]. While directory-less architectures typically do not form an overlay network, overlay-like structures are used in the alliances of Allia [RCJ+02].

From the perspective of this work, *directory-based solutions* are more interesting. In these, certain nodes act as service directories. Most directory-based architectures use an overlay network. In [KT03] this overlay is a virtual backbone and the directory is maintained by the nodes in the virtual backbone. In [KKO03a] the overlay is a set of service rings formed by nodes physically close and offering similar services. A service access point in each ring stores the service directory. In both solutions, services are advertised to a node acting as directory, and all directories are queried in searching. In [KKO03b] the overlay consists of groups of nodes called lanes. The directory is replicated to nodes within the same lane. Service requests are sent to all lanes through anycasting. It is worth noticing that traditional peer-to-peer overlays are rather unsuitable for service discovery in resource-constrained ad hoc networks as they are completely decoupled from the physical topology.

In *hybrid solutions*, a directory is used if there is one within a limited scope. Thus, the directory only serves the nodes within its scope. If no directory is within the scope, a directory-less approach is used. In ad hoc networks hybrid solutions are uncommon [MBB06].

## 5.3 Our contribution

This chapter studies how an overlay can be formed to support resource discovery systems in an ad hoc network and, in particular, how the diversity of node capacities can be utilized to support resource discovery. The fundamental idea is to identify the nodes with high capacity and concentrate the processing and storage on these nodes. As the work concerns the network layer, also routing solutions are presented.

In Section 5.4 we discuss the feasibility of combining proactive and reactive routing. We find similarities between ad hoc networks and overlay networks, and therefore we propose how the Search/Index Space model can be applied to ad hoc networks. This material has not been published earlier.

In Section 5.5 we present the problem of routing and service discovery in ad hoc networks consisting of devices with varying capabilities. We make the observation that in an ad hoc network formed by consumer devices, the devices with high mobility typically have low capacity, and vice versa. Based on this observation, we propose classifying the devices according to their capacity and mobility. We take two approaches for this classification, presented in Section 5.6 and Section 5.7, respectively. In the first approach, published in [CBK02], [CBK04], [CGK+04], [CKB05], [CVK+06], the class depends on local decisions. This work has been led by Jose Costa-Requena, where the present author has participated in developing the node classification and task distribution. In the second approach, published in [BKC05], the class depends on the neighboring devices. A shorter version of [BKC05] is published in [BKC06].

In Section 5.6 we propose a modular architecture based on the node classification. This architecture combines proactive and reactive routing and service discovery so that the nodes with high capacity and low mobility support the nodes with lower capacity and higher mobility. The former type of nodes run proactive protocols while the latter utilize a reactive protocol. We have published the architecture in [CBK02] and [CBK04]. The scheme described in this work is slightly modified.

Section 5.7 takes a different approach to the above architecture. We propose an algorithm for clustering nodes according to their capacity and mobility. The algorithm connects nodes in the cluster using a tree-shaped overlay, where the most powerful and least mobile nodes are in the center of the cluster. We further propose an algorithm for connecting these clusters together, forming an overlay spanning the whole network. We perform simulations that compare the different variants of the algorithm and describe the formed network. We finally provide a framework about how this overlay can be utilized in routing and service discovery so that the load is concentrated in the nodes with the highest capacity. The above algorithms, simulations, and descriptions are published in [BKC05] and

[BKC06]. We also complement the published results with some unpublished results.

## 5.4 Combining proactive and reactive routing

For ad hoc networks both proactive and reactive routing protocols have been proposed. Proactive protocols maintain up-to-date routing tables for all destinations in the network. Synchronizing routing information consumes a considerable amount of bandwidth. If nodes are mobile, whereas the topology is constantly changing, a large part of the routing information will never be used before it is aged. As the mobility increases, the frequency of routing updates must be increased to keep up with the topology changes. Reactive protocols have been developed for this kind of dynamic topology. A reactive protocol creates the route on demand as the first packet for a given destination is encountered. Route generation creates a burst of traffic, but once the route is available, no routing traffic is required. The routing information is stored as long as traffic is sent on the route. As a consequence of node mobility, the routing protocol may need to repair or regenerate routes, which generates traffic. While proactive protocols constantly generate traffic, which increases with increasing mobility, the reactive protocol generates a given amount of traffic per required route. Therefore, a reactive protocol is a more efficient choice if only a few routes are active at a given moment or if the network is very dynamic so that routes are short-lived. The choice between proactive and reactive routing thus fundamentally depends on the relationship between the mobility and the number of active routes. As different networks have different properties, any given protocol has not been considered optimal for all cases.

To combine the good properties of both proactive and reactive routing, hybrid routing protocols have been proposed. One hybrid routing approach is the Zone Routing Protocol (ZRP). In ZRP, proactive routing is used within a zone around each node. Reactive routing is used for packets to destinations outside the sender's zone. The proactive routing information is through the concept of bordercasting utilized to guide the reactive route request so that the overhead is lowered. Limiting the proactive routing protocol to a limited zone reduces the overhead of the broadcasting of routing information. Exact information about destinations far away is not required – it is sufficient to guide the request in the right direction and only near the destination utilize precise routing information. Additional benefit comes in networks where most traffic is directed to nodes nearby. Although these are important advantages, they do not alone motivate the use of a hybrid routing protocol.

The major advantage of hybrid routing is that the optimal strategy in most situations is between fully reactive and fully proactive routing. This can be motivated by an analysis similar to our Search/Index Space model. In this model, route requests correspond to search requests and route updates correspond to proactive index updates. Mobility directly affects the frequency needed for distributing route updates: the more often the topology changes, the more often the routing information must be

distributed. The frequency of route requests depends on the usage. The relationship between these frequencies, like in the Search/Index Space mode, determines the optimal balance of proactive and reactive operations. Both the proactive and reactive distribution have a given overhead that is protocol dependent. Based on these similarities we can study ad hoc networks with (2.32). We present the interpretation of the Search/Index Space model in ad hoc networks in Table 5.1 using PIC and ZRP as example algorithms.

**Table 5.1. Applying the Search/Index Space model to overlay and ad hoc networks.**

| Symbol in the Search/Index Space model | Interpretation in an overlay network | Interpretation in an ad hoc network |
|---|---|---|
| $P$ | The cluster size (in PIC) is the proactive component. | The zone size (in ZRP) is the proactive component. |
| $N$ | The optimal cluster size increases as the network size increases. | The optimal zone size increases as the network size increases. |
| $f_s$ | The optimal cluster size increases as the frequency of searches increases. | The optimal zone size increases as the frequency of route requests increases. |
| $f_i$ | The optimal cluster size decreases as the frequency of index updates increases. | The optimal zone size decreases as the mobility increases. |
| $\Omega_s / \Omega_i$ | The optimal cluster size increases if the overhead of the search algorithm is higher than the overhead of the index distribution algorithm. | The optimal cluster size increases if the overhead of the route request algorithm is higher than the overhead of the route update algorithm. |

We do not claim that (2.32) can be applied to ad hoc networks as such, but we argue that the parameters affect the balance in the same direction. Simulation studies in [PH99] support our assumptions:

- an optimal zone size that reduces the traffic exists,
- an increasing mobility decreases the optimal zone size,
- an increasing network size increases the optimal zone size, and
- an increasing node degree increases the optimal zone size.

The last result seems to indicate that the node degree affects the reactive search algorithm in ZRP more than it affects the locally operating proactive routing algorithm.

In this work, we do not aim to provide a detailed generic analysis adapted for ad hoc networks because the situation is more complex than that of overlay networks. In ad hoc networks, the proactive and reactive protocols operate under different conditions because the protocols work either on a different scale or in different parts of the network. For example, in ZRP proactive routing is used on a local scale while reactive routing is used on a global scale. In hierarchical protocols (e.g. CEDAR) the logical upper-layer links pass through several physical lower-layer

nodes. This is in contrast to hierarchical overlay networks, where all hierarchical layers use comparable links. Analyzing how mobility affects these scopes can be difficult and varies widely between protocols. Mobility must be modeled to obtain the frequency of proactive route updates in order to provide reliable routes. Furthermore, the overhead of the proactive and reactive protocols must be modeled separately. The overhead often depends on the node degree, but analytically estimating the effect of the degree on the overhead of bordercasting is difficult.

## 5.5 Utilizing capacity heterogeneity

In this work, we study an ad hoc network formed by consumer devices. The devices in this kind of network are assumed to have a large variation in capacity and mobility. In addition to mobile devices, we expect the network to contain fixed devices as well. This type of scenario needs different resource discovery architectures than those designed for cellular networks, where the devices are mainly mobile phones with rather similar capacity. Moreover, as the resources to be discovered are often network layer entities, the resource discovery system should operate at the network layer. While capacity heterogeneity often is considered as a problem, we instead try to utilize the situation to our benefit. We aim to allocate most of the load to the nodes with the highest available capacity.

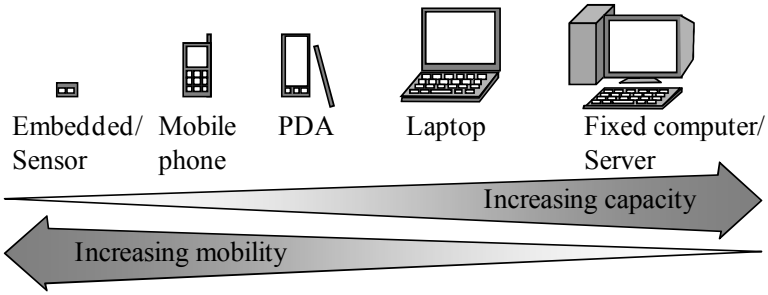


**Figure 5.1. Typical relationship between capacity and mobility.**

Examining the capacity and mobility of popular customer devices, leads us to the following assumption: the devices with high mobility typically have low capacity and vice versa. With capacity we refer to properties of the device such as processing power, memory, and battery power. Especially the power supply is considered crucial from a capacity perspective. As depicted in Figure 5.1, the highest capacity is typically found in the stationary devices running on a constant power supply, such as fixed computers and servers. Laptops have lower capacity and are at times connected to a fixed power supply. Mobility is reduced as the laptop seldom moves while being used. Mobile phones have limited capacity and almost constantly run on battery power. They are used while moving. Personal data assistants (PDAs), residing between mobile phones and laptops, are currently being replaced with miniature laptops and smart phones. In the future, several small portable devices such as music players and health sensors are becoming networked and various household and entertainment devices will have embedded computers. It

may not be feasible to connect the smallest devices to all existing long-range networks. Instead, the smaller and capacity-limited devices should use the support of surrounding higher-capacity devices.

## 5.6 Virtual backbone for combining proactive and reactive protocols

In [CBK04] we propose to classify the devices into two groups: smart nodes and dummy nodes. A *smart node* is defined as a node with low mobility and high capacity and a *dummy node* as a node with high mobility and low capacity. We propose to utilize capacity heterogeneity to support routing and service discovery. Smart nodes are interconnected with other smart nodes and run a proactive routing protocol, such as OLSR, which requires larger storage, steady available bandwidth and high stability (low mobility). These nodes support the weaker nodes, which run a reactive routing protocol, such as AODV, requiring minimal persistent data. The smart nodes fundamentally form a virtual backbone.

In resource discovery, it makes sense to place indices on the most stable and powerful nodes. These nodes act as upper-layer nodes (super-nodes) in a resource discovery system. Other nodes are connected as lower-layer nodes (ordinary nodes) to these nodes. The main challenges are thus to (1) identify high capacity stable nodes with low mobility, and to (2) allocate one such node to each of the remaining nodes. In our smart/dummy node scheme, indices are located at the smart nodes. These nodes are connected by a virtual backbone consisting of the most stable nodes and links in the network, which is used for search and index distribution between the smart nodes.

### 5.6.1 Local role determination

The simplest way to categorize devices into smart and dummy nodes is to perform local decisions based on the node type and capacity. We take this approach in [CBK04]. Certain devices with low capacity, such as PDAs and phones, are by default dummy nodes, while laptops and fixed computers are smart nodes. Roles can change dynamically. For example, a smart node may become a dummy node when the battery is depleted. Role determination can also utilize information about surrounding devices, so that a node may act as a smart node if there is no smart node within a given distance even though it is configured as a dummy node. The role determination mechanism can be different in different nodes.

To implement the role determination, each node calculates its *preference value*. The initial value is the default preference that is preconfigured depending on node type. The value is adjusted depending on the resources. For example, if the battery is half empty, the preference value can be halved. The calculation of the preference is a local decision, which can be different in different nodes. A node acts as a smart node if the preference value is above a *threshold*; otherwise it is considered as a dummy node. The threshold depends on the preference of the surrounding devices. The preference value is transported to all neighbors in the neighbor detection protocol. In one approach, the threshold is the average

preference value of the surrounding device with an added constant. The constant determines the percentage of devices acting as smart nodes.

A node may also decide to become a smart node if it has a stable link to another neighboring smart node. A link is considered as stable if it has been operating a given minimum time. Such a case appears if both nodes are static or move in the same direction with similar speed. It is beneficial that smart nodes are connected with stable links.

### 5.6.2 Forming a backbone

If each node individually determines its role, there is no guarantee that all smart nodes can communicate with each other. Where several smart nodes are neighbors, an isolated island of communicating smart nodes is formed. A smart node therefore needs to detect other smart nodes several hops away. Once the surrounding smart nodes are found, virtual links are set up to some of them. An upper hierarchical layer is formed by the smart nodes. The links on the upper layer are either physical links or virtual links. Smart nodes may thus need the support of dummy nodes to communicate. A virtual backbone is formed by the smart nodes and the dummy nodes connecting islands of smart nodes.

When a node becomes a smart node it performs an Attach procedure. The smart node sends a broadcasted message with a TTL of one. The message describes the node's capabilities. A smart node within the range can decide whether to accept or deny the attach request. The request is denied if the number of surrounding smart nodes is above a defined threshold. If the request is accepted, the attaching node can establish a link to it.

If no smart nodes are within a single hop, two alternatives are possible. A dummy node may be aware of a smart node, to which it relays the request. The dummy nodes cache the address of the smart node. Alternatively, the neighboring dummy nodes relay the request to all their neighbors. In either case, the scope is limited by a TTL. The attaching smart node uses an expanding ring search, which finishes when a given number of surrounding smart nodes are found. The attaching smart node consequently receives routes to a desired number of smart nodes, to which it selects the ones to which links are established. Once the smart node has links, either direct or virtual, to its surrounding smart nodes, it starts providing its proactive services. Most importantly, it starts a proactive routing protocol operating between all smart nodes in the network. The attach request can be implemented as extensions to normal reactive route requests, where the destination is any smart node.

Because smart nodes bear a higher load, users must be motivated to provide a higher amount of their capacity for the good of the network. We analyze the incentive of users to participate as smart nodes in [CKB05]. The study reveals that a rewarding mechanism is needed to provide an incentive for users.

When the capacity of a node is reduced, it becomes a dummy node. The node then performs a Detach procedure, whereas it detaches itself from the surrounding smart nodes and the proactive routing protocol.

When the backbone is used in routing, a proactive routing protocol, such as OLSR, is running between the smart nodes. Dummy nodes

implement a reactive protocol, such as AODV. Dummy nodes experience a reduced bandwidth and battery consumption as they transmit only when user data needs to be sent. To enable communication with dummy nodes, also smart nodes need to implement the reactive routing protocol. Thus, a smart node participates in two routing protocols. A dummy node that requires a route to a given destination sends a route request according to the normal operation of its reactive protocol. The dummy node does not necessarily need to be aware of the existence of a virtual backbone. Once the request reaches a smart node, the smart node replies with the route on behalf of the destination. The proactively obtained routing information consequently becomes available to the reactive protocol.

The work principally continued by Costa-Requena on this architecture has produced the Scalable Ad Hoc Routing Protocol (SARP). Describing the actual routing protocol is out of the scope of this thesis. The routing protocol is presented in more detail in [CGK+04], [CVK+06] and [Cos07].

## 5.7 Clustering based on capacity

A problem with individual role determination in the above scheme is that the control over the density of smart nodes is weak. A high density of smart nodes creates an excessive overhead. A low density makes interconnection of smart nodes difficult. Virtual links through dummy nodes put an additional forwarding overhead on these dummy nodes. Furthermore, it is difficult to select remotely located smart nodes in order to avoid partitioning of the network. Virtual links are also unreliable in highly mobile scenarios. We will therefore take another approach to the construction of a backbone.

In [BKC05] we present an algorithm for dividing nodes with highly varying capacity into clusters and an algorithm connecting these clusters using a virtual backbone. The aims are the following:

- To generate a backbone constituting a CDS that can be used as a platform for service discovery and routing.
- To utilize the diversity in resources and mobility of nodes by allocating most of the traffic to nodes with high capacity and low mobility.
- To operate with a low and constant message overhead and with low computational requirements.
- To operate continuously without distinct phases of computation and no need to restart the computation synchronously in all nodes.
- To maintain the stability of formed clusters, especially for nodes supporting service discovery.
- To avoid the ripple effect of re-computing the whole network due to local changes.

Our algorithm operates with a single message – the Hello message. The Hello message can be integrated into the messages of some other protocol (e.g. the neighbor discovery of a routing protocol), making the overhead of the clustering algorithm minimal. As the message is sent periodically, the overhead is constant and predictable. The objective is to maintain the

clusters stable, so that indices for service discover can reside in specific nodes. It is important to note, that the routes along the backbone are not guaranteed to be optimal in respect to hop-count. Instead, we prefer to form the backbone of the nodes that are stable and have a high capacity.

We represent the ad hoc network using a unit disk graph. Thus, the network is modeled as an undirected graph $G = (V, E)$, where $V$ is the set of nodes and $E$ is the set of edges. All nodes in the network share the same communication channel and every node is assumed to have an identical omni-directional radio transmitter and receiver. A message transmitted by a node is received by every node within a fixed radio-range of radius $Z$. Nodes operate in a promiscuous mode, i.e. they are able to receive transmission directed to other nodes than themselves if the sender is within the range $Z$. We define the neighbors of a node $v$ as the set of nodes $N(v)$ from which $v$ has received a Hello message within a given expiration time. We define the full neighborhood of a node $v$ as $N^+(v) = N(v) \cup v$.

### 5.7.1 Objective of clustering

The basis of clustering is a numerical *preference value* $P(v)$ calculated for each node $v$. The preference value describes both the capacity and the mobility of the node. The value is calculated by a *preference function*, which is not defined in this work. The inputs of the function are numerical parameters describing the capacity, including the available memory, the processing power, and the remaining battery, and the mobility. A device may have a preference that is several orders of magnitude higher than the one of another device. An example of this situation is the comparison of a laptop with a constant power supply to a PDA running out of battery.

The formula for combining the capacity and the mobility into a preference value is implementation specific, provided that the following requirements are satisfied:

1. Low mobility must increase the preference value while high mobility must decrease it.
2. High available capacity must increase the preference value while low available capacity must decrease it.
3. The preference values used by different nodes in the network are comparable.

One approach is to use a weighted sum of the different components constituting the capacity and divide the sum by a value representing mobility. The mobility can be measured, for example, using GPS. However, since the use of physical coordinates imposes additional hardware requirements and does not account for group mobility, we instead prefer describing mobility as a measure of the average frequency of connections and disconnections of neighboring nodes. The preference value can change in time. In order to provide a stable operation, the change can be smoothed using an exponentially weighted average of the current and previous values.

The clustering algorithm divides the nodes of the network into clusters of one or more nodes. Each cluster has a cluster-head which has the locally highest preference. Every node knows and has a route to the cluster-head of its cluster. Specifically to our approach is that a node is not necessarily a neighbor of its cluster-head; instead the path to the cluster-head may traverse other nodes. Clusters are thus multi-hops. Each node knows a direct neighbor that is logically closer to the cluster-head than itself. This neighbor is the *dominator* of the node. The dominator of a cluster-head is the node itself. The cluster can be modeled as a tree. The root in the tree is the cluster-head. A node at the distance $k$ from the cluster-head dominates nodes at a distance of $k$+1.

The purpose of the algorithm is to create clusters so that the preference value increases on each hop toward the cluster-head. Branches starting from the cluster-head follow nodes with decreasing preferences. The capacity of the nodes toward the cluster center increases with the highest capacity in the cluster-head. Nodes forward routing information and, depending on the used routing protocol, user data through the cluster-head. The cluster-head also operates as the index in a resource discovery system. As the index updates are forwarded to the cluster-head, the load is the highest on the higher-capacity nodes near the center.

As the goal is to be stable and lightweight rather than perfectly optimal, there may be situations where the preference order is not strict. The motivation for preferring stability to optimality is the need to allocate rather persistent information, such as indices, to a stable node – the cluster-head. As each node in the cluster has a path to this cluster-head, the cluster-head can be queried, and the queries follow paths that concentrate the load in high-capacity and low-mobility nodes.

### 5.7.2 Node attributes

Every node has a set of attributes summarized in Table 5.2. The clustering algorithm operates by continuously assigning a color, a dominator, and a cluster to each node. Each node $v$ selects one of its direct neighbors or itself as its *dominator*, $dom(v) \in N^{+}(v)$. A direct neighbor refers to a node from which a Hello message has been received within the Hello timer expiration interval. The aim is to select the highest-preference neighbor as the dominator, at the same time observing stability. The selection is described in detail in Section 5.7.3. A *dominatee* of a node $v$ is defined as a node $u$ for which $dom(u) = v$.

Moreover, each node $v$ is assigned a *color* $C(v) \in \{white, green, black\}$ depending on its function in the network. The color is determined by neighboring nodes. The *white* color represents the special case of a node that has recently started or it has no neighbors. Hence, white is the initial color of a node. A *green* node (leaf node) has at least one neighbor. Green is the default color for nodes that have received Hello messages from other nodes. A green node becomes a *black* node if one (or several) of the neighbors selects it as its dominator. Thus, the black node (dominator) is one of the most stable and high-capacity nodes within its surrounding. The aim is to keep a small set of the highest-preference nodes as black nodes while the other connected nodes are green. The

participation of green nodes in message forwarding is minimized. The black nodes form a CDS within a cluster: each connected node is either a black node or a neighbor of a black node. In addition to the color, a node can have an additional role of a *cluster-head* or a *bridge node*.

**Table 5.2. Attributes of a node *v*.**

| Attribute | Symbol | Description | Source |
|---|---|---|---|
| Address | $A(v)$ | A unique identifier of node *v*. | Manual configuration. |
| Preference | $P(v)$ | A value determined by the capacity and mobility of node *v*. | Calculated by the preference function. |
| Color | $C(v)$ | The color of a node *v*, which can be white, green, or black. | Determined by the clustering algorithm. |
| Dominator | $dom(v)$ | A neighbor of node *v* or node *v* itself. | Determined by the clustering algorithm. |
| Cluster | $\pi(v)$ | The identifier of the cluster (address of the cluster-head) of node *v*. | Determined by the clustering algorithm. |
| Cluster-head distance | $hdist(v)$ | The distance in hops to the cluster-head of node *v*'s cluster. | Determined by the clustering algorithm. |

A *cluster-head* is defined as a node *v* for which $dom(v) = v$. Thus, a cluster-head considers itself as the most stable and high-capacity of all surrounding nodes and has selected itself as its dominator. The cluster-head is always a black node but not all black nodes are cluster-heads. Each node *v* belongs to some cluster $\pi(v)$ that is formed around a cluster-head $\lambda(\pi)$. In order to identify nodes in the same cluster, the cluster needs a unique name. The cluster is therefore identified with the address $A(\lambda(\pi))$ of its cluster-head. Other nodes obtain the name of their cluster from its dominator when this dominator is selected. The maximum radius of a cluster can be limited with a *cluster radius limit*, $R_{max}$, so that a node can be at most $R_{max}$ hops from its cluster-head. An example cluster is depicted in Figure 5.2. The arrow from a node *v* to a node *u* indicates that $dom(v) = u$.

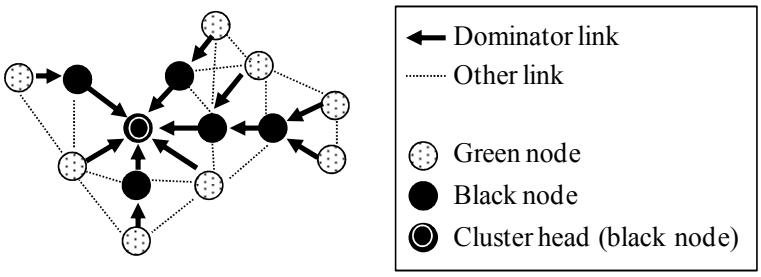


**Figure 5.2. Example cluster.**

The colors are ordered so that *white* < *green* < *black*. In determining the dominator, the color order is the primary selection criteria and the preference the secondary selection criteria. Hence, the black node with the highest preference is chosen if there is an eligible black node. Otherwise, the green node with the highest preference is chosen. A white node is chosen only if there are no black or green neighbors. We call this the (color, preference) order. The reason for observing the color first is to maintain stability: a black node remains black even if a new higher preference neighbor appears. This avoids modifying routes, as these follow the backbone of black nodes.

Both a green and a black node can act as a *bridge node*. A bridge node is defined as a node $v$ that has a neighbor $u$ belonging to a different cluster as itself: $\pi(v) \neq \pi(u)$. The clustering algorithm, described in Section 5.7.3, defines the organization into clusters and the connecting algorithm, described in Section 5.7.4, defines the connection of clusters together.

### 5.7.3 Clustering algorithm

When a node $v$ starts, it sets its attributes to their initial values: $C(v)$ = *white*, $dom(v) = A(v)$, $\pi(v) = A(v)$, and $hdist(v) = 0$. It starts a timer which triggers a Hello message to be sent periodically at a defined interval $T_{hello}$. The Hello message is sent over a single hop on a broadcast address, so that all neighbors of the sending node can receive it. The Hello message sent by $v$ contains the fields ($A(v)$, $C(v)$, $P(v)$, $dom(v)$, $\pi(v)$, $hdist(v)$, $dv(v)$) describing the attributes of $v$. It also piggybacks a small routing table $dv(v)$ described in the following section.

Upon receiving a Hello message from $v$, node $u$ stores the information of the sender in a neighbor table, and starts a timer $T_u$ with the time $T_{expire}$. If the timer $T_u$ expires, the entry for $u$ is removed. The expiration timer $T_{expire}$ is a multiple of the periodical timer $T_{hello}$.

An *attribute determination procedure* is invoked (1) after each received Hello message, and (2) if a neighbor entry is removed due to expiration. The attribute determination procedure consists of two phases. In phase 1, the dominator, cluster and cluster-head distance are determined and in phase 2 the color is determined. We provide two variants of the first phase.

In the first phase, the node $v$ creates a dominator-candidate list $E(v)$, which includes each node $w$ of the neighbor table for which $hdist(w) < R_{max}$. The parameter $R_{max}$ limits the cluster radius so that node $v$ will not choose a dominator that would result in positioning $v$ farther than $R_{max}$ hops from a cluster-head.

In the first variant of this phase, node $v$ also includes itself in the dominator-candidate list:

$$E(v) = \left\{ w \,\middle|\, w \in N^{+}(v), hdist\ (w) < R_{\max} \right\}. \tag{5.1}$$

In the second variant, node $v$ includes itself only if some direct neighbor has chosen $v$ as its dominator:

$$E(v)=\begin{cases}\{w \mid w \in N^{+}(v), hdist(w) < R_{\max}\} \\ \qquad\qquad , if\ \exists u \in N(v): v = dom(u) \\ \{w \mid w \in N(v), hdist(w) < R_{\max}\} \\ \qquad\qquad , otherwise\end{cases} \tag{5.2}$$

The rest is identical for both variants. From the dominator-candidate list, node $v$ selects the node $w$ with the highest (color, preference) rating and sets it as its dominator:

$$\begin{aligned} &dom(v) = w, w \in E(v) \quad so\ that \\ &\forall u \in E(v): C(w) \geq C(u) \vee \left(C(w) = C(u) \wedge P(w) \geq P(u)\right) \end{aligned} \tag{5.3}$$

After that, it updates its cluster

$$\pi(v)=\begin{cases} A(v) & ,if\ dom(v) = v \\ \pi(dom(v)) & ,if\ dom(v) \neq v \end{cases} \tag{5.4}$$

and head-distance attribute

$$hdist(v)=\begin{cases} 0 & ,if\ dom(v) = v \\ hdist(dom(v)) + 1 & ,if\ dom(v) \neq v \end{cases} \tag{5.5}$$

In the second phase, node $v$ determines its color $C(v)$ by checking the dominator $dom(u)$ of each direct neighbor $u$.

$$C(v)=\begin{cases} black\ , if\ \exists u: u \in N(v): dom(u) = v \\ green\ , if\ |N(v)| > 0 \vee \forall u: u \in N(v): dom(u) \neq v \\ white\ , if\ |N(v)| = 0 \end{cases} \tag{5.6}$$

The difference between the two variants is illustrated in Figure 5.3, where nodes are marked with letters and $P$ indicates the preference. In the first variant, node A notices that it has the highest (color, preference) of the nodes A, B, and C. It selects itself as its dominator. The number of cluster-heads is reduced in the second variant. Node A again detects that it has the highest (color, preference) rating. Since no neighbor has selected A as its dominator, A selects its dominator by comparing the (color, preference) of B and C. This node, B, becomes a black node. Note that if we limit the cluster radius with $R_{max}$=1, both variants give the same results because neither B nor C is eligible.

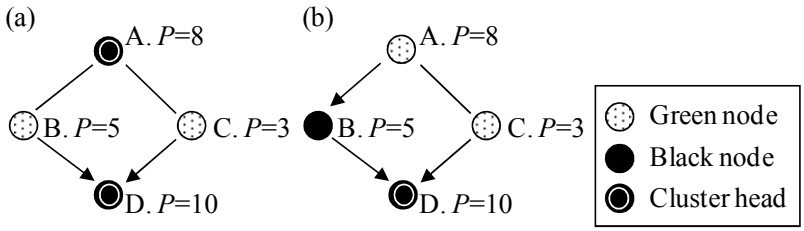


**Figure 5.3. Difference between (a) variant 1 and (b) variant 2.**

We further examine a strategy for improving stability by avoiding moving a node between two clusters. In this *Keep-Cluster strategy*, a node $v$ does not include into $E(v)$ any neighbor $u$ for which $\pi(u) \neq \pi(v)$ unless $E(v)$ otherwise would be empty.

### 5.7.4 Connecting algorithm

The above clustering algorithm divides the network into clusters so that every node belongs to exactly one cluster. Within each cluster, the black nodes form a backbone. To connect clusters together and to form a backbone spanning the whole network, some nodes operate as bridges. For each neighboring cluster, one node is selected as a bridge to that cluster. The bridge is selected so that it is the node that connects the cluster-heads of the two clusters with the minimum distance. In case of equal distances, the node with the highest preference is selected.

Since the Hello message contains the field $\pi(v)$, a node can detect neighbors belonging to a different cluster. For each neighboring cluster $\pi(v)$, an entry $(\pi(v), hdist(v)+1, P(v), v)$ is inserted into the local routing table. As $\pi(v)$ is the address of the cluster-head, this entry thus indicates a route to the cluster-head in the given cluster. The distance in the routing table indicates the distance to this cluster-head. In case several entries for the same cluster $\pi(v)$ are available, the one with the lowest $hdist(v)$ is selected, and in case of a draw, the entry with the highest $P(v)$ is selected.

The local routing table is transported toward the cluster-head using the $dv(v)$ field of the Hello protocol. When a Hello message is received from a dominatee $v$, the entries of $dv(v)$ are copied into the local routing table with the distance increased by one. In case several entries for the same neighboring cluster are available, the one with the lowest distance (or highest preference) is selected. The $dv(v)$ field of a Hello message sent by a node other than a dominatee is ignored.

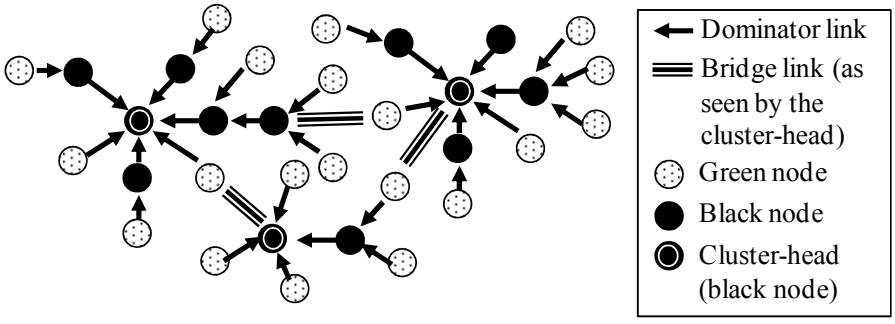


**Figure 5.4: Example of three connected clusters.**

As information is passed in only one direction, no loops can appear. The cluster-head retrieves a routing table with one entry per neighboring cluster. These are not necessarily the same entries as further down a branch; a message can follow a shorter route to a neighboring cluster if such a route is known. The routing tables are used for implementing

different routing models as described in Section 5.7.6. Figure 5.4 shows an example topology created by the proposed algorithm. A node connecting a cluster to a neighboring cluster is called a bridge node, and the link between the bridge node and the corresponding bridge node in the opposite cluster is called a bridge link.

### 5.7.5 Simulation

The proposed clustering scheme is simulated in order to verify its correct operation, to compare the different variants, and to evaluate the properties of the clustering created by the scheme.

#### Simulation setup

The $N$ nodes are initially uniformly randomly distributed in the 1000 x 1000 m roaming area. The random waypoint [GP02] mobility model is used. A node selects a uniformly random speed between 1 m/s and $v_{max}$ and a random destination within the roaming area and moves toward this destination until it is reached. Then the node pauses for a time uniformly random between 0 s and $T_{wait}$, whereafter it selects the following destination.

The simulation includes only the network layer as the physical layer is assumed to operate on a much faster time scale. A message sent by a node is received correctly by all nodes within a distance of $Z$. Hello messages are sent at $T_{hello}$=1 s intervals. A node is considered as a neighbor to all nodes that have received its Hello messages within a $T_{expire}$=2 s interval. Each node has a fixed preference value. This represents the situation, where the resources change slowly compared to the simulation time.

Each scenario is simulated $N_{sim}$=5 times, each run lasting for $T_{sim}$=600 s, and the results are averaged. The network state is sampled at 1000 evenly spaced instances and the results are averaged. Unless otherwise mentioned, the default parameter values in Table 5.3 are used.

In the simulations, we measure the percentage of nodes with different colors and roles. We also measure the stability of each role. Stability is measured using the average interval between color changes, cluster changes and head-selections. The average interval is calculated as

$$\overline{T} = \frac{NT_{sim}}{N_e}, \tag{5.7}$$

where $N_e$ is the measured number of event, $N$ is the number of nodes and $T_{sim}$ is the simulation time.

**Table 5.3. Simulation parameters and their default values.**

| Parameter | Symbol | Default value | Description |
|---|---|---|---|
| Number of nodes | $N$ | 100 | The number of nodes in the network. |
| Radio range | $Z$ | 300 m | The maximum distance between two nodes considered neighbors. |
| Network area | $A$ | 1000 x 1000 m | The size of the area within which nodes are allowed to move. |
| Cluster radius limitation | $R_{max}$ | 1 | The maximum number of links between a node and its cluster-head. |
| Hello interval | $T_{hello}$ | 1 s | The interval between two consecutive Hello messages sent by a node. |
| Hello timeout | $T_{expire}$ | 2 s | The time after which an entry for a neighbor is removed if no Hello message has been received. |
| Maximum speed | $v_{max}$ | 10 m/s | Nodes move with a speed uniformly random between 0 and $v_{max.}$ |
| Maximum waiting time | $T_{wait}$ | 100 s | Nodes wait a time uniformly random between 0 and $T_{wait}$ before choosing the following random destination. |
| Simulation time | $T_{sim}$ | 600 s | The time a simulated scenario is run. |
| Scenario repetitions | $N_{sim}$ | 5 | The number of times a simulated scenario is repeated. |

## Simulation results

In Section 5.7.3 we presented two variants of the clustering algorithm. In the first variant, a node can select itself as its dominator if it has the locally highest (color, preference) rating. In the second variant, the node can select itself only if it has one or several dominatees. Recall, that selecting itself as a dominator implies becoming a cluster-head. We first examine the difference between these variants. Figure 5.5 shows the percentage of black nodes and cluster-heads in both variants under an increasing $R_{max}$. Note that the number of cluster-heads equals the number of clusters as each cluster contains exactly one cluster-head. The difference between the variants is minimal. Figure 5.6 shows the stability, i.e. the average interval between events. In the second variant, the stability of cluster-heads is higher, and therefore we choose to use the second variant in all the following simulations.

In the above simulation we varied the cluster radius limit $R_{max}$, which indirectly controls the maximum cluster size. The largest difference is between the values $R_{max}$=1 and $R_{max}$=2. In the case $R_{max}$=1 all black nodes are cluster-heads, and the curves consequently coincide. For $R_{max}$>1 the number of clusters is halved and more black nodes are required. The stability increases with the increased cluster size. A node can keep its dominator for a longer time as the limitation does not force it to change.

However, for $R_{max}$>4 the stability drops as mobility causes long branches to break more frequently. Increasing the maximum cluster size above $R_{max}$>2 does not increase the number of black nodes or clusters. It rather increases the variance in the cluster size, whereas the largest clusters grow at the same time as the percentage of clusters with a single black node increases.

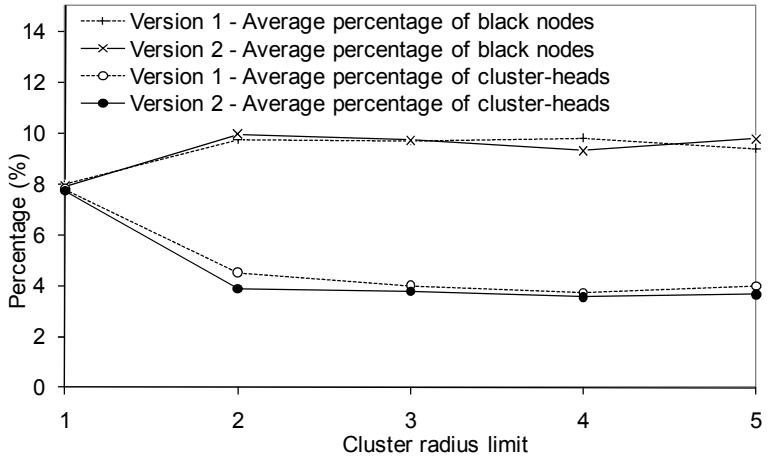


**Figure 5.5: Node types in comparison of algorithm versions.**

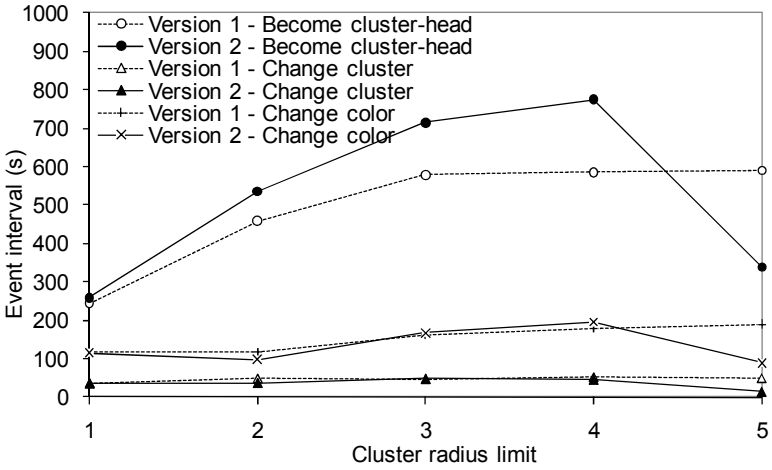


**Figure 5.6: Stability in comparison of algorithm versions.**

Testing the Keep-cluster strategy reveals that it is not practically useful. While it correctly decreases the interval between cluster changes, as shown in Figure 5.8, it performs worse in all other aspects. Especially remarkable is the high percentage of black nodes, shown in Figure 5.7. This can be explained with a higher probability of selecting a low-preference node as dominators: a green neighbor becomes a black node when it is selected as dominator, even though a black node (of another cluster) is in the neighborhood. The keep-cluster strategy also decreases the stability of cluster-heads. We do not use this strategy in further simulations.

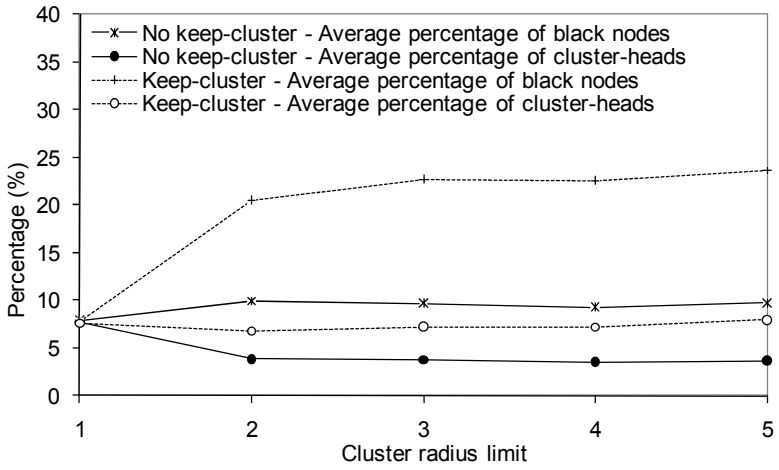


**Figure 5.7: Node types in Keep-Cluster strategy.**

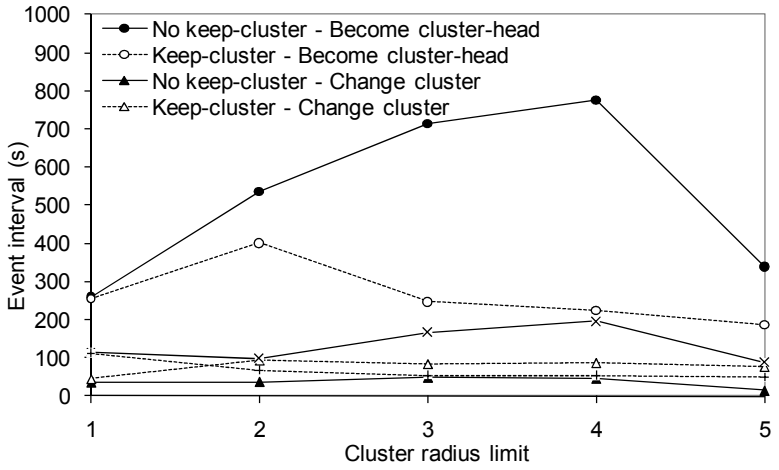


**Figure 5.8: Stability in Keep-Cluster strategy.**

The advantage of clustering is expected to increase in a network with a high density of nodes. As a measurement of density, we use the average degree of a node, i.e. the number of nodes within the radio-range of the node. The expected average node degree is calculated as

$$D = \frac{N}{A}\pi Z^2 \qquad (5.8)$$

.

The density can be varied by either varying the radio-range or the number of nodes in a network of a fixed size. In this work we use the former approach; simulations using the latter approach can be found in [BKC05].

Figure 5.9 shows the percentage of different node types under an average degree varying between 0.39 and 19.24. The degree is accomplished by varying the radio range between $Z$=50 m and $Z$=350 m. The number of nodes is $N$=50 and the maximum speed $v_{max}$=20 m/s. In this and the following simulations $R_{max}$=1. When the density is low ($Z \leq 150$ m), the large percentage of white nodes indicates that many

nodes are isolated and the network is disconnected. The fraction of black nodes is less than 30% in most connected networks and less than 20% in dense networks. About half of the black nodes are cluster-heads in dense networks. A typical cluster contains one cluster-head, one other black node and eight green nodes when $D$=10. Figure 5.10 shows an example of the clusters generated.

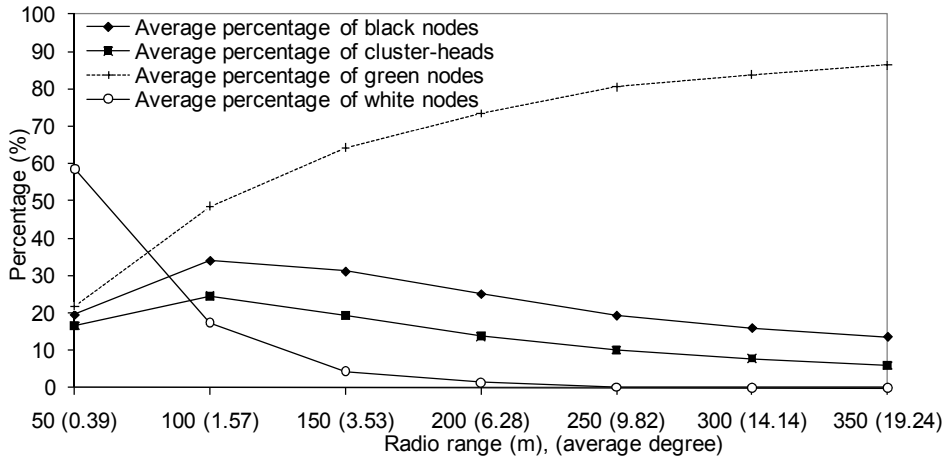


**Figure 5.9: Node types in networks of varying density.**

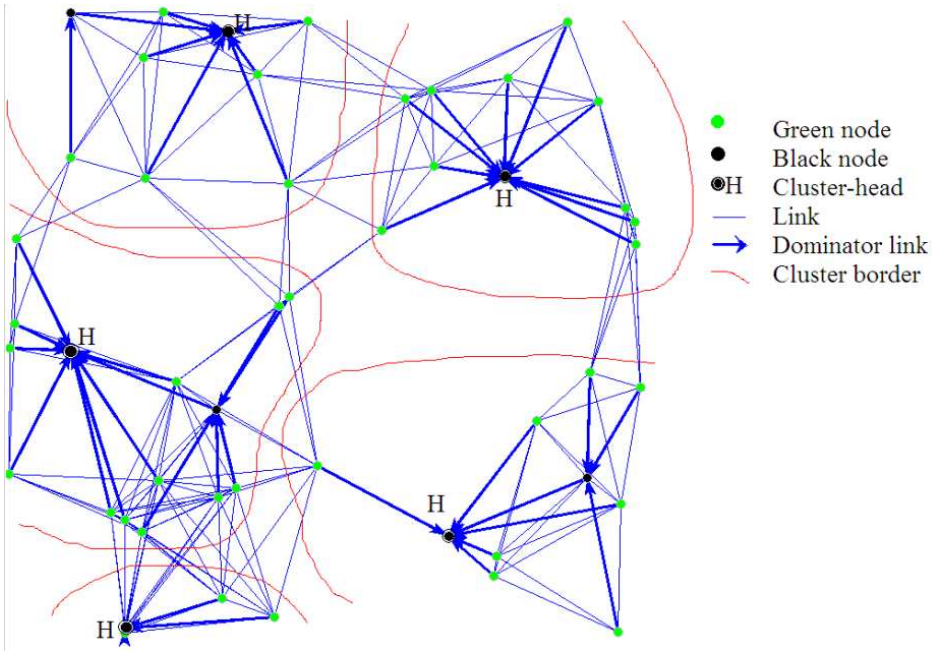


**Figure 5.10: Example screenshot with added cluster borders.**

Figure 5.11 shows that the stability increases as the network becomes dense. Black nodes are more stable than other nodes, and cluster-heads are more stable than other black nodes. A node that has become a cluster-head remains a cluster-head for a fairly long time despite the changing topology. Thus, the cluster-head is suitable for storing index data and maintaining routing responsibility. Figure 5.12 shows that the average

cluster is slightly larger than the average number of neighbors in the neighbor information table (NIT). As the density increases, this difference becomes smaller. The average size of the distance vector (DV) table of the nodes is fairly constant as the number of neighbor clusters remains the same.

Mobility does not affect the distribution of node types, as shown in Figure 5.13. In this simulation, $N$=50 nodes with a fixed radio range $Z$=200 move with a uniformly random speed between 0 m/s and the maximum speed $v_{max}$, which is varied. As expected, the stability shown in Figure 5.14 decreases with increasing mobility.

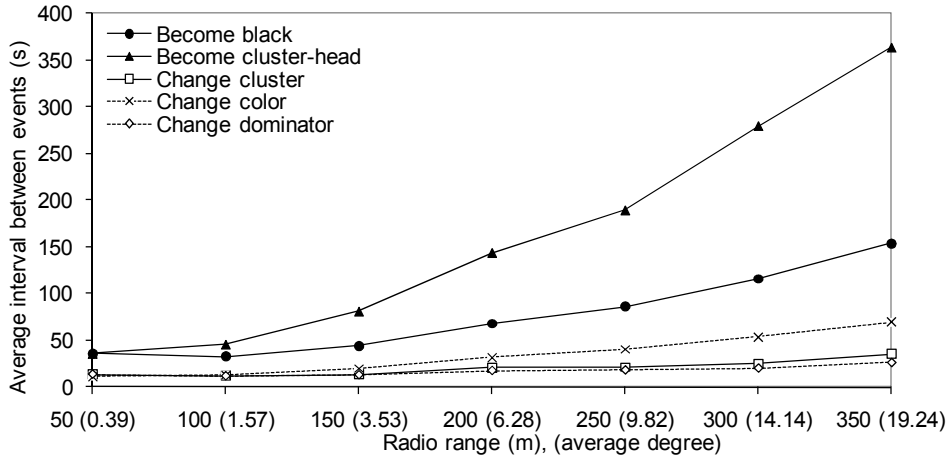


**Figure 5.11: Stability in networks of varying density.**

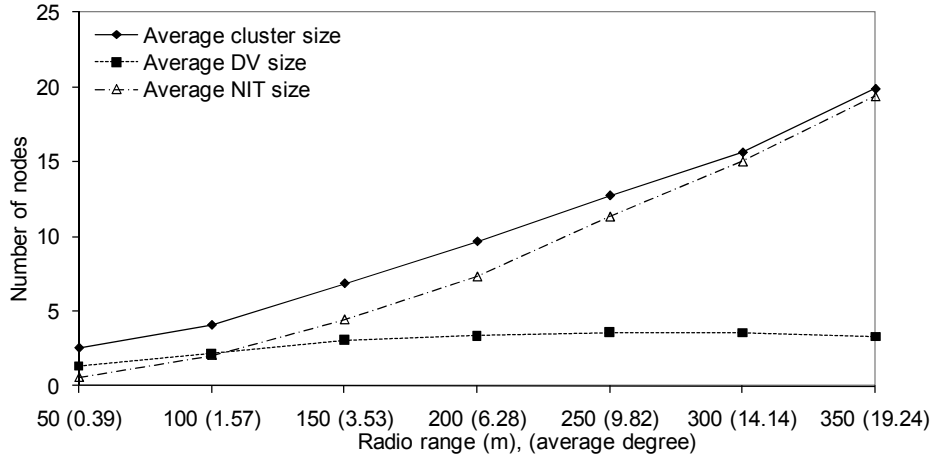


**Figure 5.12: Average cluster size and size of the DV and NIT tables.**

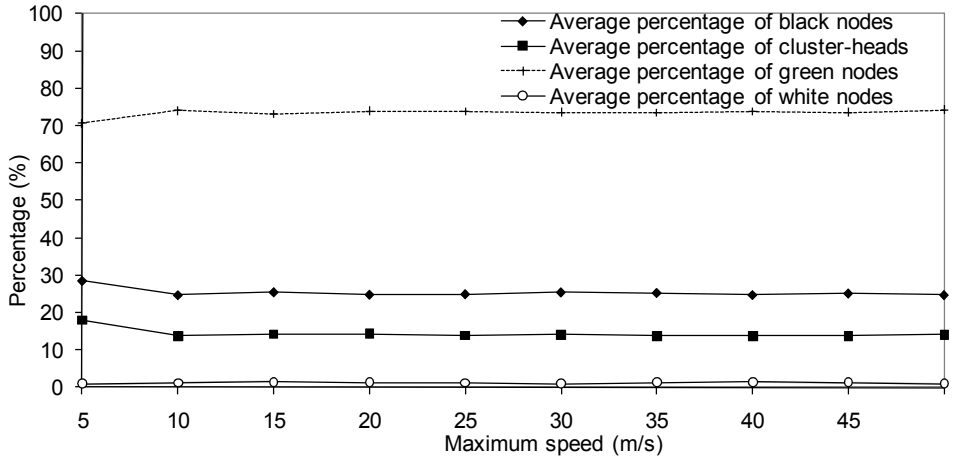


**Figure 5.13: Node types under varying mobility.**

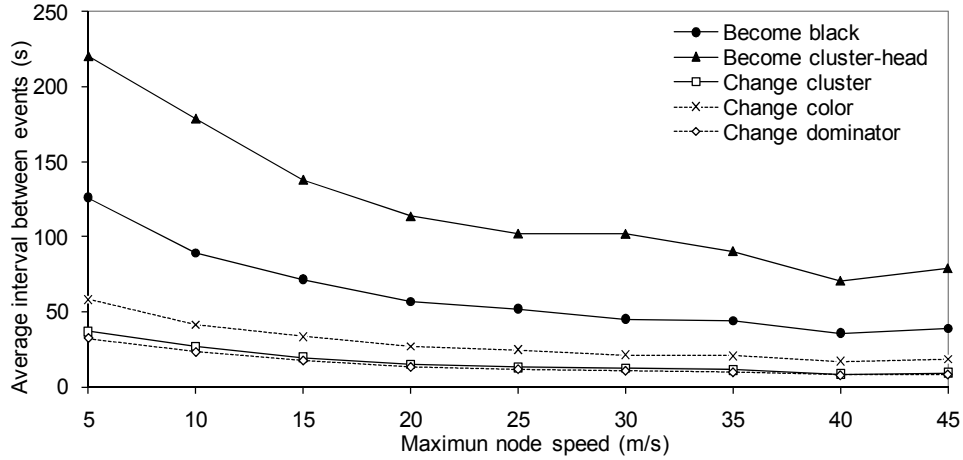


**Figure 5.14: Stability under varying mobility.**

### 5.7.6 Routing and service discovery based on clustering

The black nodes form a CDS within a cluster. Together with the bridge nodes, the black nodes form a CDS spanning the whole network. Broadcasting a message along the backbone reaches all nodes in the network. This backbone broadcast can replace flooding in routing protocols to minimize overhead. A backbone broadcast in a link-state routing protocol provides the same routes as in normal flooding. Using a distance vector protocol or a reactive protocol results in routes following the backbone instead of the shortest routes. The reason is that a route generated by a distance vector routing protocol follows the route that the update packet has traversed. This may be a desired property as the backbone nodes have a higher capacity and stability.

Alternatively, a routing approach similar to [DSB97] can be used. In this approach, cluster-level source routing is used. The packet is routed by a bridge node to the bridge node that connects the current cluster to the following cluster on the route. Information about the bridges and neighboring clusters is proactively distributed by the cluster-head to all

bridges in the cluster. Alternatively, the bridge queries the cluster-head for the information. In order to form the source route, the cluster of the destination node and the topology of clusters must be known. This information is proactively maintained. The proactive protocol is controlled by the cluster-heads. Each node sends its link state updates to the cluster-head, which determines which updates to distribute along each sub-tree of the backbone.

In addition to traditional unicast and broadcast, we propose in [BKC05] a set of routing models based on the cluster structure. The most important routing models and their example use cases are summarized in Table 5.4 and presented in the following.

*Local head unicast* is implemented by repeatedly forwarding a message to the dominator until the cluster-head is reached. Unicast usually requires a return path. The return path can be stored as state information in the intermediate nodes as the message is forwarded.

In a *head broadcast*, the message is forwarded to all cluster-heads in the network. It utilizes the fact that a cluster-head has routes to all neighboring clusters and their cluster-heads. The message is first forwarded to the local cluster-head using repeated forwarding to the dominator. The message is then flooded at the cluster-level until it has reached all cluster-heads. In this flooding process, a cluster-head must remember the identity of a packet that has been seen earlier. Alternatively, a list of the traversed clusters can be included in the message. The list contains the addresses of the corresponding cluster-heads. A bridge node can detect and discard a packet that is entering the same cluster again as it recognizes its own cluster-head's address in the list.

In a *local backbone broadcast*, a message is forwarded by all black nodes in a cluster. Each black node repeats a message that (1) has not earlier been received, and (2) is received from a node in the same cluster.

In a *backbone broadcast*, a message is forwarded by all black nodes and all bridge nodes. Each black node and bridge node repeats a message that has not earlier been received.

A backbone broadcast can generally replace flooding both in reactive and proactive routing protocols. Combining a local backbone broadcast and head broadcast allows using two hierarchical levels in routing and service discovery. For example, a local backbone broadcast can be used in a proactive routing protocol used within a cluster, which is complemented by reactive inter-cluster routing based on a head broadcast. A local head unicast is useful for centralized services within a cluster. For example, a node can reactively query its cluster-head for a route that is based on information collected by a proactive routing protocol running between the cluster-heads.

**Table 5.4. Routing models based on the cluster structure.**

| Name | Reaches cluster-head in local cluster | Reaches cluster-head in other clusters | Reaches all black nodes in local cluster | Reaches all black nodes in other clusters | Example use cases |
|---|---|---|---|---|---|
| Local head unicast | Yes | No | No | No | Local centralized directory |
| Head broadcast | Yes | Yes | No | No | Inter-cluster routing protocols |
| Local backbone broadcast | Yes | No | Yes | No | Intra-cluster routing protocols |
| Backbone broadcast | Yes | Yes | Yes | Yes | In routing protocols for replacing flooding |

In service discovery, the cluster-heads maintain service directories. The service directories benefit from the stability of the cluster-head. The cluster-head is a node with the locally highest capacity and/or lowest mobility. The role of the cluster-head is fairly stable and the routes toward the cluster-head follow nodes with high capacity. We provide a proactive and a reactive approach for service discovery. In the former approach, a service provider publishes its service description to all cluster-heads in the network using a head broadcast. A service requester queries its own cluster-head using a local head unicast when the service is desired. In the reactive approach, the service provider sends its service description to its local cluster-head using a local head unicast. The service description is not forwarded. Instead, a service requester broadcasts the service query is to all cluster-heads with a head broadcast. The search/index ratio determines which approach is most suitable.

## 5.8 Summary

This chapter studied resource discovery in an ad hoc network formed between consumer devices, which are assumed to have highly varying capacity and mobility. We noticed that the least mobile nodes typically have the highest capacity. We presented two approaches for structuring the network into a virtual backbone consisting of high-capacity low-mobility nodes. In our first approach, nodes are divided into two classes (smart nodes and dummy nodes) primarily based on local properties but also observing surrounding devices. A backbone is constructed by interconnecting smart nodes. The weakness of this approach is the lack of control over the density of smart nodes and the resulting need to interconnect islands of smart nodes.

Our second approach is based on clustering. The proposed clustering algorithm ensures that every node has a backbone node as a neighbor and that all clusters are interconnected through bridges. The algorithm continuously maintains the clusters without requiring separate phases for

construction and maintenance. The algorithm is simple and it uses a single periodically sent message. The overhead is fairly low and the Hello message can be piggybacked in other traffic. The load is constant and predictable. Local changes do not lead to global restructuring.

Once the backbone is available, by using either of the algorithms, it can be used for service discovery and routing. The backbone nodes, having a higher capacity and a lower mobility, are suitable for maintaining service directories. Moreover, the overhead of flooding can be reduced by distributing a message over the backbone. For our second algorithm we presented methods for routing to specific types of nodes in the clustered network, for example, to all cluster-heads. The cluster-head stores centralized information about its cluster and each node in the cluster has a route available to the cluster-head. The cluster-head can therefore serve as a service directory or as a connection point between proactive and reactive routing.

# Chapter 6

# Conclusions

In this chapter, we present an overview of the results and summarize the contributions. We suggest topics for future research.

## 6.1 Results and discussion

The choice of architecture for a resource discovery system depends on several requirements set by the application and the network environment:

- *The type of supported queries.* If the application only requires exact-match single-key queries, or if the queries are simple structured combinations of these, the obvious choice is a structured system. If the application must be able to support all types of queries, or if the data types and the query format are not known *a priori*, an unstructured or loosely structured system is a better choice.
- *Determinism.* Many types of distributed applications such as social networks, collaboration and information retrieval applications require deterministic behavior. The system must then be able to locate a requested resource if this resource exists. If determinism is not needed, for instance in ordinary file sharing with well-replicated files, probabilistic search methods are sufficient, or the search scope can be limited using a TTL. However, guarantees for finding a requested file may be desirable in file sharing as well.
- *Load allocation.* The best network-level efficiency is achieved with centralized index nodes. Therefore, we recommend as high a degree of centralization as feasible. However, the centralized nodes need a high capacity. Furthermore, the users maintaining the centralized nodes need a motive for providing more capacity to the system than other users do. Such scenarios appear when an external service provider offers a commercial service. When all nodes have similar capacity and are maintained by ordinary users, the load must be evenly distributed between the participating nodes.

- *Topological restrictions.* In some situations the overlay topology is determined or restricted by external factors. For example, an inability to maintain links between all participating parties limits the usability of architectures likes PIC and PSC. In a social network, the users' social contacts form a topology, which can be used as the overlay topology directly. In ad-hoc networks, it is preferred to use an overlay topology that closely follows the physical topology.

In this work, we have focused on architectures that are deterministic and support complex queries. Structured systems have not been covered by this work. When nodes with a higher capacity are available, we have taken a more centralized approach while otherwise the goal has been to distribute the load evenly. In the studied cases, we have encountered various restrictions regarding the overlay topology. These have affected the architectural choices.

Most of today's architectures are reactive. Indexing is often limited to a small set of centralized nodes, such as super-nodes, which must handle most of the load. In several situations, it is desirable that all nodes participate in indexing, whereas the load is equally divided between the nodes. Such uniform indexing architectures are still rare. The few existing architectures include PIC and PSC. While indexing lowers the search costs, it introduces indexing costs. So far, not much research has been spent on examining how to balance between indexing and searching, i.e. between proactive and reactive operations.

In Section 2.5 we proposed the Search/Index Space model. This model allows studying the tradeoff between reactive and proactive traffic. With the Search/Index Space model we can show that in many practical scenarios, the lowest total traffic is obtained when a certain degree of indexing is used. This optimal balance between searching and indexing depends on the ratio $r$ between the frequencies of search messages and index updates. According to (2.34), fully reactive algorithms, despite their popularity today, are only optimal for the marginal case when $r \leq \Omega_i/N\Omega_s$ in an $N$-node network. Here, $\Omega_i$ denotes the index distribution overhead, and $\Omega_s$ denotes the search distribution overhead. In typical flooding-based search algorithms $\Omega_s \approx 5$. Index distribution algorithms like Direct Index achieve good efficiency with $\Omega_i \approx 1$. Therefore, a traditional flooding-based architecture such as Gnutella is feasible for $r \leq 1/5N$. At the opposite extreme, a fully proactive architecture, such as Direct Index, is optimal when $r \geq N\Omega_i/\Omega_s$ according to (2.35). Using a value of $\Omega_i \approx 1$, this architecture is optimal when $r \geq N/5$. Contrary to flooding, the overhead in Direct Index is independent of the node degree, which is beneficial in cases such as social networks.

In situations other than in these extremes, i.e. for $\Omega_i/N\Omega_s \leq r \leq N\Omega_i/\Omega_s$, it is optimal to combine proactive and reactive operations. Such hybrid architectures are particularly beneficial in large networks. PIC and PSC are hybrid architectures whose traffic can be minimized with an optimally selected cluster size. The optimal cluster size calculated by the Search/Index Space model is given by (2.32) for PSC and (2.33) for PIC. While being efficient, these architectures suffer from two major disadvantages: (1) full connectivity between clusters is required, and (2)

the topology is difficult to construct and maintain in a practical network so that optimality is preserved.

The first disadvantage is tackled by the IPIC architecture proposed in Section 3.4. For this architecture two new deterministic search methods were provided: Stack-Based Random Walk (SBRW) and Replicating Stack-Based Random Walk (RSBRW). The IPIC architecture does not require full connectivity between clusters. It avoids the exponential growth of state information when the network size grows. The drawback of IPIC is a slight performance degradation: depending on whether SBRW or RSBRW is used, either the message overhead or the delay increases. For PSC, we have not found a feasible alternative that would allow an arbitrary cluster topology.

While clustered architectures such as PIC, PSC and IPIC provide good performance, their adaptability to changing network conditions is low. Clusters are relatively static. To solve this problem, we developed an approach with overlapping "clusters", called zones. The Zone Indexing architecture presented in Section 3.5 provides an overhead similar to PIC and PSC with controlled delay bounds. It is based on a mindset similar to the Zone Routing Protocol, which combines proactive and reactive routing in ad hoc networks. As Zone Indexing automatically measures the traffic, it avoids the problem of approximating the search/index ratio $r$. The complexity of Zone Indexing is higher than the complexity of PIC and PSC due to the zone size control but, on the other hand, a corresponding mechanism for adjusting cluster size in PIC and PSC would increase the complexity of these architectures as well. As Zone Indexing can use any ring topology, it could be used to supplement Chord with complex queries using the same ring.

We found that the optimal scalability of a uniform system supporting complex queries is $O(\sqrt{N})$. This can be achieved with PIC, PSC and Zone Indexing. As today's commonly used unstructured systems provide a scalability of $O(N)$, the improvement is significant. However, the ability to support complex queries comes at the cost of not being able to achieve the $O(\log N)$ scalability of structured systems.

Our questionnaire studies in Section 4.4 provided an overview of the behavior and opinions of users regarding peer-to-peer services. One of the most important observations was that users want to limit the access to given types of resources to people they know. The more personal the content is, the lower the number of people that are allowed to access the content. A similar tendency is observed in searching: people are more interested in accessing files made by the people they know. The cost of sharing as well as legal restrictions may reduce the willingness to share also content that is less personal and commonly available. The studies support our assumption that there is a need for group support in resource sharing applications. Group support and access control are not found in today's peer-to-peer systems, which publish resources to everybody. Group support is motivated especially in the mobile environment, as a major portion of the resources created on phones are personal in nature and the cost of sharing (measured in bandwidth, battery consumption and money) is higher.

Support for groups can be implemented in both an operator-controlled and a completely distributed environment. The centralized control in the former allows several models for group managements to be implemented. However, in a completely distributed scenario, the lack of central control complicates role-based and collaborative group management. For that reason we developed a distributed group management method based on the user's contacts. The group is implicitly formed by merging each member's local view.

In an operator controlled environment, it is advisable to transfer the bulk of the load to the fixed network. The resource sharing service is supported by fixed servers that are interconnected with each other. The architecture is consequentially hierarchical with the mobile device connected to a server as a client. Practically any peer-to-peer architecture can be used to interconnect servers. The simplest but least efficient topology is a random topology, leading to a semi-centralized architecture. A loosely structured topology, such as PIC, PSC, or Zone Indexing provides better performance and a lower overhead. However, in a commercial environment, the technical performance is not the only criteria. The architecture must be compatible with the inter-operator agreements, the charging schemes, and the control requirements of the operator, and guarantee non-disclosure of customer and network information. The architecture must ensure that the indices are contained within the operator's network. Because of these demands, we propose using IPIC in the upper hierarchical layer.

When operator support is unavailable or not desired, a peer-to-peer network can be formed between the mobile devices directly. As the resources are limited, centralized points among these devices must be avoided. Architectures based on flooding are robust and resilient albeit very inefficient. The PIC and PSC architectures provide good performance but are not suitable because of their lack of an automatic cluster maintenance mechanism. On the other hand, Zone Indexing has been designed to minimize the overhead and to automatically optimize the performance while maintaining the topology.

An overlay built on top of a social network provides an interesting alternative. One approach is to use a completely proactive architecture, which is feasible when searches are frequent compared to index updates. We presented Direct Index – a proactive architecture suitable for groups of moderate size. The update load of Direct Index is proportional to the group size and is independent of the node degree, which is beneficial in social networks with a typically high node degree. The proactive approach generates a minimal search load since searches are local and need only a single roundtrip for confirming the results. It allows the use of policies to control distribution and access.

As SIP is supported natively in IMS networks, we consider using SIP for resource discovery. Particularly valuable in current networks is the ability to route peer-to-peer signaling via SIP servers in the fixed network, allowing firewalls and NATs to be traversed. The extensions needed for index publication and searching are easy to implement. We presented two different approaches. However, the limitations found in

current SIP stacks prevent implementing some functions in a way that corresponds to the ideology of the protocol. Therefore workarounds were required in the prototype implementations.

The prototype implementations showed that the capacity and the network capabilities of modern cellular phones are adequate for peer-to-peer software. The user perceived experience is mainly affected by the user interface and the network constraints. The bandwidth currently restricts the downloading of large media files, but is adequate for exchanging songs, photos and text files. Large media files can be accessed using streaming, allowing the file to be viewed before it is completely downloaded. These limitations are not specific to peer-to-peer. Provided that an appropriate architecture is selected, current cellular phones do not set noticeable technical limitations on resource discovery.

The studied ad hoc networks are assumed to be formed by wireless consumer devices with varying capacity and mobility. We utilize the capacity heterogeneity in making service discovery and routing efficient. We observed an inverse relationship between mobility and capacity, allowing us to abstract the capacity and the mobility into a preference value. Based on this observation we select the devices with the highest capacity and the lowest mobility to form a virtual backbone. This backbone assists the other nodes by performing heavier tasks and storing information.

We took two different approaches in categorizing the devices and forming the backbone. In the first approach, devices are divided into two classes (smart nodes and dummy nodes) according to independent choices by the nodes. Smart nodes connect to neighboring smart nodes. In this approach, it is difficult to control the density of smart nodes. Virtual links are therefore necessary to interconnect islands of smart nodes when the density is low. It is also difficult to guarantee that the backbone remains connected.

The second approach creates the backbone using clustering. It ensures that every node has a backbone node as a neighbor and that all clusters are interconnected through bridges. Nodes are categorized into three classes: cluster-heads, backbone nodes (black nodes) and ordinary nodes (green nodes). In many situations, it is worth sacrificing some extra forwarding hops in order to concentrate the load on high-capacity nodes. Therefore, the resulting clusters can have a diameter larger than two hops when it is justified by the location of high-capacity nodes. The number of clusters is consequently smaller than in many similar algorithms. The maintenance of the clustering is continuous without separate phases for construction and maintenance. The algorithm uses a single periodically sent message that can be piggybacked in other messages.

Constructing a service discovery system for an ad hoc network requires (1) allocating service directories to powerful nodes and (2) connecting service directories together with an overlay. In our approach, service directories are maintained by the backbone nodes (smart nodes or cluster-heads), which have a higher capacity and better stability. Any overlay could be used to connect the service directories. However, overlays designed for fixed networks are ill-suited for ad hoc networks as

the mismatch between the physical network and the overlay topology has more severe consequences in the ad hoc network. A virtual backbone following the physical topology provides a more efficient overlay. Both proposed algorithms connect service directories via high-capacity nodes.

We argue that in most situations it makes sense to combine proactive and reactive routing. The different routing approaches operate on different hierarchical layers. A virtual backbone allows dividing nodes into two layers: the backbone nodes on the higher layer and the other nodes on the lower layer. For clustering the hierarchical division is even more evident: the intra-cluster operation on the lower layer and the inter-cluster operation on the higher layer. A virtual backbone and clustering can be seen as dual problems: a virtual backbone connects clusters together, and clustering allocates a backbone node to every non-backbone node. If a backbone of stable nodes is available, a proactive routing protocol can be run globally over this backbone. The non-backbone nodes use a reactive routing protocol locally to access the proactive routing information of a nearby backbone node. The roles can also be exchanged: the proactive protocol operates within a cluster and the reactive protocol uses the backbone for remote destinations. This approach is rather similar to ZRP, but with disjoint clusters.

## 6.2 Summary of contribution

This work has studied resource discovery architectures where indexing is utilized to reduce the search overhead. The architectures are limited to those supporting complex queries and deterministic search.

The work has focused on architectures providing uniformly distributed load. To model such architectures we proposed the Search/Index Space model. We applied the model to determine the optimal level of index distribution. This includes determining the optimal cluster size in PIC and PSC. The model was also used to determine when a completely reactive or a completely proactive solution is optimal. To support the design and analysis, the work specified a set of desired properties and metrics and provided modeling of update frequencies, overhead and centralization. A new simulator called PONGsim was developed.

This work proposes three novel architectures. Firstly, we extended the PIC architecture by allowing clusters to be connected with an arbitrary cluster topology. This requires a new search algorithm, which we call Stack-based Random Walk (SBRW), operating on the cluster-level. To reduce the delay associated with random walks, we proposed replication of the walker in the Replicating Stack-based Random Walk (RSBRW). Secondly, we proposed a new architecture called Zone Indexing, which allows the balance between reactive and proactive operations to be dynamically maintained at an optimal level. The network allows nodes to join and leave without the need to restructure the network. We proposed an algorithm to automatically maintain an optimal zone size. Shortcuts were introduced in order to reduce delay. We proposed a method for overlay maintenance under heavy churn. Thirdly,

we proposed a fully proactive architecture, called Direct Index. This architecture is more efficient than flooding due to the proactively available information. It is suitable for specific scenarios, such as a decentralized social network.

We studied how resource sharing systems can be implemented in cellular networks. We performed four user surveys to examine the feasibility of resource sharing in mobile networks from the user's perspective. The user studies examined questions such as the willingness to distribute and obtain different types of resources. Access control turned out to be one of the most requested features in a resource sharing network. We studied the application of various architectures to an operator controlled resource sharing service as well as a completely distributed resource sharing platform. We suggested how access control and groups can be implemented. In particular, we proposed a fully decentralized system supporting groups for controlling resource access. We developed two schemes for using SIP-based signaling in peer-to-peer systems. Finally, we examined the technical limitations of the device and the network. By developing and testing prototypes we proved that the concept works technically. The tests included both the operator-controlled and the fully distributed model. It was observed that NATs and firewalls are the main challenges in a practical deployment.

We studied ad-hoc networks formed of consumer devices with highly varying capacity and mobility. We proposed categorizing devices according to their capacity and mobility, and forming a virtual backbone of the devices with high capacity and low mobility. We proposed two methods for node categorization and backbone construction. The first interconnects groups of high-capacity nodes via low-capacity nodes. The second forms clusters centered around high-capacity nodes and connects these clusters together through bridges. We presented how the virtual backbone can be used in resource discovery and routing.

The main contributions are the following:

1. the Search/Index Space model,
2. the IPIC architecture and the related SBRW and RSBRW search algorithms,
3. the Zone Indexing architecture and the related zone adjustment, delay control and topology maintenance algorithms,
4. the Direct Index architecture,
5. results of user studies based on surveys,
6. access control and group management methods,
7. deployment scenarios for resource discovery in a cellular network,
8. signaling schemes based on SIP,
9. prototype implementations of resource discovery systems in cellular networks,
10. two algorithms for creating overlays in ad hoc networks, and
11. a simulator for evaluating the performance of overlay networks.

## 6.3 Future research

The work has focused on deterministic resource discovery supporting complex queries. This covers only a part of the possible architectures. Where determinism is not needed, a large range of probabilistic systems can be used. Especially determining the suitable amount of replication becomes interesting in such systems. For systems not requiring complex queries, structured systems are feasible. A lot of research on structured systems is currently ongoing. Especially the support of certain types of complex queries in structured systems is a promising research area. A large fraction of the resource discovery systems are likely to be based on structured systems. However, we find it improbable that every possible system will be structured. Similarly, in computer programs data structures such as trees and hash tables have not replaced the linked list and the array. Instead, it is feasible to combine structured and unstructured components, which opens up new research areas.

Between structured and unstructured systems, there is a large gap that we have been examining. Our research looks at only a part of the different solutions possible by adding some structure to an unstructured system without limiting searches to exact-match single-key queries. As the Zone Indexing architecture shows, we can rearrange links in new ways to obtain systems with interesting properties, especially in terms of handing dynamic node membership. Considering churn is important in the system design as the churn rate largely determines the index update interval and the overhead of the structure maintenance.

An important design parameter is the degree of centralization. In particular, if the devices have a widely varying capacity, how can load be allocated according to the available capacity? This becomes challenging considering both the indexing and the search load.

We fundamentally base the balance between proactive and reactive operations on the search/index ratio $r$. However, to better be able to analyze practical systems, we need to know the search/index ratios of various existing systems. Finding these is challenging because the search frequency varies widely between applications and searches are performed in bursts. The index frequency depends on several system parameters, most importantly on the churn rate of the system but also on the user behavior. The possibility to advertise several resources in the same message and the user adding multiple resources (e.g. a directory of files) simultaneously complicates the evaluation. In systems such as PIC and PSC, the search/index ratio must be known in order to divide nodes into clusters optimally. The inability of determining this ratio may be a obstacle in motivating the development of new hybrid proactive-reactive architectures. Zone Indexing solves the problem by automatically adjusting to the current ratio.

So far we have not made an effort to model the performance of IPIC using the Search/Index space model. The modeling of IPIC-SBRW is rather straightforward while the modeling of the RSBRW search method becomes more challenging due to the replications.

We provided an algorithm for adjusting the zone size in a Zone Indexing network. The provided algorithm is sufficient for moderate changes in zone size but slow if all nodes join simultaneously. The algorithm can be improved using control theory. It is however important to reduce oscillations and quick variations as these create redundant index information. The influence of replication in a Zone Indexing network has not been examined. Replication can allow reduced search delays and overhead as a search can be interrupted when a sufficient number of resources have been found. Also expanding ring searching can be applied to Zone Indexing. Generally, the effect of replication could be analyzed in the general case using extensions to the Search/Index Space model.

The Zone Indexing architecture has been evaluated using simulations. In order to evaluate the architecture in practice, we aim to implement a large-scale prototype running, for example, in PlanetLab [PlanetLab]. Integration with an existing file sharing application would allow evaluating the real benefit of hybrid systems, as the search/index ratio is based on actual user behavior. Similarly, we aim to test the Direct Index approach with users in order to evaluate the user experience of group based sharing and to examine the group sizes and social topologies forming the base for sharing. For the operator-controlled scenarios, it would be interesting to obtain input from operators. In general, for resource sharing applications in the cellular network it is necessary to evaluate the commercial possibilities and obstacles. On the application level, the prototyping of distributed versions of common centralized applications, such as social network (e.g. similar to Facebook) and media sharing (e.g. similar to YouTube) reveals the feasibility both from technical and from commercial perspective.

Routing protocols, as long as they have been automatic, have been proactive. Only with the introduction of ad hoc networks, the reactive approach has become interesting. While considerable efforts have been spent on developing proactive and reactive routing protocols, only a few approaches have tried to combine the two concepts. Even less research is spent on analyzing the optimal balance between proactive and reactive operations. We presented a discussion on applying the Search/Index Space model to ad hoc networks. The optimal balance between reactive and proactive routing in ad hoc networks could be examined in detail with a similar model. This model may become complex or inaccurate as it is dependent on the physical topology and on the used routing methods.

While the large-scale combination of reactive and proactive operations in both routing and service discovery is still uncommon, it seems that both areas could benefit largely from a hybrid approach. The same kind of problem can be found in other areas, such as in publish/subscribe systems, web-caching and content distribution networks. The techniques of combining proactive and reactive operations may find fruitful applications in several systems outside routing and service discovery.

# References


[AHP02] A. Arora, C. Haywood, K. S. Pabla, "JXTA for J2ME - Extending the Reach of Wireless with JXTA Technology," in *JavaOne Conf.*, 2002.

[AJ07] F. Audet, C. Jennings, *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP*, RFC 4787, January 2007.

[ALP+01] L. A. Adamic, R. M. Lukose, A. R. Puniyani, B. A. Huberman, "Search in Power-Law Networks," in *Physical Review E*, vol. 64, issue 4, 2001.

[ANS05] A. Adams, J. Nicholas, W. Siadak. *Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*, RFC 3973, January 2005.

[AP00] A. D. Amis, R. Prakash. "Load-Balancing Clusters in Wireless Ad Hoc Networks," in *Proc. 3rd IEEE Symp. on Application-Specific Systems and Software Engineering Technology (ASSET'00)*, pp. 25-32, March 2000.

[AS04] S. Androutsellis-Theotokis, D. Spinellis, "A survey of peer-to-peer content distribution technologies," in *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 335-371, December 2004.

[AWD04] M. Abolhasan, T. Wysocki, E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," in *Ad Hoc Networks*, vol. 2, 1–22, 2004.

[AX02] A. Andrzejak, Z. Xu, "Scalable, efficient range queries for grid information services," in *Proc. 2nd IEEE Int. Conf. on Peer to Peer Computing*, pp. 33-40, September 2002.

[BA99] A.-L. Barabási, R. Albert, "Emergence of Scaling in Random Networks," in *Science*, vol. 286, no. 5439, pp. 509-512, October 1999.

[Bas99] S. Basagni, "Distributed and mobility-adaptive clustering for multimedia supporting multi-hop wireless networks," in *Vehicular Technology Conf. 1999 (VTC 1999)*, vol. 2, pp. 889-893, 1999.

[BAS+04] A.R. Bharambe, M. Agrawal, S. Seshan, "Mercury: supporting scalable multi-attribute range queries," in *Proc. 2004 conf. on Applications, technologies, architectures, and protocols for computer communication*, pp. 353-366, 2004.

[BCF+03] B. Bakos, G. Csucs, L. Farkas, J. K. Nurminen, "Peer-to-peer protocol evaluation in topologies resembling wireless networks - An experiment with Gnutella query engine," in *Proc. 11th IEEE Int. Conf. on Networks, 2003 (ICON2003)*, pp. 673-680, September-October 2003.

[Bei04] N. Beijar, *Telephony Routing with Support for Number Portability in Interconnected Circuit and Packet Switched Networks*, Licentiate thesis, Helsinki University of Technology, 2004.

[Bei07a] N. Beijar, "Extending Parallel Index Clustering for Multi-Operator Mobile Peer-to-Peer Services," in *The 4th IEEE Int. Conf. on Broadband Communications, Networks, and Systems, IEEE Broadnets 2007*, ISBN 978-1-4244-1432-1, pp. 415-422, Raleigh, North Carolina, USA, 2007.

[Bei07b] N. Beijar, "Index Distribution in a Group-Based Resource Sharing Application," in *The 3rd Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2007)*, ISBN 978-1-4244-1318-8, pp. 345-352, New York, USA, November 2007.

[Bei07c] N. Beijar, *Binary protocol for the Direct Index algorithm*, Technical report, 2007.

[Bei09] N. Beijar, *Introduction to the Python Overlay Network Graphical Simulator (PONGsim)*, Technical report, Helsinki University of Technology, 2009.

[Bei10] N. Beijar, "Zone Indexing: Optimizing the Balance between Searching and Indexing in a Loosely Structured Overlay," in *Computer Networks*, vol. 54, issue 12, pp. 2041-2055, August 2010.

[BFM98] T. Berners-Lee, R.T. Fielding, L. Masinter, *Uniform Resource Identifiers (URI): Generic Syntax*, RFC 2396, August 1998.

[BFN05a] B. Bakos, L. Farkas, J. K. Nurminen, "Phonebook crawler – search engine for mobile P2P social networks," in *IASTED Int. Conf. on Databases and Applications*, Innsbruck, Austria, February 2005.

[BFN05b] B. Bakos, L. Farkas, J. K. Nurminen, "Search engine for phonebook-based smart phone networks," in *Proc. 61st Vehicular Technology Conf. (VTC'2005)*, Stockholm, Sweden, 2005.

[BFN06] B. Bakos, L. Farkas, J. K. Nurminen, "P2P Applications on Smart Phones using Cellular Communications," in *Proc. 2006 IEEE Wireless Communications and Networking Conf. (WCNC 2006)*, Las Vegas, USA, 2006.

[BH09] A. Berl, H. de Meer, “Integration of Mobile Devices into Popular Peer-to-Peer Networks,” in *Proc. 5th Euro-NGI*, 2009.

[BHP+04] D. Bauer, P. Hurley, R. Pletka, M. Waldvogel, “Bringing efficient advanced queries to distributed hash tables,” in *Proc. 29th Annu. IEEE Int. Conf. on Local Computer Networks (LCN’04)*, pp. 6–14, 2004.

[BitTorrent] BitTorrent, http://www.bittorrent.com/.

[BKC05] N. Beijar, R. Kantola, J. Costa-Requena, “A Lightweight Clustering Algorithm for Utilizing Capacity Heterogeneity in Ad Hoc Networks,” in *The 4th Annu. Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2005*, Île de Porquerolles, France, 2005.

[BKC06] N. Beijar, R. Kantola, J. Costa-Requena, “A Lightweight Clustering Algorithm Utilizing Capacity Hereogeneity,” in *Challenges in Ad Hoc Networking*, Edited by K. Al Agha, I. Guérin Lassous, G. Pujolle, ISBN 0-387-31171-8, Springer, USA, 2006.

[BKL01] P. Basu, N. Khan, T. D. C. Little, “A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks,” in *21st Int. Conf. on Distributed Computing Systems Workshops (ICDCSW '01)*, pp. 413-418, April 2001.

[Blo70] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” in *Communications of the ACM,* vol. 13, no. 7, pp. 422-426, 1970.

[Bluetooth] Bluetooth Special Interests Group, https://www.bluetooth.org.

[BM02] A. Z. Broder, M. Mitzenmacher, “Network Application of Bloom Filters: A Survey,” in *Proc. 40th Annu. Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, October 2002.

[BML+05] N. Beijar, M. Matuszewski, J. Lehtinen, T. Hyyryläinen, “Mobile Peer-to-Peer Content Sharing Services in IMS,” in *The Int. Conf. on Telecommunication Systems, Modeling and Analysis 2005, ICTSM2005,* Dallas, Texas, USA, 2005.

[CBK02] J. Costa-Requena, N. Beijar, R. Kantola, “Replication of routing tables for mobility management in ad hoc networks,” in *The 1st Annu. Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2002*), Chia, Sardegna, Italy, 2002.

[CBK04] J. Costa-Requena, N. Beijar, R. Kantola, "Replication of Routing Tables for Mobility Management in Ad Hoc Networks," in *Wireless Networks*, vol. 10, no. 4, pp. 367-375, Kluwer, 2004.

[CDN+05] P. A. Chirita, A. Damian, W. Nejdl, W. Siberski, "Search strategies for scientific collaboration networks," in *Proc. 2005 ACM Workshop on Information Retrieval in Peer-to-Peer Networks*, pp. 33-40, Bremen, Germany, 2005.

[CG02] A. Crespo, H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proc. 22nd Int. Conf. on Distributed Computing Systems*, pp. 23-32, 2002.

[CG03] B. F. Cooper, H. Garcia-Molina, "SIL: Modeling and Measuring Scalable Peer-to-Peer Search Networks," in *Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 2003.

[CG06] B. F. Cooper, H. Garcia-Molina, "SIL: A model for analyzing scalable peer-to-peer search networks," in *Computer Networks*, vol. 50, issue 13, pp. 2380-2400, September 2006.

[CGK+04] J. Costa-Requena, J. Gutiérrez, R. Kantola, J. Creado, N. Beijar, "Network Architecture for Scalable Ad Hoc Networks," in *The 11th Int. Conf. on Telecommunications (ICT2004)*, Fortaleza, Ceará, Brazil, 2004.

[CJ03] T. Clausen, P. Jacquet, *Optimized Link State Routing Protocol (OLSR)*, IETF RFC 3626, October 2003.

[CJY+02] D. Chakraborty, A. Joshi, Y. Yesha, T. Finin, "GSD: a novel group-based service discovery protocol for MANETS," in *4th Int. Workshop on Mobile and Wireless Communications Network*, pp. 140-144, September 2002.

[CK08] S. Cheshire, M. Krochmal, *DNS-Based Service Discovery*, IETF Internet Draft, draft-cheshire-dnsext-dns-sd-05.txt, Work in progress, September 2008.

[CKB05] J. Costa-Requena, R. Kantola, N. Beijar, "Incentive Problem for Ad Hoc Networks Scalability," in *Int. Conf. on Networking and Services (ICNS'05)*, Papeete, Tahiti, French Polynesia, 2005.

[CL02] F. Chung, L. Lu, "The average distances in random graphs with given expected degrees," in *Proc. National Academy of Sciences*, vol. 99, no. 25, pp. 15879-15882, December 2002.

[Clip2] Clip2 Distributed Search Solutions, *The Gnutella protocol specification v0.4*, http://www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf (retrieved 27.8.2010).

[CLG+04] A. Crainiceanu, P. Linga, J. Gehrke, J. Shanmugasundaram, "Querying peer-to-peer networks using P-trees," in *Proc. 7th Int. Workshop on the Web and Databases*, Paris, France, pp. 25-30, 2004.

[CLR+01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press and McGraw-Hill, ISBN 0-262-03293-7, 2001.

[CMJ07] B. Campbell, R. Mahy, C. Jennings, *The Message Session Relay Protocol (MSRP)*, IETF RFC 4975, September 2007.

[Cos07] J. Costa-Requena, *A Hybrid Routing Approach for Ad Hoc Networks*, Doctoral dissertation, ISBN 978-951-22-8910-3, Helsinki University of Technology, Espoo, 2007.

[CRB+03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, "Making Gnutella-like P2P Systems Scalable," in *Proc. 2003 conf. on Applications, technologies, architectures, and protocols for computer communications*, ACM Press, ISBN 1-58113-735-4, pp. 407-418, New York, NY, USA, 2003.

[CRS+02] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, D. Gurle, *Session Initiation Protocol (SIP) Extension for Instant Messaging*, IETF RFC 3428, December 2002.

[CS02] E. Cohen, S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," in *Proc. of SIGCOMM '02,* Pittsburgh, USA, 2002.

[CST00] M. Chatterjee, S. K. Sas, D. Turgut, "An on-demand weighted clustering algorithm (WCA) for ad hoc networks," in *IEEE Global Telecommunications Conference 2000 (GLOBECOM '00)*, vol. 3, pp. 1697-1701, San Francisco, CA, USA, November-December 2000.

[CVK+06] J. Costa-Requena, T. Vadar, R. Kantola, N. Beijar, "AODV-OLSR scalable ad hoc routing proposal," in *1st Int. Symp. on Wireless Pervasive Computing*, 2006.

[DB04] G. Ding, B. Bhargava, "Peer-to-peer file-sharing over mobile ad hoc networks," in *2th IEEE Annu. Conf. on Pervasive Computing and Communications Workshops*, pp. 104-108, Orlando, Florida, March 2004.

[DBB+97] B. Das, V. Bharghavan, "Routing in Ad hoc Networks Using Minimum Connected Dominating Sets," in *IEEE Int. Conf. on Communications (ICC)*, 1997.

[DL04] Q. Deng, H. Lv, "Analyzing Unstructured Peer-to-Peer Search Networks with QIL," in *Proc. 2004 IEEE Int. Conf. on Services Computing (SCC'04)*, 2004.

[DSB97] B. Das, R. Sivakumar, V. Bharghavan, "Routing in ad hoc networks using a spine," in *Proc. 6th Int. Conf. on Computer Communications and Networks, 1997*, pp. 34-39, September 1997.

[Dun] R. Dunlap, *Self-Organizing Parallel Search Clusters*, http://www.cc.gatech.edu/~rocky/docs/splitting_parallel_search_clusters.pdf (retrieved 27.8.2010).

[ENK08] P. Ekler, J. K. Nurminen, A. J. Kiss, "Experiences of implementing BitTorrent on Jave ME platform," in *5th IEEE Consumer Communications and Networking Conf., 2008 (CCNC 2008)*, pp. 1154-1158, Las Vegas, USA, January 2008.

[eMule] Official eMule Homepage, http://www.emule-project.net.

[Facebook] Facebook, http://www.facebook.com/.

[Fan08] N. Fan, *Mobile Peer-to-Peer Simulator*, Master's thesis, University of Oulu, Oulu, March 2008.

[FBG+04] P. A. Felber, E. W. Biersack, L. Garcés-Erice, K.W. Ross, G. Urvoy-Keller, "Data indexing and querying in DHT peer-to-peer networks," in *Proc. 24th IEEE Int. Conf. On Distributed Computing Systems*, pp. 200-208, 2004.

[FCA+02] L. Fan, P. Cao, J. Almeida, A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," in *IEEE/ACM Trans. on Networking*, vol. 8, no. 3, pp. 281-293, 2000.

[Fee99] L. M. Feeney, *A Taxonomy for Routing Protocols in Mobile Ad Hoc Networks*, SICS technical report T99/07, October 1999.

[Flickr] Flickr, http://www.flickr.com/.

[FML+03] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, C. Diot, "Packet-level traffic measurements from the Sprint IP backbone," in *IEEE Network*, vol. 17, no. 6, pp. 6-16, November-December 2003.

[For07] B. Forstner, "Semantic Peer-to-Peer Information Retrieval for Mobile," in *MDD 2007*, Aalborg, Denmark, 2007.

[Freenet] The Freenet Project, http://freenetproject.org/.

[Fring] Fring, http://www.fring.com/.

[GAA03] A. Gupta, D. Agrawal, A. El Abbadi, "Approximate range selection queries in peer-to-peer systems," in *Proc. 1st Biennial Conf. on Innovative Data Systems Research*, 2003.

[GCD04] N. Ganguly, G. Canright, A. Deutsch, "Design of a Robust Search Algorithm for P2P Networks," in *11th Int. Conf. on High Performance Computing*, December 2004.

[GFB+04] L. Garces-Erice, P.A. Felber, E.W. Biersack, G. Urvoy-Keller, K.W. Ross, "Data indexing in peer-to-peer DHT networks," in *Proc. 24th Int. Conf. on Distributed Computing Systems*, pp. 200-208, 2004.

[giFT] The giFT project, *The FastTrack protocol*, http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL?rev=HEAD&content-type=text/vnd.viewcvs-markup (retrieved 1.6.2010).

[GJ79] M. L. Garey, D. S. Johnson, *Computers and Interactability: A Guide to the Theory of NP-Completeness*, WH Freeman and Co., San Francisco, 1979.

[GK98] S. Guha1, S. Khuller, "Approximation Algorithms for Connected Dominating Sets," in *Algorithmica*, vol. 20, no. 4, April 1998.

[GM06] M. Garcia-Martin, M. Matuszewski, *A Session Initiation Protocol (SIP) Event Package and Data Format for Describing Generic Resources*, draft-garcia-sipping-resource-event-package-01, December 2006.

[GMB+06] M. Garcia-Martin, M. Matuszewski, N. Beijar, J. Lehtinen, *A Framework for Sharing Resources with the Session Initiation Protocol (SIP)*, draft-garcia-sipping-resource-sharing-framework-01, Work in progress, December 2006.

[GMS05] C. Gkantsidis, M. Mihail, A. Saberi, "Hybrid search schemes for unstructured peer-to-peer networks," in *Proc. 24th Annu. Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2005)*, vol. 3, pp. 1526- 1537, March 2005.

[Gnutella] Gnutella, http://www.gnutella.com/.

[Google] Google, http://www.google.com/.

[GP02] R. Giovanni, S. Paolo, "An analysis of the node spatial distribution of the random waypoint mobility model for ad hoc networks," in *Proc. 2nd ACM Int. Workshop of Principles of Mobile Computing*, pp. 44-50, October 2002.

[GPV+99] E. Guttman, C. Perkins, J. Veizades, M. Day, *Service Location Protocol, Version 2*, IETF RFC 2608, June 1999.

[GS04] J. Gao, P. Steenkiste, "An adaptive protocol for efficient support of range queries in DHT-based systems," in *Proc. 12th IEEE Int. Conf. on Network Protocols (ICNP)*, pp. 239-250, October 2004.

[GSG02] K. P. Gummadi, S. Saroiu, S. D. Gribble, “King: Estimating Latency between Arbitrary Internet End Hosts,” in *Proc. ACM IMW*, pp. 5-18, November 2002.

[GT95] M. Gerla, J. Tsal, J. Multicluater, “Mobile, multimedia radio network,” in *ACM-Baltzer Journal of Wireless Networks*, vol. 1, issue 3, pp. 255-265, 1995.

[HDV+03] S. Helal, N. Desai, V. Verma, Lee Choonhwa, ”Konark - a service discovery and delivery protocol for ad hoc networks,” in *IEEE Wireless Communications and Networking, 2003 (WCNC 2003)*, vol. 3, pp. 2107-2113, March 2003.

[HHB09] E. Harjula, J. Hautakorpi, N. Beijar, “Peer-to-Peer SIP for Mobile Computing: Challenges and Solutions,” in *Mobile Peer-to-Peer Computing for Next Generation Distributed Environments: Advancing Conceptual and Algorithmic Applications*, Book chapter, Ed. Seet Boon-Chong, ISBN 978-1-60566-715-7, IG I Publishing, 2009.

[HHH+04] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, I. Stoica, “Complex queries in DHT-based peer-to-peer networks,” in *Proc. IPTPS*, 2004.

[HHL+03] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, I. Stoica, “Querying the internet with PIER,” in *Proc. 29th int. conf. on Very large data bases*, vol. 29, pp. 321-332, 2003.

[HJ98] M. Handley, V. Jacobson, *SDP: Session Description Protocol*, IETF RFC 2327, April 1998.

[HJS+03] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, A. Wolman, “Skipnet: a scalable overlay network with practical locality properties,” in *Proc. 4th conf. on USENIX Symp. on Internet Technologies and Systems*, 2003.

[HKV09] M. Heikkinen, A. Kivi, H. Verkasalo, “Measuring Mobile Peer-to-Peer Usage: Case Finland 2007,” in *Proc. of Passive and Active Network Measurement Conf. (PAM 2009)*, pp. 165-174, 2009.

[HN09] M. Heikkinen, J. Nurminen, “Consumer Attitudes towards Different Aspects of Mobile Peer-to-Peer Services,” in *Proc. of 1st Int. Conf. on Advances in P2P Systems (AP2PS 2009)*, Accepted for publication, October 2009.

[How03] R. Howorth, “Fedora smartens peer-to-peer kit,” in *IT Week*, http://www.infomaticsonline.co.uk/itweek/comment/2086033/fedora-smartens-peer-peer-kit (retrieved 27.8. 2010), November 2003.

[Hyy06] T. Hyyryläinen, *Mobile P2P Client Implementation on Symbian*, Special assignment course S-38.3138, Helsinki University of Technology, Espoo, March 2006.

[IEEE1990] Institute of Electrical and Electronics Engineers, *Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges*, 802.1D-1990, IEEE standard, 2004.

[IEEE2004] Institute of Electrical and Electronics Engineers, *IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges*, 802.1D-2004, IEEE standard, 2004.

[IEEE2007] Institute of Electrical and Electronics Engineers, *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 802.11-2007, IEEE standard, 2007.

[iSkoot] http://iskoot.com/.

[ITU98] International Telecommunications Union Telecommunication Standardization Sector, Study group 16, *Packet-based multimedia communications systems*, ITU-T Recommendation H.323, February 1998.

[JBS92] M. C. Jeruchim, P. Balaban, K. S. Shanmugan, *Simulation of Communication Systems*, ISBN 0-306-46267-2, Plenum Press, New York, 1992.

[JFP04] S. Jain, K. Fall, R. Patra, "Routing in a delay tolerant network," in *Proc. of ACM SIGCOMM*, August 2004.

[JHM07] D. Johnson, Y. Hu, D. Maltz, *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*, IETF RFC 4728, February 2007.

[JJ07] S. Jin, H. Jiang, "Novel approaches to efficient flooding search in peer-to-peer networks", in *Computer Networks*, vol. 51, issue 10, pp. 2818-2832, Elsevier, July 2007.

[JMR07] C. Jennings, R. Mahy, A. B. Roach, *Relay Extensions for the Message Session Relay Protocol (MSRP)*, IETF RFC 4976, September 2007.

[Jos06] S. Josefsson, *The Base16, Base32, and Base64 Data Encodings*, IETF RFC 4648, October 2006.

[JZM+08] X. Jiang, H. Zheng, C. Macian, V. Pascual, *Service Extensible P2P Peer Protocol*, IETF Internet-Draft, draft-jiang-p2psip-sep-01, Work in progress, February 2008.

[Kazaa] Kazaa, http://www.kazaa.com/.

[KBB+04] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, M. Faloutsos, “Is P2P dying or just hiding?,” in *Global Telecommunications Conf. 2004 (IEEE GLOBECOM '04)*, Dallas, Texas, USA, November 2004.

[KCB01] R. Kantola, J. Costa-Requena, N. Beijar, “Interoperable routing for IN and IP Telephony,” in *Computer Networks*, vol. 35, no. 5, pp. 597-609, Elsevier, 2001.

[KCB00] R. Kantola, J. Costa-Requena, N. Beijar, “A Common Numbering Infrastructure for IN and IP Telephony,” in *Intelligent Network Workshop (IN2000)*, Cape Town, South Africa, 2000.

[KEP07] I. Kelényi, P. Ekler, Z. Pszota, Symtorrent version 1.30, Budapest University of Technology and Economics, Department of Automation and Applied Informatics, http://symtorrent.aut.bme.hu/, October 2007.

[KFM07] I. Kelényi, B. Forstner, B. Molnár, Symella version 1.40, Budapest University of Technology and Economics, Department of Automation and Applied Informatics, http://symella.aut.bme.hu/, October 2007.

[KG02] Taek Jin Kwon, M. Gerla, “Efficient flooding with Passive Clustering (PC) in ad hoc networks,” in *ACM SIGCOMM Computer Communication Review*, vol. 32, issue 1, pp. 44-56, January 2002.

[KGZ02] V. Kalogeraki, D. Gunopulos, D. Zeinalipour-Yazti, “A local search mechanism for peer-to-peer networks,” in *Proc. 11th Int. Conf. on Information and Knowledge Management*, McLean, Virginia, USA, pp. 300-307, 2002.

[Kir98] S. Kirsner, “The legend of Bob Metcalfe,” in *Wired*, issue 6.11, November 1998.

[KIS+03] T. Kato, N. Ishikawa, H. Sumino, J. Hjelm, Y. Yu, S. Murakami, “A platform and applications for mobile peer-to-peer communications,” in *Proc. Workshop on Emerging Applications for Wireless and Mobile Access (MobEA)*, May 2003.

[KK08] N. Klym, M. J. Montpetit, *Innovation at the Edge: Social TV and Beyond*, MIT Communications Futures Program VCDWG Working Paper, MIT Communications Futures Program, http://cfp.mit.edu/publications/CFP_Papers/Social%20TV%20Final%202008.09.01%20for%20distribution.pdf (retrieved 1.10.2009), September 2008.

[KKK+07] I. N. Kostamo, O. Kassinen, T. Koskela, M. Ylianttila, “Analysis of Concept and Incentives for Digital Content Superdistribution,” in *6th Conf. on Telecommunication Techno-Economics, 2007 (CTTE 2007)*, June 2007.

[KKO03a] M. Klein, B. König-Ries, P. Obreiter, “Service rings - a semantic overlay for service discovery in ad hoc networks,” in *Proc. 14th Int. Workshop on Database and Expert Systems Applications, 2003*, pp. 180-185, September 2003.

[KKO03b] M. Klein, B. König-Ries, P. Obreiter, *Lanes – A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Networks*, University of Karlsruhe, Technical report, no. 2003-6, May 2003.

[KLL+06] H. Khartabil, E. Leppanen, M. Lonnfors, J. Costa-Requena, *An Extensible Markup Language (XML)-Based Format for Event Notification Filtering*, IETF RFC 4661, September 2006.

[KM02] T. Klingberg, R. Manfredi, *Gnutella 0.6*, http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html (retrieved 27.8.2010).

[KM07] W. Kiess, M. Mauve, “A survey on real-world implementations of mobile ad-hoc networks,” in *Ad Hoc Networks*, vol. 5, pp. 324–339, 2007.

[KMW04] F. Kuhn, T. Moscibroda, R. Wattenhofer, “Unit disk graph approximation,” in *Workshop on Discrete Algothrithms and Methods for MOBILE Computing and Communications*, pp. 17-23, Philadelphia, PA, USA, 2004.

[KT03] U. C. Kozat, L. Tassiulas, “Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues,” in *Ad Hoc Networks*, vol. 2, issue 1, pp. 23-44, January 2004.

[Kui02] E. Kuitto, *Lukiolaistyttöjen ja -poikien ystävyyskäytännöt sekä käsitykset ystävyydestä (The friendship practices and the conceptions of friendship among the high school girls and boys)*, Master’s thesis, University of Helsinki, 2002.

[KWZ08] F. Kuhn, R. Wattenhofer, A. Zollinger, “Ad hoc networks beyond unit disk graphs,” in *Wireless Networks*, 14:715–729, 2008.

[LAD+05] L. Li, D. Alderson, J. C. Doyle, W. Willinger, “Towards a Theory of Scale-Free Graphs: Definition, Properties, and Implications,” in *Internet Mathematics*, vol. 2, no. 4, pp. 431-523, 2005.

[Lag09] S. Lagerström, *Sociala nätverk i mobiltelefon*, Bachelor's Thesis, Helsinki University of Technology, Espoo, April 2009.

[LCC+02] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. 16th annu. ACM Int. Conf. on supercomputing (ICS),* 2002.

[LCP+04] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," in *IEEE Communications Survey and Tutorial*, March 2004.

[Leh06] J. Lehtinen, *Design and Implementation of Mobile Peer-to-Peer Application*, Master's thesis, Helsinki University of Technology, Espoo, January 2006.

[Leh08] J. Lehtinen, *Mobile Peer-to-Peer over Session Initiation Protocol*, Licentiate thesis, Helsinki University of Technology, Espoo, August 2008.

[Lev09] T. Levä, *Questionnaire Study on mobile Peer-to-Peer*, Special assignment course S-38.3138, Helsinki University of Technology, Espoo, January 2009.

[LinkedIn] LinkedIn, http://www.linkedin.com/.

[LK01] H. Lim, C. Kim, "Flooding in wireless ad hoc networks," in *Computer communications*, vol. 24, issues 3-4, pp. 353-363, February 2001.

[Lug08] G. Lugano, "Mobile social networking in theory and practice," in *First Monday*, vol. 13, no. 11, November 2008.

[LZX+03] Yunhao Liu, Zhenyun Zhuang, Li Xiao, Lionel M. Ni, "AOTO: Adaptive Overlay Topology Optimization in Unstructured P2P Systems," in *Global Telecommunications Conf., 2003 (IEEE GLOBECOM '03)*, vol. 7, pp. 4186-4190, December 2003.

[LZX+04] Yunhao Liu, Zhenyun Zhuang, Li Xiao, Lionel M. Ni, "A distributed approach to solving overlay mismatching problem," in *Proc. 24th Int. Conf. on Distributed Computing Systems*, pp. 132-139, 2004.

[Mar02] E. P. Markatos, "Tracing a Large-Scale Peer to Peer System: An Hour in the Life of Gnutella," in *2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGRID'02)*, 2002.

[MBB06] A. N. Mian, R. Beraldi, R. Baldoni, *Survey of Service Discovery Protocols in Mobile Ad Hoc Networks*, Technical Report 4/06, Universit degli Studi di Roma La Sapienza, Rome, Italy, 2006.

[MBL+06a] M. Matuszewski, N. Beijar, J. Lehtinen, T. Hyyryläinen, “Mobile Peer-to-Peer Content Sharing Application,” in *The 3rd IEEE Consumer Communications and Networking Conf. (CCNC 2006)*, vol. 2, pp. 1324-1325, 2006.

[MBL+06b] M. Matuszewski, N. Beijar, J. Lehtinen, T. Hyyryläinen, “Content sharing in mobile P2P networks: myth or reality?,” in *Int. Journal of Mobile Network Design and Innovation*, vol. 1, 3/4, pp. 197-207, Inderscience Enterprises Ltd, 2006.

[MBL+07] M. Matuszewski, N. Beijar, J. Lehtinen, T. Hyyryläinen, “Understanding Attitudes Towards Mobile Peer-to-peer Content Sharing Services,” in *IEEE Int. Conf. on Portable Information Devices, PORTABLE07*, pp. 1 - 5, Orlando, Florida, USA, 2007.

[MCB+04] K. Marossy, G. Csucs, B. Bakos, L. Farkas, J. K. Nurminen, “Peer-to-peer content sharing in wireless networks,” in *15th IEEE Int. Symp. on Personal, Indoor and Mobile Radio Communications, 2004 (PIMRC 2004),* vol. 1, pp. 109- 114, September 2004.

[Mea02] M. Mealling, *Dynamic Delegation Discovery System (DDDS) – Part Three: The Domain Name System (DNS) Database*, IETF RFC 3403, October 2002.

[MGB+07] M. Matuszewski, M. A. García-Martín, N. Beijar, J. Lehtinen, “Resource Sharing and Discovery on Top of IMS,” in *IEEE Consumer Communications and Networking Conf., CCNC2007*, Las Vegas, Nevada, USA, 2007.

[Minidom] xml.dom.minidom - Lightweight DOM implementation, http://docs.python.org/library/xml.dom.minidom.html.

[MM02] N. Maibaum, T. Mundt, “JXTA: a technology facilitating mobile peer-to-peer networks,” in *Proc. International Mobility and Wireless Access Workshop (MobiWac 2002)*, pp. 7-13, Fort Worth, TX, USA, October 2002.

[MMule] MobileMule, http://mobil.emule-project.net/.

[Moy94] J. Moy, OSPF Version 2, IETF RFC 1583, March 1994.

[myHeimat] myHeimat, http://www.myheimat.de.

[MySQL] Kevin Atkinson, MySQL++, http://tangentsoft.net/ mysql++/.

[MZ01] A. B. McDonald, T. F. Znati, “Design and performance of a distributed dynamic clustering algorithm for ad hoc networks”, in *Proc. 34th Annu. Simulation Symp.*, pp. 27-35, April 2001.

[Napster] Napster, http://www.napster.com/.

[Nor09] E. Nordbäck, *Intresseanalys av gruppkommunikation med mobiltelefon*, Bachelor's Thesis, Helsinki University of Technology, Espoo, April 2009.

[Octopus] Octopus network, http://www.octo.fi/.

[OIK03] T. Ohta, S. Inoue, Y. Kakuda, "An adaptive multihop clustering scheme for highly mobile ad hoc networks," in *The Sixth Int. Symp. on Autonomous Decentralized Systems 2003 (ISADS 2003)*, pp. 293- 300, April 2003.

[ORe05] T. O'Reilly, *What is Web 2.0 – Design Patterns and Business Models for the Next Generation of Software*, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (retrieved 27.8.2010), September 2005.

[OSM+05] L. Oliveira, I. Siqueira, D. Macedo, A. Loureiro, H. C. Wong, J. Nogueira, "Evaluation of Peer-to-Peer Network Content Discovery Techniques over Mobile Ad Hoc Networks", in *IEEE Int. Symp. on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 51-56, 2005.

[ÖV99] M. T. Özsu, P. Valduriez, *Principles of Distributed Database Systems*, 2nd Edition, Prentice-Hall, 1999.

[P2PNext] P2P-Next, http://www.p2p-next.org/.

[P2PSIP] IETF P2PSIP WG Draft charter, http://www.ietf.org/html.charters/p2psip-charter.html.

[Pal09] A. Palmgren, *Mikrobloggning*, Bachelor's Thesis, Helsinki University of Technology, Espoo, April 2009.

[Pan09] V. Pankakoski, *Implementation of a Mobile Peer-to-Peer Client*, Special assignment course S-38.3138, Helsinki University of Technology, 2009.

[Partysip] The partysip SIP proxy server, http://www.partysip.org/.

[PB94] C. E. Perkins, P. Bhagwat, "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers," in *ACM SIGCOMM Computer Communication Review*, vol. 24, issue 4, pp. 234-244, October 1994.

[PBC+03] R. Price, C. Bormann, J. Christoffersson, H. Hannu, Z. Liu, J. Rosenberg, *Signaling Compression (SigComp)*, IETF RFC 3320, January 2003.

[PBD03] C. Perkins, E. Belding-Royer, S. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing*, IETF RFC 3561, July 2003.

[PC04] G. Pujolle, H. Chaouchi, GBN and STP/SP: “Beyond TCP/IP crisis over wireless networks,” in *The Personal, Indoor and Mobile Radio Communications Symp. (PIMRC)*, Barcelona, Spain, 2004.

[PH99] M. R. Pearlman, Z. J. Haas, “Determining the optimal configuration for the zone routing protocol,” in *IEEE Journal on Selected Areas in Communications*, vol. 17, issue 8, pp. 1395-1414, August 1999.

[PlanetLab] PlanetLab - An open platform for developing, deploying, and accessing planetary-scale services, http://www.planet-lab.org/.

[PMK+04] M. Poikselkä, G. Mayer, H. Khartabil, A. Niemi, *The IMS: IP Multimedia Concepts and Services in the Mobile Domain*, ISBN 0-470-87113-X, John Wiley & Sons, England, 2004.

[PONGsim] N. Beijar, Python Overlay Network Graphical Simulator (PONGsim), http://www.netlab.tkk.fi/tutkimus/mobilep2p/pongsim.

[PRR97] C. Plaxton, R. Rajaraman, A. Richa, “Accessing nearby copies of replicated objects in a distributed environment,” in *ACM Symp. on Parallel Algorithms and Architectures*, 1997.

[Python] Python Software Foundation, Python Programming Language - Official Website, http://www.python.org/.

[QB06] Y. Qiao, F. E. Bustamante, “Structured and unstructured overlays under the microscope: a measurement-based view of two P2P systems that people use,” in *Proc. USENIX '06 Annu. Tech. Conf.*, pp. 341-355, Boston, 2006.

[RBR+04] M. Roussopoulos, M. Baker, D. S.H. Rosenthal, T. J. Giuli, P. Maniatis, J. Mogul, “2 P2P or Not 2 P2P?,” in *The 3rd Int. Workshop on Peer-to-peer Systems*, February 2004.

[RCJ+02] O. Ratsimor, D. Chakraborty, A. Joshi, T. Finin, “Allia: alliance-based service discovery for ad hoc environments,” in *Proc. 2nd int. workshop on Mobile commerce*, pp. 1-9, 2002.

[RD01] A. Rowstron, P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” in *Proc. 18th IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware)*, November 2001.

[reSIP] The Open Source Community, reSIProcate, http://www.resiprocate.org/.

[Rey07] V. H. Morales Reyes, *Design and implementation of a distributed file directory for mobile peer-to-peer*, Master's thesis, Helsinki University of Technology, July 2007.

[RF02] M. Ripeanu, I. Foster, "Mapping gnutella network: Macroscopic properties of large-scale peer-to-peer systems," in *1st Int. Workshop on Peer-to-Peer Systems (IPTPS'02)*, March 2002.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A Scalable Content-Addressable Network," in *Proc. ACM SIGCOMM '01*, August 2001.

[RFI02] M. Ripeanu, I. Foster, A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design," in *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.

[Rie95] M. J. Riezenman, "The search for better batteries," in *IEEE Spectrum*, vol. 32, no. 5, pp. 51-56, May 1995.

[Rit01] J. Ritter, *Why gnutella can't scale. no, really*, http:// www.darkridge.com/~jpr5/doc/gnutella.html (retrieved 27.8.2010), 2001.

[RM06] J. Risson, T. Moors, "Survey of research towards robust peer-to-peer networks: Search methods," in *Computer Networks*, vol. 50, no. 17, pp. 3485-3521, December 2006.

[RMM08] J. Rosenberg, R. Mahy, P. Matthews, *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*, draft-ietf-behave-turn-09.txt, work in progress, July 2008.

[Roa02] A. B. Roach, *Session Initiation Protocol (SIP)-Specific Event Notification*, IETF RFC 3265, June 2002.

[Ros04] J. Rosenberg, *A Presence Event Package for the Session Initiation Protocol (SIP)*, IETF RFC 3856, August 2004.

[Ros07] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*, draft-ietf-mmusic-ice-19, work in progress, October 2007.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, *SIP: Session Initiation Protocol*, IETF RFC 3261, June 2002.

[RV03] P. Reynolds, A. Vahdat, "Efficient peer-to-peer keyword searching," in *Middleware 2003*, LNCS 2672, pp. 21-40, 2003.

[RW95] L. Råde, B. Westergren, *Mathematics Handbook for Science and Engineering*, 3rd edition, ISBN 91-44-25053-3, Sweden, 1995.

[SBR04] N. Sarshar, P. Oscar Boykin, V. P. Roychowdhury, "Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable," in *Proc. 4th Int. Conf. on Peer-to-Peer Computing*, pp. 2-9, 2004.

[SBW07] S. A. Hosseini Seno, R. Budiarto, Tat-Chee Wan, "Survey and new Approach in Service Discovery and Advertisement for Mobile Ad hoc Networks," in *Int. Journal of Computer Science and Network Security*, vol. 7, no. 2, pp. 275-284, February 2007.

[Sch01] R. Schollmeier, "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications," in *1st Int. Conf. on Peer-to-Peer Computing (P2P'01)*, pp. 101, 2001.

[SEK+92] M. F. Schwartz, A. Emtage, B. Kahle, B. Clifford Neuman, "A comparison of internet resource discovery approaches," in *Computing Systems*, vol. 5, no. 4, pp. 461-493, 1992.

[SETI] SETI@home, http://setiathome.berkeley.edu/.

[SG03] Qixiang Sun, Hector Garcia-Molina, "Partial lookup services," in *Proc. 23rd Int. Conf. on Distributed Computing Systems*, pp. 58-67, 2003.

[SGA+04] O. D. Sahin, A. Gupta, D. Agrawal, A. El Abbadi, "A peer-to-peer framework for caching range queries," in *Proc. 20th Int. Conf. on Data Engineering*, pp. 165- 176, March-April 2004.

[SGF02] R. Schollmeier, I. Gruber, M. Finkenzeller, "Routing in Mobile Ad Hoc and Peer-to-Peer Networks - A Comparison," in *Int. Workshop on Peer-to-Peer Computing*, pp. 1-15, 2002.

[SGG02] S. Saroiu, K. P. Gummadi, S. D. Gribble, "A measurement study of Peer-to-Peer File Sharing Systems," in *Proc. of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, CA, USA, January 2002.

[SGG03] S. Saroiu, K. P. Gummadi, S. D. Gribble, "Measuring and analyzing the characteristics of Napster and Gnutella hosts," in *Multimedia Systems*, 9:170–184, 2003.

[SIPshare] Earthlink SIPshare, http://www.research.earthlink.net/p2p.

[Skype] Skype, http://www.skype.com/.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balkakrishnan, “Chord: A scalable peer-to-peer lookup service for Internet applications,” in *Proc. SIGCOMM*, pp. 149-160, 2001.

[Soi09] T. Soinio, *Access Control and Group Management in a Mobile Peer-to-Peer System*, Master’s thesis, Helsinki University of Technology, Espoo, March 2009.

[Soi10] J. Soitinaho, *Approximate Information Filtering in Publish/Subscribe P2P Networks*, Licentiate thesis, Helsinki University of Technology, Espoo, 2010.

[Spotify] Spotify – a world of music, Instant, simple and free, http://www.spotify.com/en/.

[SRC84] J. H. Saltzer, D. P. Reed, D. D. Clark, “End-to-end arguments in system design,” in *ACM Transactions on Computer Systems (TOCS)*, vol. 2, issue 4, pp. 277-288, November 1984.

[SS02] R. Schollmeier, G. Schollmeier, “Why peer-to-peer (P2P) does scale: an analysis of P2P traffic patterns,” in *2nd Int. Conf. on Peer-to-Peer Computing (P2P'02)*, pp. 112-119, September 2002.

[Sun99] Sun Microsystems Inc, *Why Jini Technology Now?*, White paper, rev. 1.0, http://www.sun.com/software/jini/whitepapers/whyjininow.pdf (retrieved 2.10.2008), January 1999.

[T08] ------. ”Netti kaatui kun Kauhajoella pamahti,” in *Taloussanomat*, http://www.taloussanomat.fi/tietoliikenne/2008/09/23/netti-kaatui-kun-kauhajoella-pamahti/200824821/12 (retrieved 23.9.2008).

[TinyXML] L. Thomason, TinyXML, http://www.grinninglizard.com/tinyxml/.

[TKK+08] P. Tiago, N. Kotilainen, H. Kokkinen, M. Vapa, J. K. Nurminen, “Mobile Search - Social Network Search Using Mobile Devices,” in *The 5th IEEE Consumer Communications and Networking Conf. (CCNC 2008)*, pp. 1201-1205, January 2008.

[TKL+07] W. W. Terpstra, J. Kangasharju, C. Leng, A. P. Buchmann, “Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search,” in *ACM SIGCOMM Computer Communication Review*, vol. 37, issue 4, pp. 49-60, October 2007.

[TP03] P. Triantafillou, T. Pitoura, “Towards a unifying framework for complex query processing over structured peer-to-peer data networks,” in *Proc. 1st Int. Workshop on Databases, Information Systems and Peer-to-peer Computing (DBISP2P)*, pp. 169-183, September 2003.

[Twitter] Twitter, http://www.twitter.com.

[Upnp08] UPnP Forum, *UPnP Device Architecture 1.0*, April 2008, http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf (retrieved 27.8.2010).

[VDB05] K. Vanthournout, G. Deconinck, R. Belmans, “A taxonomy for resource discovery,” in *Personal and Ubiquitous Computing*, vol. 9, no. 2, pp. 81–89, March 2005.

[Ver07] H. Verkasalo, *A Cross-Country Comparison of Mobile Service and Handset Usage*, Licentiate’s thesis, Helsinki University of Technology, Finland, 2007.

[VW07] G. Vickery, S. Wunsch-Vincent, *Participative Web And User-Created Content: Web 2.0, Wikis and Social Networking*, ISBN:978-92-64-03746-5, Organization for Economic Cooperation and Development (OECD), Paris, France, 2007.

[WAF04] Peng-Jun Wan, Khaled M. Alzoubi, Ophir Frieder, “Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks,” in *Mobile Networks and Applications*, vol. 9, no. 2, April 2004.

[WIH05] H. Wan, N. Ishikawa, J. Hjelm, “Autonomous Topology Optimization for Unstructured Peer-to-Peer Networks,” in *Proc. 11th Int. Conf. on Parallel and Distributed Systems 2005 (ICPADS’05)*, 2005.

[WPD88] D. Waitzman, C. Partridge, S. Deering, *Distance Vector Multicast Routing Protocol*, RFC 1075, November 1988.

[WS98] D. J. Watts, S. H. Strogatz, “Collective dynamics of ’small-world’ networks,” in *Nature*, 393:440–442, 1998.

[YC05] J. Y. Yu, P. H. J. Chong, “A survey of clustering schemes for mobile ad hoc networks,” in *IEEE Communications Surveys & Tutorials*, vol. 7, issue 1, pp. 32-48, 2005.

[YG02] B. Yang and H. Garcia-Molina, “Efficient search in peer-to-peer networks,” in *Int. Conf. on Distributed Computing Systems (ICDCS)*, 2002.

[YLY+04] Hao Yang, Haiyun Luo, Fan Ye, Songwu Lu, Lixia Zhang, “Security in mobile ad hoc networks: challenges and solutions,” in *IEEE Wireless Communications*, vol. 11, no. 1, pp. 38-47, 2004.

[ZH05] R. Zhang, Y. C. Hu, "Assisted Peer-to-Peer Search with Partial Indexing," in *Proc. 24th Annu. Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM 2005)*, vol. 3, pp. 1514-1525, March 2005.

[ZKJ01] B. Zhao, J. Kubiatowicz, A. Jospeh, *Tapestry: an infrastructure for fault-tolerant wide-area location and routing*, Report no. UCB/CSD-01-1141, 2001.

[ZLZ+05] Q. Zheng, X. Lu, P. Zhu, W. Peng, "An efficient random walks based approach to reduce file locating delay in unstructured P2P networks," in *Proc. Global Telecommunications Conf. (IEEE GLOBECOM '05)*, 2005.