PEER-TO-PEER INFORMATION RETRIEVAL

ALMER S. TIGELAAR

PhD dissertation committee

Chairman and Secretary		
	Prof. dr. ir. A. J. Mouthaan	University of Twente, NL
Supervisor		
	Prof. dr. P. M. G. Apers	University of Twente, NL
Assistant Supervisor		
	Dr. ir. D. Hiemstra	University of Twente, NL
Members		
	Prof. dr. J. P. Callan	Carnegie Mellon University, US
	Prof. dr. F. Crestani	Università della Svizzera Italiana, CH
	Prof. dr. ir. A. P. de Vries	Delft University of Technology, NL
	Prof. dr. F. M. G. de Jong	University of Twente, NL
	Prof. dr. D. K. J. Heylen	University of Twente, NL
	Dr. ir. J. A. Pouwelse	Delft University of Technology, NL

CTIT PhD Dissertation Series No. 11-222
Centre for Telematics and Information Technology (CTIT)
P.O. Box 217, 7500 AE Enschede, The Netherlands

SIKS Dissertation Series No. 2012-29
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems

The research in this thesis was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 639.022.809

ISBN: 978-90-365-3400-0
ISSN: 1381-3617, No. 11-222
DOI: 10.3990/1.9789036534000

PEER-TO-PEER INFORMATION RETRIEVAL

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Wednesday, September 26th, 2012 at 14:45

by

Almer Sampath Tigelaar
born on May 9th, 1982
in Avissawella, Sri Lanka

This dissertation is approved by:

Prof. dr. Peter M. G. Apers (supervisor)
Dr. ir. Djoerd Hiemstra (assistant supervisor)

‘Don’t tell me what I can’t do!’

John Locke (Lost)

PREFACE

'After I got my PhD, my mother took great relish in introducing me as, "this is my son, he's a doctor but not the kind that helps people."'

Randy Pausch

It was a warm summer day in August. I was standing in a line waiting for my turn. It was hot, my T-shirt was soaked, my feet hurt, but I felt good, real good. I looked around and listened to the music dampened through my earplugs. I saw people dancing in the distance to my right, someone zip-lining past them and groups of people drinking and chatting in an open air lounge-like area to my left. The social atmosphere was good here, it had been for days. This was Hungary, this was Budapest, this was Sziget Festival. The queue dissolved and finally it was my turn. I went inside the massive white tent which was filled with endless rows of computers. I had only 10 minutes of Internet access: make them count.

I quickly opened my mailbox and found a mail by Djoerd Hiemstra. He worked at the database group of the University of Twente, a different research group from where I got my Master's degree just a month earlier. I had contacted him before I left and he was now asking whether I would be interested in a PhD position on a new project he was starting. There were more mails to plough through and I just spent a couple of long nights doing things that, well, are fun, but not exactly conducive to thinking straight, so I typed a quick reply telling him that I would get back to him when I returned to the Netherlands.

That was four years ago. We had a meeting, I expressed interest, applied for the job and the rest is history. Now, four years later, the thesis is finished and the project is drawing to a close. As any PhD student will

tell you: the process is not easy. You spend four years with a subject until you practically eat, drink and sleep it. It is a test of aptitude that makes you an expert on your terrain. However, it is certainly not for everyone.

You get a lot of freedom to structure your own research process and to pursue the direction of your choosing. However, this is a double-edged sword, as there is the risk of engaging in unnecessary detours, running in circles and getting stuck in dead-ends. A good supervisor can do wonders to prevent these pitfalls and get you back on track. The nice things about doing a PhD are undeniably that you get to meet lots of interesting people, enjoy a great deal of personal freedom and travel frequently.

I had a great time during the ESSIR summer school in Padova, Italy, and was grateful that a friend was able to facilitate a visit to the information retrieval research group, in nearby Lugano, led by Fabio Crestani. I am happy he agreed to be on my committee. I also had the opportunity to spend four months at Carnegie Mellon University in Pittsburgh as a visiting scholar. It was challenging at times, but enjoyable. I would like to thank Jamie Callan for his excellent guidance, and am honoured that he is on my committee. Furthermore, I thank Anagha whom I greatly enjoyed cooperating with and David for his technical support. Also, Luís, Wang and Arnold for making it a pleasant stay, my 'host' family: Stacy, Andy, Ellie, Ben, Larry and Jackie, as well as the people at CMU Film Club.

I remember a brief visit to the Delft University of Technology where Djoerd and I had a pleasant interaction about peer-to-peer systems with Johan Pouwelse, who also agreed to be on my committee. Dirk Heylen inquired several times about the status of my PhD, and I am happy he joined my committee, as well as Franciska de Jong from the same group. I encountered Arjen de Vries at several conferences, and am honoured he is part of my committee too.

I would like to thank my colleagues at the database group. First and foremost my daily supervisor, Djoerd Hiemstra, who was always a great help for generating new and fresh ideas, my supervisor Peter Apers, the few moments we shared were useful, and Dolf Trieschnigg for his feedback. I would also like to thank Maarten for his guidance, Maurice for occasional tips, Jan for assisting with technical issues and Ida & Suse for helping out with too many things to list here. I also want to thank my direct colleagues over the years, particularly: Riham, Victor and Mohammad, and also: Robin, Harold, Sander, Anand, Sergio, Juan, Mena and

Rezwan.

And then, of course, there are those really close people that stay with you throughout the years, to whom I am thankful for supporting me during the highs and lows inevitably part of such a long timespan: Mario, Desi, Marco, Guido, Martijn, Menno, Sara, Edwin, Annemarie and Robert. As well as: Alexander, Isaac, Zwier, Danny, François, Thijs, Ruben, Jan, Gert, Mareije, Marco P., Niels N., Maher, Nisa, Fenne and Bayan.

Then there's all the people with whom I lived together during my PhD: Robert, Thomas, Chiel, Dirk-Maarten, Vanessa, Janina, Niels, Maarten, Julia, Sara, Jochem, Lieke, Maike, Chris, Dirk, Stefan, Marissa, Inga, Katharina, Twan and René. Thanks for providing a home.

I also want to thank my family: Klaas, Marrie, Rianne, Dennis, Maura, Sander, Erma and Britt. I am happy that my sister Rianne and my trusted friend Mario agreed to be my paranymphs.

I want to thank all those that read (parts) of this thesis and provided helpful suggestions. Firstly, Marco for reading large parts, nagging me about the mathematical notation, meticulously checking my references, and for sparring about the propositions with me. Secondly, Mario for spotting those final lay-out issues. Gratitude to the many others that primarily read the introductory chapter, and various other parts: Dolf, Victor, Mohammad, Menno, Sara, Maike and Robert.

Finally, I want to thank the people involved in creating all the fiction and non-fiction I read, listened, watched and played over the past four years. Thanks to them there was always something to look forward to, even when the going got tough.

Almer සම්පත් Tigelaar,
August 2012.

CONTENTS

1 INTRODUCTION

'We could say we want the web to reflect a democratic world vision. To do that, we get computers to talk with each other in such a way as to promote that ideal.'

Tim Berners-Lee

What is peer-to-peer information retrieval and why should you care and read this thesis? Where can you find the parts that are interesting for you? All these questions are answered in this chapter.

Over the past decade the Internet has become an integral part of our daily lives. We follow the news, view fascinating videos and listen to our favourite music on-line. One task is essential to all these on-line activities: *finding information*. Nowadays, a rich palette of web search engines exists, from the big: Google, Bing and Yahoo; to the specific: YouTube for videos, Wikipedia for encyclopedia articles, IMDB for films; to the small: the search field on the website of your local library, university or favourite blog author. These search engines allow us to quickly find what we are looking for in large data collections in mere fractions of a second: a feat unprecedented in human history. Massive amounts of computers in large data centres enable the big search giants to provide their services. The research area that concerns itself with improving search technology is called *information retrieval*, formally defined as: 'the technique and process of searching, recovering and interpreting information from large amounts of stored data' (McGraw-Hill, 2002). In short:

searching for needles in a haystack. Conventionally, the haystack is a *document collection*, the needle is the specific *document* you are looking for and the text you use to describe that specific needle is called the *query*: the text you enter in a search box.

The web is not solely a consumption medium and offers more than a searchable gateway to information alone. In contrast with, for example, broadcast television, the World Wide Web encourages us to actively contribute and share. This ranges from creating our own content, to sharing our favourite existing stories, music and videos with others. The drive to share things with our peers is human nature. Given this, it is not surprising that applications that enable us to share content with each other directly have gained widespread popularity, nor is it surprising that these are called *peer-to-peer* applications. Familiar examples are Napster, Kazaa and BitTorrent, lesser known ones are Usenet, Skype and Spotify. The central idea of all peer-to-peer applications is the same: use the processing and storage capacity available at the edges of the Internet: the machines that we use every day, our desktop, laptop and tablet computers. The research field that focuses on improving these peer-to-peer systems is called *peer-to-peer computing*. It focuses on leveraging vast amounts of computing power, storage and connectivity from personal computers around the world (Zeinalipour-Yazti et al., 2004). These loosely coupled computers, the peers, are considered to be equal and supply as well as consume resources in order to complete application-specific tasks.

In this thesis, we bring together the field of peer-to-peer computing and information retrieval. The goal is to provide the foundation for a web search engine which, like the existing ones, enables us to find information in a fraction of a second, but that uses the computers in our homes to do so. But, why would we want to do this?

1.1 MOTIVATION

The big commercial search engines dominate search in the world's largest document collection: the World Wide Web. Their search services use many machines arranged in data centres which they exclusively control. These machines download as many web pages as they can find, a process referred to as *crawling*, and index these documents. Conceptually, an index contains, for each term, the documents in which the term occurred. Using this information, search engines can quickly suggest relevant pages

for a given query. The approach of storing the index in large data centres and performing the searches there is termed *centralised search*: a single party provides a search service over a large collection of documents.

The dominance of large search engines raises at least three ethical concerns. Firstly, search engine companies can afford to buy vast amounts of storage space and computing power which enables them to store and access very large indices. Since not everyone can do this, the search engine operators effectively control the information that can be found, thereby establishing an information monopoly and censorship capabilities (Kulathuramaiyer and Balke, 2006; Mowshowitz and Kawaguchi, 2002). This position can be abused to suppress freedom of speech and censor crimes committed by oppressive regimes. Secondly, conflicts of interest may occur, particularly with respect to products and services of competitors (White, 2009; Edelman, 2010). The ability to monitor people's interest and controlling what they get to see, makes it easy to influence the success of products in the marketplace beyond conventional advertising and to play the stock market in an unprecedented way. Thirdly, the elaborate tracking of user behaviour forms a privacy risk (Tene, 2008). In contrast with regulated service providers like physicians, lawyers and bankers, there is little legal protection for sensitive personal information revealed to on-line search services.

There are also a number of technical concerns. Firstly, the web is a medium which encourages users to create and publish their own content. This has been a driving force behind its explosive growth and one can question whether centralised solutions can keep up with the rapid pace at which content is added, changed and removed (Lewandowski et al., 2006). There is a need for alternative scalable solutions that can take over when their centralised counterparts start to fail. Secondly, a large amount of dynamically generated content is hidden behind web search forms that cannot easily be reached by centralised search engines: the *deep web* (Bergman, 2001). Thirdly, in central search the central party decides how, when and what parts of a website are indexed: it does not enable websites and search systems to completely independently manage their search data (Galanis et al., 2003).

It would be better if no single party dominates web search. Users and creators of web content should collectively provide a search service. This would restore to them control over what information they wish to share as well as how they share it. Importantly, no single party would dominate in

such a system, mitigating the ethical drawbacks of centralised search. Additionally, this enables handling dynamic content and provides scalability, thereby removing the technical weaknesses of centralised systems. Unfortunately, no mature solution exists for this. However, peer-to-peer information retrieval could form the foundation for such a collective search platform. The way people share content with each other maps naturally onto the peer-to-peer paradigm (Oram, 2001): a peer does not only passively consume resources, but also actively contributes resources back.

Peer-to-peer information retrieval is a fascinating and challenging research area. In the past decade a number of prototype peer-to-peer search engines have been developed in this field (Akavipat et al., 2006; Suel et al., 2003; Bender et al., 2005b; Luu et al., 2006). While promising, none of these has seen widespread real-world adoption. There are several likely reasons for this. Firstly, peer-to-peer systems are challenging to develop. Secondly, there is a lack of commercial interest in research and development of these systems, perhaps since their centralised counterparts are easier to monetise. Thirdly, they are not yet capable of providing a viable alternative to contemporary centralised search engines. Despite these reasons, we believe that peer-to-peer information retrieval systems are a promising alternative to centralised solutions. The challenges that remain can be resolved by focused research and development. While a peer-to-peer web search engine would not instantly solve all of the ethical and technical drawbacks of centralised search, it would be a good first step in the right direction, as it enables solutions that directly and explicitly involve users and content creators to a degree which is difficult or impossible in a centralised approach. For example: users could help improve search result quality by moderating and providing feedback (Chidlovskii et al., 2000; Freyne et al., 2004); content creators can make their dynamic content available and help with keeping the index fresh; both users and creators can contribute their computing resources, like spare bandwidth, storage and processing cycles, to provide a large distributed index with good scalability properties (Krishnan et al., 2007).

With this thesis we aim to provide a new direction for applying the peer-to-peer paradigm to information retrieval and hope to inspire further research in this area. Sufficient interest can lead to the development and large-scale deployment of real-world peer-to-peer information retrieval systems that rival existing centralised client-server solutions in terms of scalability, performance, user satisfaction and freedom.

1.2 RESEARCH TOPICS

The main goal of this thesis is to take several steps to make real-world peer-to-peer web search systems more viable. The first part of the thesis is theoretical and focuses on describing and framing existing concepts and systems. It centres on the following research topics (RT's):

> *RT 1. What is peer-to-peer information retrieval, how does it differ from related research fields and what are its unique challenges?*

A key element, that has been missing so far, is a clear and unambiguous definition of what peer-to-peer information retrieval really is. One reason for this is that it overlaps with at least two related research fields. Firstly, peer-to-peer file sharing systems in which free text search is performed to find files of interest. Secondly, federated information retrieval that concerns itself with how a federation of search engines can collectively provide a search service. There is a need for a clear definition of what separates peer-to-peer information retrieval from these related fields: an understanding of its unique challenges. Furthermore, we briefly investigate the economics of peer-to-peer systems.

> *RT 2. What general peer-to-peer architectures, information retrieval systems and optimisation techniques already exist?*

Over the years, a number of peer-to-peer system architectures evolved. We aim to provide an overview of the existing architectures, highlighting their strengths and weaknesses from an information retrieval perspective, as these are the foundation of any real-world system. Furthermore, we also want to identify which peer-to-peer information retrieval systems so far have been influential, what techniques can be used to optimise such systems and how economics can be applied to peer-to-peer systems.

Conducting experiments with all peer-to-peer architectures is impossible due to time constraints. For the remainder of the thesis we would like to pick one architecture to use as the basis for our experiments. Based on the investigation we conducted for the previous two research topics, we make an informed choice concerning the architecture we prefer for further research and development.

The second part of the thesis assumes an architecture where one logical party, termed the *tracker*, is responsible for suggesting relevant peers in

response to queries. It focuses on practical experiments using this particular architecture to create a peer-to-peer web search system and focuses on the following research topics:

> *RT 3. How can content at uncooperative peers, and existing external search systems, be made searchable?*

There are many existing systems that could be integrated in a peer-to-peer web search engine. However, changing their interface is either unwanted or impossible. We would like to have some way of deciding whether queries can be directed to these systems with only a minimal search interface. We investigate and adapt an existing technique that sends computer-generated probing queries to these systems to estimate their content. The information obtained this way can, at a later stage, be used to direct actual user queries to the systems believed to be capable of providing relevant search results. This enables access to a broader amount of information, aiding the transition to a fully cooperative peer-to-peer environment.

> *RT 4. How could we best represent the content of cooperative peers to maximise the relevance of search results?*

Cooperative peers can provide information about their content to the tracker, so it can perform more effective peer selection. However, it is not obvious what this information should be and how it should be used. Furthermore, we would like to know the costs of transmission and storage of this information, so that we can maximise relevance based on a minimal amount of information. We compare the performance of a broad range of possible representations.

> *RT 5. How can we distribute the query load and minimise the latency for retrieving search results?*

Users of modern centralised web search engines have become accustomed to fast sub-second response times for their queries. There is a need for a mechanism that also makes this possible for peer-to-peer web search engines. We investigate keeping copies of previously obtained search results at each peer as a solution. We take into account the volatile nature of peer-to-peer systems and explore a simple incentive mechanism for retaining search result caches.

1.3 THESIS STRUCTURE

This thesis aims to first introduce peer-to-peer systems from an information retrieval perspective which is done in Chapter 2 (RT1&2). This is followed by a survey of existing systems and techniques in Chapter 3 (RT2). Section 3.5 contains the choice for our experimental setting. That concludes the theoretical part of this thesis.

The first two experimental chapters focus primarily on maximising relevance of search results: in Chapter 4 we look at a technique called query-based sampling to model uncooperative peers (RT3), followed by Chapter 5 in which we try to find out how to best represent cooperative peers for maximising search result relevance (RT4).

In our final experiment, in Chapter 6, we shift focus to minimising latency whilst retaining search result relevance: we show how search result caching can be used to perform load balancing while keeping into account the unique characteristics of peer-to-peer networks and also briefly touch on using reputations as an incentive mechanism for caching (RT5).

2

PEER-TO-PEER NETWORKS

'The whole is greater than
the sum of its parts.'

Aristotle

This chapter presents the high-level concepts and distinctions central to peer-to-peer systems. It starts with a general overview and moves towards challenges specific to information retrieval.

A node is a computer connected to a network. This network facilitates communication between the connected nodes through various protocols enabling many distributed applications. The Internet is the largest contemporary computer network with a prolific ecosystem of network applications. Communication occurs at various levels called *layers* (Kurose and Ross, 2003, p. 55). The lowest layers are close to the physical hardware, whereas the highest layers are close to the software. The top layer is the application layer in which communication commonly takes place according to the *client-server paradigm*: *server* nodes provide a resource, while *client* nodes use this resource. An extension to this is the *peer-to-peer paradigm*: here each node is equal and therefore called a *peer*. Each peer could be said to be a client and a server at the same time and thus can both *supply* and *consume* resources. In this paradigm, peers need to cooperate with each other, balancing their mutual resources

This chapter is based on Tigelaar et al. (2012): *Peer-to-Peer Information Retrieval: An Overview*, that appeared in *ACM Transactions on Information Systems*, Volume 32, Issue 2 (May 2012). ©ACM, 2012. http://doi.acm.org/10.1145/2180868.2180871.

in order to complete application-specific *tasks*. For communication with each other, during task execution, the peers temporarily form *overlay networks*: smaller networks within the much larger network they are part of. Each peer is connected to a limited number of other peers: its *neighbours*. Peers conventionally transmit data by forwarding from one peer to the next or by directly contacting other, non-neighbouring, peers using routing tables. The *architecture* of a peer-to-peer network is determined by the shape of its overlay network(s), the placement and scope of indices, local or global, and the protocols used for communication. The choice of architecture influences how the network can be utilised for various tasks such as searching and downloading.

In practice the machines that participate in peer-to-peer networks are predominantly found at the edge of the network, meaning they are not machines in big server farms, but computers in people's homes (Kurose and Ross, 2003, p. 165). Because of this, a peer-to-peer network typically consists of thousands of low-cost machines, all with different processing and storage capacities as well as different link speeds. Such a network can provide many useful applications, like: file sharing, streaming media and distributed search. Peer-to-peer networks have several properties that make them attractive for these tasks. They usually have no centralised directory or control point and thus also no *central point of failure*. This makes them *self-organising*, meaning that they automatically adapt when peers join the network, depart from it or fail. The communication between peers uses a language common among them and is *symmetric* as is the provision of services. This symmetry makes a peer-to-peer network *self-scaling*: each peer that joins the network adds to the available total capacity (Bawa et al., 2003; Risson and Moors, 2006).

In the following sections we will first discuss applications of peer-to-peer networks and the challenges for such networks, followed by an in-depth overview of commonly used peer-to-peer network architectures.

2.1 APPLICATIONS

Many applications use peer-to-peer technology. Some examples:

- *Content Distribution*: Usenet, Akamai, Steam.
- *File Sharing*: Napster, Kazaa, Gnutella, BitTorrent.

- *Information Retrieval:* Sixearch, YaCy, Seeks.
- *Instant Messaging*: ICQ, MSN.
- *Streaming Media*: Tribler, Spotify.
- *Telephony*: Skype, SIP.

Significant differences exist among these applications. One can roughly distinguish between applications with mostly *private data*: instant messaging and telephony; and *public data*: content distribution, file sharing, information retrieval and streaming media. The term peer-to-peer is conventionally used for this latter category of applications, where the sharing of public data is the goal which is also the focus of this thesis. The interesting characteristic of public data is that there are initially only a few peers that supply the data and there are many peers that demand a copy of it. This asymmetry can be exploited to widely replicate data and provide better servicing for future requests. Since file sharing networks are the most pervasive peer-to-peer application, we will frequently use it as an example and basis for comparison, especially in the initial sections of this chapter that focus on the common characteristics of peer-to-peer computing. However, in Section 2.6 we will shift focus to the differences and give a definition of peer-to-peer information retrieval and what sets it apart from other applications.

The concepts *query*, *document* and *index* will often be used in this thesis. What is considered to be a query or document, and what is stored in the index, depends on the application. For most content distribution, file sharing and streaming media systems, the documents can be files of all types. The index consists of metadata about those files and the queries are restricted to searching in this metadata space. Information retrieval usually involves a large collection of text documents of which the actual content is indexed and searchable by using free text queries. For searching in instant messaging networks, and telephony applications, the documents are user profiles of which some fields are used to form an index. The query is restricted to searching in one of these fields, for example: 'nickname'.

2.2 CHALLENGES

There are many important challenges specific to peer-to-peer networks (Daswani et al., 2003; Triantafillou et al., 2003; Milojicic et al., 2003):

- *How to make efficient use of resources?*
  Resources are bandwidth, processing power and storage space. The higher the efficiency, the more requests a system can handle and the lower the costs for handling each request. Peers may vary wildly in their available resources which raises unique challenges.

- *How to provide acceptable quality of service?*
  Measurable important aspects are: low latency and sufficient high-quality results.

- *How to guarantee robustness?*
  Provide a stable service to peers and the ability to recover from data corruption and communication errors whatever the cause.

- *How to ensure data remains available?*
  When a peer leaves the network its content is, temporarily, not accessible. Hence, a peer-to-peer network should engage in quick distribution of popular data to ensure it remains available for as long as there is demand for it.

- *How to provide anonymity?*
  The owners and users of peers in the network should have at least some level of anonymity depending on the application. This enables censorship resistance, freedom of speech without the fear of persecution and privacy protection.

Additionally, several behaviours of peers must be handled:

- *Churn*
  The stress caused on a network by the constant joining and leaving of peers is termed churn. Most peers remain connected to the network only for a short time. Especially if the network needs to maintain global information, as in a network with a decentralised global index, this can lead to, recurring and costly, shifting and rebalancing of data over the network. This behaviour also reduces the

availability of data. Peers may leave willingly, but they can also simply crash (Klampanos et al., 2005). A peer-to-peer network should minimise the communication needed when a peer leaves or joins the network (Stutzbach and Rejaie, 2006).

- *Free riding*
  A peer-to-peer network is built around the assumption that all peers in the network contribute a part of their processing power and available bandwidth. Unfortunately, most networks also contain peers that only use resources of other peers without contributing anything back. These peers are said to engage in free riding. A peer-to-peer network should both discourage free riding and minimise the impact that free riders have on the performance of the network as a whole (Krishnan et al., 2002).

- *Malicious behaviour*
  While free riding is just unfair consumption of resources, malicious behaviours actively frustrate the usage of resources, either by executing attacks or 'poisoning' the network with fake or corrupted data. A peer-to-peer network should be resilient to such attacks, be able to recover from them and have mechanisms to detect and remove poisoned data (Kamvar et al., 2003; Keyani et al., 2002).

Finally, it remains difficult to evaluate and compare different peer-to-peer systems. For this we define the following research challenges:

- *Simulation*
  Most peer-to-peer papers use self-developed simulation frameworks. This may be surprising since several peer-to-peer simulators exist. However, these have several problems like having limited ways to obtain statistics, poor documentation and are generally hard to use (Naicken et al., 2006, 2007). Creating a usable framework for a wide-range of peer-to-peer experiments is a challenge.

- *Standardised test sets*
  Simulations should use standardised test sets, so that results of different solutions to peer-to-peer problems can be compared. For a file sharing network this could be a set of reference files, for an information retrieval network a set of documents, queries and relevance judgements. Creating such test collections is often difficult and labour-intensive. However, they are indispensable for science.

2.3 TASKS

We distinguish three tasks that every peer-to-peer network performs:

1. *Searching*: Given a query return some list of document references.
2. *Locating*: Resolve a document reference to concrete locations from which the full document can be obtained.
3. *Transferring*: Actually download the document.

From a user perspective the first step is about identifying what one wants, the second about working out where it is and the third about obtaining it (Joseph, 2002). Peer-to-peer networks do not always decentralise all of these tasks and not every peer-to-peer architecture caters well to each task, as we will see later. The key point is that searching is different from locating. We will concretely illustrate this using three examples.

Firstly, in an instant messaging application, searching would be looking for users that have a certain first name or that live in a specific city, for example: for all people named Zefram Cochrane in Bozeman, Montana. This search would yield a list with various properties of matching users, including a unique identifier, from which the searcher picks one, for example: the one with identifier 'Z2032'. The instant messaging application can use this to locate that user: resolving the identifier to the current machine address of the user, for example: 5.4.20.63. Finally, transferring would be sending an instant message to that machine.

Secondly, in information retrieval the search step would be looking for documents that contain a particular phrase, for example 'pizza baking robots'. This would yield a list of documents that either contain the exact phrase or parts thereof. The searcher then selects a document of interest with a unique identifier. Locating would involve finding all peers that share the document with that identifier and finally downloading the document from one of these.

As a final example let us consider the first two tasks in file sharing networks. Firstly, *searching*: given a query find *some possible* files to download. This step yields unique file identifiers necessary for the next step, commonly a hash derived from the file content. Secondly, *locating*: given a specific file identifier find me other peers that offer *exactly* that file. What distinguishes these is that in the first, one still has to choose what one wants to download from the search results, whereas in the second, one

knows exactly what one wants already and one is simply looking for replicas. These two tasks are cleanly split in, for example, BitTorrent (Cohen, 2003). A free text search yields a list of possible torrent files: small metadata files that each describe the real downloadable file with hash values for blocks of the file. This is followed by locating peers that offer parts of this real file using a centralised machine called the tracker. Finally, the download proceeds by obtaining parts of the file from different peers. BitTorrent thus only decentralises the transfer task, and uses centralised indices for both searching and locating. However, both BitTorrent extensions and many other file sharing networks increasingly perform locating within the network using a distributed global index. A distributed global index can also be used for the search task. Networks that use aggregated local indices, like Gnutella2 (Stokes, 2002), often integrate the search and locate tasks: a free-text search yields search results with, for each file, a list of peers from which it can be obtained.

2.4 ARCHITECTURES

There are multiple possible architectures for a peer-to-peer network. The choice for one of these affects *how* the network can be searched. To be able to search, one requires an index and a way to match queries against entries in this index. Although we will use a number of examples, it is important to realise that what the index is used for is application-specific. This could be mapping filenames to concrete locations in the case of file sharing, user identifiers to machine addresses for instant messaging networks, or terms to documents in the case of information retrieval. In all cases the challenge is keeping the latency low whilst retaining the beneficial properties of peer-to-peer networks like self-organisation and load balancing (Daswani et al., 2003). Based on this there are several subtasks for searching that all affect the latency:

- *Indexing: Who constructs and updates the index? Where is it stored and what are the costs of mutating it?*
  The peers involved in data placement have more processing overhead than others. There can be one big global index or each peer can index its own content. Peers can specialise in only providing storage space, only filling the index or both. Where the index is stored also affects query routing.

- *Querying Routing: Along what path is a query sent from an issuing peer to a peer that is capable of answering the query via its index?*
  Long paths are expensive in terms of latency, and slow network links and machines worsen this. The topology of the overlay network restricts the possible paths.

- *Query Processing: Which peer performs the actual query processing (generating results for a specific query based on an index)?*
  Having more peers involved in query processing increases the latency and makes fusing the results more difficult. However, if less peers are involved it is likely that relevant results will be missed.

These search subtasks are relevant to tasks performed in all peer-to-peer networks. In the following paragraphs we discuss how these subtasks are performed in four commonly used peer-to-peer architectures using file sharing as example, since many techniques used in peer-to-peer information retrieval are adapted from file sharing networks.

2.4.1 *Centralised Global Index*

Early file sharing systems used a *centralised global index* located at a dedicated party, usually a server farm, that kept track of what file was located at which peer in the network. When peers joined the network they sent a list of metadata on files they wanted to share containing, for example, filenames, to the central party that would then include them in its central index. All queries that originated from the peers were directly routed to and processed by that central party. Hence, indexing and searching was completely centralised and followed the client-server paradigm. Actually obtaining files, or parts of files, was decentralised by downloading from peers directly. This is sometimes referred to as a brokered architecture, since the central party acts as a mediator between peers. The most famous example of this type of network is Napster. This approach avoids many problems of other peer-to-peer systems regarding query routing and index placement. However, it has at least two significant drawbacks. Firstly, a central party limits the scalability of the system. Secondly, and more importantly, a central party forms a single point of technical, and legal, failure (Aberer and Hauswirth, 2002; Risson and Moors, 2006).

2.4.2 *Distributed Global Index*

Later systems used a *distributed global index* by partitioning the index over the peers: both the index and the data are distributed in such networks. These indices conventionally take the form of a large key-value store: a distributed hash table (Stoica et al., 2001). When a peer joins the network it puts the names of the files it wants to share as keys in the global index and adds its own address as value for these filenames. Other peers looking for a specific file can then obtain a list of peers that offer that file by consulting the global distributed index. Each peer stores some part of this index. The key space is typically divided in some fashion over peers, making each peer responsible for keys within a certain range. This also determines the position of a peer in the overlay network. For example: if all peers are arranged in a ring, newly joining peers would bootstrap themselves in between two existing peers and take over responsibility for a part of the key space of the two peers. Given a key, the peer-to-peer network can quickly determine what peer in the network stores the associated value. This key-based routing has its origins in the academic world and was first pioneered in Freenet (Clarke et al., 2001). There are many ways in which a hash table can topologically be distributed over the peers. However, all of these approaches have a similar complexity for lookups: typically $\mathcal{O}(\log n)$, where n is the total number of peers in the network. A notable exception to this are hash tables that replicate all the globally known key-value mappings on each peer. These single-hop distributed hash tables have a complexity of $\mathcal{O}(1)$ (Monnerat and Amorim, 2009). The primary difference between hash table architectures is the way in which they adapt when peers join or leave the network and in how they offer reliability and load balancing. A complete discussion of this is beyond the scope of this thesis, but is described by Lua et al. (2005). We restrict ourselves to briefly describing a popular contemporary hash table implementation: Kademlia, and some common drawbacks of hash tables.

Maymounkov and Mazières (2002) introduce Kademlia: a distributed hash table that relies on the XOR metric for distance calculation in the key space. Using this metric has the desirable property that a single routing mechanism can be used, whereas other hash tables conventionally switch routing strategies as they approach a key that is being looked up. Furthermore, Kademlia employs caching along the look-up path for specific keys to prevent hotspots. As is common in distributed hash tables,

each peer knows many peers near it and a few peers far away from it in the keyspace. Kademlia keeps these lists updated such that long-lived nodes are given preference for routing in the keyspace. This prevents routing attacks that rely on flooding the network with new peers. The importance of relying on long-lived peers in a peer-to-peer network has been shown before by Bustamante and Qiao (2004). Kademlia also offers tuning parameters so that bandwidth can be sacrificed to obtain lower routing latency. Kad is a large operational contemporary network, used for file sharing, that implements Kademlia.

Unfortunately, Kademlia, as many other hash tables, is vulnerable to attacks. An infamous example of this is the Sybil attack. The purpose of this attack is to introduce malicious peers that are controlled by a single entity. These can be used to execute a variety of secondary attacks, for example: frustrating routing in the network, attacking a machine external to the network by traffic flooding or obtaining control of a part of the key space. This last attack is termed an eclipse attack and allows an external entity to return whatever value it wishes for a particular key or range of keys. Steiner et al. (2007) show that as few as eight peers are enough to take over a specific key in Kademlia. They suggest that these attacks can be prevented by using public-key infrastructure, or hierarchical admission control. Although taking over a complete network of millions of peers would likely require more than taking control of several thousand peers, the authors stress that practical solutions are urgently needed to prevent Sybil attacks on existing deployed networks. Lesniewski-Laas and Kaashoek (2010) introduce Whanau, a hash table that reduces the global knowledge peers hold and is resistant to Sybil attacks to some extent: it has a high probability of returning the correct value for a particular key, even if the network is under attack.

A global index can also be implemented using gossip to replicate the full index for the entire network at each peer as done by Cuenca-Acuna et al. (2003). However, this approach is not often used and conceptually quite different from hash tables. A key difference is that each peer may have a slightly different view of what the global index contains at a given point in time, since it takes a while for gossip to propagate. In this way it is also close to aggregation described in Section 2.4.4. The slow propagation may or may not be acceptable depending on the application and size of the network. We propose to use the term *replicated global index* to distinguish this approach.

2.4.3 *Strict Local Indices*

An alternative is to use *strict local indices*. Peers join the network by contacting bootstrap peers and connecting directly to them or to peers suggested by those bootstrap peers until reaching some neighbour connectivity threshold. A peer simply indexes its local files and waits for queries to arrive from neighbouring peers. An example of this type of network is the first version of Gnutella (Aberer and Hauswirth, 2002). This network performs search by propagating a query from its originating peer via the neighbours until reaching a fixed number of hops, a fixed time-to-live, or after obtaining a minimum number of search results: query flooding (Kurose and Ross, 2003, p. 170). One can imagine this as a ripple that originates from the peer that issued the query: a breadth-first search (Zeinalipour-Yazti et al., 2004). Unfortunately, this approach scales poorly as a single query generates massive amounts of traffic even in a moderate size peer-to-peer network (Risson and Moors, 2006). Thus, there have been many attempts to improve this basic flooding approach. For example: by forwarding queries to a limited set of neighbours, resulting in a random walk (Lv et al., 2002), by directing the search (Adamic et al., 2001; Zeinalipour-Yazti et al., 2004), or by clustering peers by content (Crespo and Garcia-Molina, 2004) or interest (Sripanidkulchai et al., 2003). An important advantage of this type of network is that no index information ever needs to be exchanged or synchronised. Thus, index mutations are inexpensive, and all query processing is local and can thus employ advanced techniques that may be collection-specific, but query routing is more costly than in any other architecture discussed, as it involves contacting a large subset of peers. While the impact of churn on these networks is lower than for global indices, poorly replicated, unpopular data may become unavailable due to the practical limit on the search horizon. Also, peers with low bandwidth or processing capacity can become a serious bottleneck (Lu, 2007).

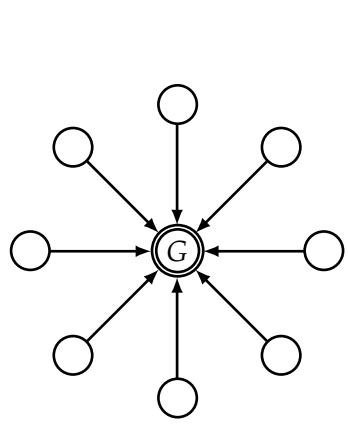
2.4.4 *Aggregated Local Indices*

A variation, or rather optimisation, on the usage of local indices are *aggregated local indices*. Networks that use this approach have at least two, and sometimes more, classes of peers: those with high bandwidth and processing capacity are designated as *super peers*, the remaining 'leaf' peers are each assigned to one or more super peers when they join the net-

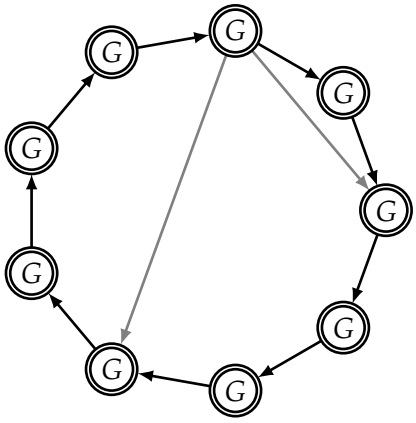
work. A super peer holds the index of both its own content as well as an aggregation of the indices of all its leafs. This architecture introduces a hierarchy among peers and by doing so takes advantage of their inherent heterogeneity. It was used by FastTrack and in recent versions of Gnutella (Liang et al., 2006; Stokes, 2002). Searching proceeds in the same way as when using strict local indices. However, only the super peers participate in routing queries. Since these peers are faster and well connected, this yields better performance compared to local indices, lower susceptibility to bottlenecks, and similar resilience to churn. However, this comes at the cost of more overhead for exchanging index information between leaf peers and super peers (Yang et al., 2006; Lu and Callan, 2006a). The distinction between leaf and super peers need not be binary, but can instead be gradual based on, for example, node uptime. Usually leaf peers generate the actual search results for queries using their local index. However, it is possible to even delegate that task to a super peer. The leafs then only transmit index information to their super peer and pose queries.

2.4.5 *Discussion*

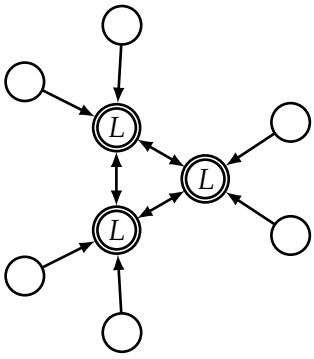
Figure 2.1 depicts the formed overlay networks for the described peer-to-peer architectures. These graphs serve only to get a general impression of what form the overlay networks can take. The number of participating peers in a real network is typically much higher. Figure 2.1a shows a centralised global index: all peers have to contact one dedicated machine, or group thereof, for lookups. Figure 2.1b shows one possible instantiation of a distributed global index shaped like a ring (Stoica et al., 2001). There are many other possible topological arrangements for a distributed global index overlay, the choice of which only mildly influences the typical performance of the network as a whole (Lua et al., 2005). These arrangements all share the property that they form *regular graphs*: there are no loops, all paths are of equal length and all nodes have the same degree. This contrasts with the topology for aggregated local indices shown in Figure 2.1c, that ideally takes the form of a *small world graph*: this has loops, random path lengths and variable degrees that result in the forming of clusters. Small world graphs exhibit a short global separation in terms of hops between peers. This desirable property enables decentralised algorithms that use only local information for finding short paths. Finally, strict local indices, Figure 2.1d, either take the form of a small world graph or



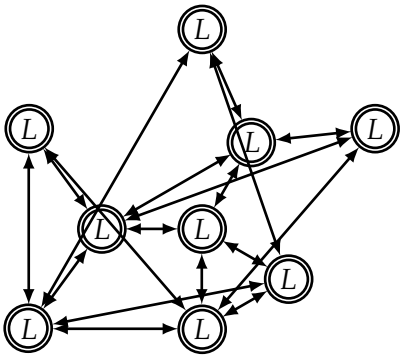
(a) central global index



(b) distributed global index



(c) aggregated local indices



(d) strict local indices

Figure 2.1: Overview of peer-to-peer index and search overlays. Each circle represents a peer in the network. Peers with double borders are involved in storing index information and processing queries. A *G* symbol indicates a peer stores a part of a global index, whereas an *L* symbol indicates a local index. The arrows indicate the origin of queries and the directions in which they flow through the system.

Table 2.1: Characteristics of Classes of Peer-to-Peer Networks

	Global Index		Local Indices	
	Centralised	*Distributed*	*Aggregated*	*Strict*
Index				
– Construction	Central Peer	All Peers	All Peers	All Peers
– Storage	Central Peer	All Peers (Shared)	Super Peers	All Peers (Indiv.)
– Mutation Cost⋆	Low	High	Low	None
Query Routing				
– Method	Direct	Forwarding	Forwarding	Forwarding
– Parties	Central Peer	Intermediate Peers	Super Peers	Neighbour Peers
– Complexity	$\mathcal{O}(1)$	$\mathcal{O}(\log n)^{\dagger}$	$\mathcal{O}(n_s)^{\ddagger}$	$\mathcal{O}(n)$
Query Processing				
– Peer Subset	Central Only	Small	Medium	Large
– Latency	Low	Medium	Medium	High
– Result Set Unit	Query	Term	Query	Query
– Result Fusion	–	Intersect	Merge	Merge
– Exhaustive	Yes	Yes	No$^{\diamond}$	No$^{\diamond}$

This list is not exhaustive, but highlights latency aspects of these general architectures important for information retrieval.
⋆In terms of network latency and bandwidth usage from Yang et al. (2006).
†There are also $\mathcal{O}(1)$ distributed hash tables (Risson and Moors, 2006; Monnerat and Amorim, 2009).
‡Applies to the number of super peers n_s.
$^{\diamond}$Searches are restricted to a subset of peers and thus to a subset of the index.

a random graph depending on whether they include some type of node clustering. A *random graph* can have loops and both random path lengths and node degrees (Aberer and Hauswirth, 2002; Kleinberg, 2006; Girdzijauskas et al., 2011). Besides the overall shape of the graph, the path lengths between peers are also of interest. Networks with *interest-based locality* have short paths between a peer and other peers with content similar to its interests. Keeping data closer to peers more likely to request it, reduces the latency and overall network load. Similarly, a network with *content-based locality* makes it easier to find the majority of relevant contents efficiently, since these are mostly near to one another: peers with similar content are arranged in clusters (Lu, 2007). These approaches are not mutually exclusive and can also be combined.

Table 2.2: Classification of Free-text *Search* in File Sharing Networks

	Global Index		Local Indices	
	Centralised	*Distributed*	*Aggregated*	*Strict*
BitTorrent	■			
FastTrack			■	
FreeNet		■		
Gnutella				■
Gnutella2			■	
Kad		■		
Napster	■			

Table 2.1 shows characteristics of the discussed peer-to-peer architectures and Table 2.2 shows an architectural classification for the search task in several existing popular peer-to-peer file sharing networks. We distinguish several groups and types of peers. Firstly, the central peer indicates the machine(s) that store(s) the index in a centralised global index. Secondly, the super peers function as mediators in some architectures. Thirdly, all the peers in the network as a whole and on an individual basis. These distinctions are important since in most architectures the peers involved in constructing the index are not the same as those involved in storage, leading to differences in mutation costs. A querying peer rarely has local results for its own queries. Hence, the network needs to route queries from the origin peer to result bearing peers. Queries can be routed either via forwarding between peers or by directly contacting a peer capable of providing results. Even the discussed distributed hash tables use forwarding between peers to 'hop' the query message through intermediate peers in the topology and close in on the peer that holds the value for a particular key. For all architectures the costs of routing a query is a function of the size of the network. However, the number of peers that perform actual processing of the query, and generate search results, varies from a single peer, in the centralised case, to a large subset of peers when using strict local indices. Lower latency can be achieved by involving fewer peers in query processing. For information retrieval networks, returned results typically apply to a whole query, except for the distributed global index that commonly stores results using individual terms as keys. For all approaches, except the central global index, it

is necessary to somehow fuse the results obtained from different peers. A distributed global index must intersect the lists of results for each term. Local indices can typically merge incoming results with the list of results obtained so far. The simplest form of merging is appending the results of each peer to the end of one large list.

The discussed approaches have different characteristics regarding locating suitable results for a query. The approaches that use a global index can search exhaustively. Therefore, it is easy to locate results for rare queries in the network: every result can always be found. In contrast, the approaches that use local indices can flood their messages to only a limited number of peers. Hence, they may miss important results and are slow to retrieve rare results. However, obtaining popular, well replicated, results from the network incurs significantly less overhead. Additionally, they are also more resilient to churn, since there is no global data to rebalance when peers join or leave the system (Lua et al., 2005). Local indices give the peers a higher degree of autonomy, particularly in the way in which they may shape the overlay network (Daswani et al., 2003). Advanced processing of queries, such as stemming, decompounding and query expansion, can be done at each peer in the network when using local indices, as each peer receives the original query. When using a global index these operations all have to be done by the querying peer, which results in that peer executing multiple queries derived from the original query, thereby imposing extra load on the network. Furthermore, one should realise that an index is only part of an information retrieval solution and that it cannot solve the relevance problem by itself (Zeinalipour-Yazti et al., 2004).

Solutions from different related fields apply to different architectures. Architectures using a global index have more resemblance to cluster and grid computing, whereas those using a local index have most in common with federated information retrieval. Specifically, usage of local indices gives rise to the same challenges as in federated information retrieval (Callan, 2000): resource description, collection selection and search result merging, as we will discuss later in Section 2.6.2.

An index usually consists of either one or two layers: a *one-step index* or a *two-step index*. In both cases the keys in the index are terms. However, in a one-step index the values are direct references to document identifiers, whereas in a two-step index the values are peer identifiers. Hence, a one-step index requires only one lookup to retrieve all the applicable

documents for a particular term. Strict local indices are always one-step. In a two-step index the first lookup yields a list of peers. The second step is contacting one or more peers to obtain the actual document identifiers. A one-step index is a straight *document index*, whereas a two-step index actually consists of two layers: a *peer index* and a document index per peer. A network with aggregated local indices is two-step when the leaf peers are involved in generating search results and the aggregated indices contain leaf peer identifiers. Two-step indices are commonly used in combination with a distributed global index: the global index maps terms to peers that have suitable results. A distributed global index requires contacting other peers most of the time for index lookups: even if we would store terms as keys and document identifiers as values, to perform a lookup one still needs to hop through the distributed hash table to find the associated value for a key. However, this is conceptually still a one-step index, since the distributed hash table forms one index layer. Some approaches use a third indexing layer intended to first map queries to topical clusters (Klampanos and Jose, 2004).

Peer-to-peer networks are conventionally classified as structured or unstructured. The approach with strict local indices is classified as unstructured and the approach that uses a distributed global index as structured. However, we agree with Risson and Moors (2006) that this distinction has lost its value, because most modern peer-to-peer networks assume *some type* of structure: the strict local indices approach is rarely applied. The two approaches are sometimes misrepresented as competing alternatives (Suel et al., 2003), whereas their paradigms really augment each other. Hence, some systems combine some properties of both (Rosenfeld et al., 2009). The centralised global index is structured because the central party can be seen as one very powerful peer. However, the overlay networks that form at transfer time are unstructured. Similarly, the aggregated indices approach is sometimes referred to as semistructured since it fits neither the structured nor the unstructured definition. We believe it is more useful to describe peer-to-peer networks in terms of their specific structure and application and the implications this has for real-world performance. Hence, we will not further use the structured versus unstructured distinction. Rather, we will focus on our primary application: searching in peer-to-peer information retrieval networks. However, we first look at the economics of peer-to-peer systems to get an understanding of potentially usable incentive mechanisms.

2.5 ECONOMICS

A peer-to-peer network is about resource exchange. This can be viewed in terms of economics, analysed using game theory and modelled using mechanism design. Taking an economic angle can make peer-to-peer technology into a more reliable platform for distributed resource-sharing. A necessary step towards this is the development of mechanisms by which the contributions of individual peers can be solicited and predicted. Incentives play a central role in providing a predictable level of service (Buragohain et al., 2003).

The concepts introduced in this section are not central to this thesis, but they do offer extra background for understanding the mechanisms used in successful contemporary peer-to-peer networks. Furthermore, the concepts presented here are used in a looser fashion in the experimental sections of this thesis and are required to understand the related work on peer-to-peer economics in Section 3.4.

2.5.1 *Economies*

In an economy decentralisation is achieved by *rational agents* that attempt to selfishly achieve their goals. We distinguish between two types of agents: *suppliers* and *consumers*. Each agent generally has some *resources* or *goods* that can be supplied to other agents. The preferences of an agent for consuming external resources can be expressed via a *utility function* that maps an external resource to a *utility value* (Ferguson et al., 1996; Leyton-Brown and Shoham, 2008). Agents should be facilitated in some way, so they can actually supply and consume resources. An economic system can be used for this, which should charge the agents for services they consume based on the value they derive from it (Buyya et al., 2001).

Before we present economic models, let us first briefly discuss the connections between economies, peer-to-peer and information retrieval. Firstly, in a peer-to-peer system a peer can be considered to be an agent. In this section we use these two terms interchangeably. Besides this, there are parallels between the utility of a search result and that of goods in an economy. Imagine that you are hungry: the first apple you eat has a high utility for you, whereas the second has less utility and the third even less. As your stomach fills up, eating more apples becomes less attractive. At some point eating yet an other apple might even make you sick and thus

have a negative utility. This effect is called *diminishing marginal utility*. For search results something similar applies assuming that they are ranked in order of descending relevance. The first result is the most important, whereas the second and third actually become less important. Eventually, having too many results adds unnecessarily confusion and diminished satisfaction. Users of web search engines favour precision: high *quality* results, over recall: a high *quantity* of results. This phenomenon is known as the paradox of choice and applies to much more than just search results (Oulasvirta et al., 2009; Flynn, 2005).

To achieve decentralisation there are two often used economic models: exchange-based and price-based. We will discuss both of these briefly. Let us first look at the *exchange-based economy*, sometimes called community, coalition, share holder or barter economy: each agent has some initial amount of resources and agents exchange resources until they all converge to the same level of satisfaction: when their *marginal rate of substitution* is the same. In this situation no further mutually beneficial exchanges are possible and the system is said to have achieved its *Pareto optimal* allocation, sometimes called Pareto efficient allocation. The defining characteristic of an exchange-based economy is that a Pareto optimal allocation method, involving selfish agents, can result in optimal decentralised resource allocation algorithms. This model works best when all participating agents are symmetric and thus provide as well as consume resources. This is conventionally the case in peer-to-peer networks.

The other often used economic model is the *price-based economy*. In this economy resources are priced based on demand, supply and the wealth in the economic system. Each agent initially has some wealth and computes the demand for some good from its utility function and budget constraint. In a price-based economy the goal of each agent is to *maximise revenue* (Ferguson et al., 1996). There are various approaches to how supply and demand are reconciled within a price-based economy. Often used ones are commodity markets, bargaining models and auctions. A complete discussion of this is beyond the scope of this thesis, but a good overview is given by Buyya et al. (2001). Most price-based models assume a competitive market, but there are plenty of cases where one company dominates a particular market and is the only supplier of a good: a *monopoly*. If competitive markets are one extreme, a monopoly is the other. Usually the situation is somewhere in between: a small number of suppliers dominate the market and set prices: an *oligopoly*.

What makes a good market model is difficult to define. Commonly used criteria include: the global good of all (social welfare), the global perspective (Pareto optimality), the amount of participation, stability of the mechanisms (resistance to manipulation), computational efficiency and communication efficiency. Measures like intervention of price regulation authorities can be used to prevent the market from collapsing. Alternatively, one can leave it to the market to consolidate naturally (Buyya et al., 2001). Clearly economic concepts, like pricing and competition, can provide solutions to reduce the complexity of service provisioning and decentralise access mechanisms to resources (Ferguson et al., 1996).

What has been described thus far applies to exchange of *privately owned goods*. There are also *public goods*: for example a lighthouse. Public goods are not excludable in supply, anyone can use them, and are non-rival in demand, everyone can use them simultaneously. They are not subject to traditional market mechanisms. Similarly, *club goods* are usually also non-rival in demand, but they are excludable in supply: only the club members can use them. A cable TV broadcast is a typical example of a club good: all subscribers can use the broadcast simultaneously. Club goods can be provided by a market by either charging only a flat membership fee, called *coarse exclusion*, or a membership fee and a usage based price, called *fine exclusion*.

The price for a public good, a *tax* to all members of the public, is ideally the *Lindahl equilibrium* which is always Pareto optimal. Unfortunately this equilibrium is hard to determine, since it requires complete knowledge of the individual demand for the good of each member of the public. Furthermore, the Lindahl equilibrium is hard to determine if malicious members misreport the benefit they gain from the good: lying is beneficial as this would mean lower taxes (Krishnan et al., 2007).

The concept of goods offers a framework to think about resource provisioning in peer-to-peer networks. There is clearly more than one way to apply these abstractions. One way would be to view the peer-to-peer network as a club. Every peer that joins the club gains access to the resources within: the club goods. In a peer-to-peer information retrieval network, this would be the search services of other peers and the results they can provide. An other way is to view the exchanged resources as private goods and apply conventional market mechanics, exchange-based or price-based, for decentralisation. Either way, we need some way to formally express and reason about such economic systems.

2.5.2 *Game Theory and Mechanism Design*

Game theory is often used to study economic situations in the form of simplified games and has been recognised as a useful tool for modelling the interactions of peers in peer-to-peer networks (Buragohain et al., 2003). Games are usually classified based on two main properties. Firstly, by the number of agents that participate: either two-person or more which is referred to as *n*-person. Secondly, by whether the game is zero-sum or not: in a zero-sum game for one agent to win an other agent must lose, whereas in a non-zero-sum game both parties can get better by cooperating (Davis, 1983). A peer-to-peer information retrieval network, where each peer can gain by exchanging search results with others, would be an example of an *n*-person non-zero-sum game.

In game theory the economic behaviour of rational agents is viewed as a *strategy*. Agents assign a *utility* to an external resource as discussed in the previous subsection. If an agent itself has limited resources, it may choose a *suboptimal* strategy and is considered to be a *bounded rational player* (Shneidman and Parkes, 2003; Leyton-Brown and Shoham, 2008). A game reaches its *weak Nash equilibrium* when no agent can gain by changing his strategy given that the strategies of all other agents are fixed. A *strong* or *strict Nash equilibrium* is when every agent is strictly worse off if he were to change his strategy given that all other agents' strategies are fixed (Golle et al., 2001). Not all Nash equilibria are Pareto optimal and not all Pareto optimums are a Nash equilibrium. We discuss a famous example to illustrate this: the prisoner's dilemma, a two-person non-zero-sum game.

Imagine that you and a friend are suspected of committing a crime together and are arrested by the police. Upon arrest, the police has found illegal weapons on both of you, which is considered a minor crime. However, they suspect the two of you have been involved in something bigger: a major crime. You are both placed in separate interrogation cells. Each of you may either remain silent or confess to the major crime. Each combination of actions has different consequences. If one confesses and the other does not, the police will set the confessor free and the other will go to jail for twenty years. If both of you confess, you both go to jail for five years. If both of you remain silent you both go to jail for one year for the minor crime of weapons possession. There is no way for the two of you to communicate since you are in separate rooms. We make the

		You	
		Confess	*Silent*
Friend	*Confess*	$(Y = 5, F = 5)$	$(Y = 20, F = 0)$
	Silent	$(Y = 0, F = 20)$	$(Y = 1, F = 1)$

Figure 2.2: Pay-off matrix for the Prisoner's Dilemma. The values are the number of years in prison for (Y)ou and your (F)riend depending on the choices you and him make shown in the row and column labels.

assumption that both of you are rational and act exclusively in your own best interest. We define remaining silent as playing cooperatively and confessing as playing non-cooperatively with respect to the other player. The pay-off matrix is shown in Figure 2.2. The problem with this game is that the only rational course of action is to play non-cooperatively: to both confess, even though you are both worse off, five years in prison, than if you would both remain silent: one year in prison. Consider that if you do not know what your friend will do, he will probably confess since that will set him free at your expense. If he confesses you are best of also confessing, since that reduces your sentence from twenty to only five years. Only very naïve players would both remain silent. This is an example of a problem that has a Nash equilibrium that is *not* Pareto optimal. There are numerous other game theoretic problems that follow a similar pattern (Davis, 1983; Myerson, 1997).

Ying stands to yang as game theory stands to *mechanism design*. Where game theory reasons about *how* agents will *play* a game, mechanism design reasons about how to *design* games that produce *desired outcomes*. In conventional mechanism design, players calculate a strategy and feed this to a centre that calculates and declares the outcome. Since determining optimal strategies and calculating outcomes can be quite hard, *algorithmic mechanism design* focuses on constructing mechanisms that retain computation feasibility. Finally, *distributed algorithmic mechanism design* further assumes that the mechanism is carried out via distributed computation and therefore maps better onto peer-to-peer networks. Distributed mechanisms can sometimes achieve better complexity results than centralised mechanisms (Shneidman and Parkes, 2003).

In all forms of mechanism design, the aim of the designer is to create a *good mechanism*. What is good can be defined in many ways: efficient, budget balanced, et cetera. A mechanism may have a centre: a central party

that makes decisions. Mechanism can be designed as *one-shot* or *repeated* and agent behaviour that may not seem rational in the short term, is hopefully rational in the long term. Furthermore, agents may be *faulty*: they stop working, drop messages or act arbitrarily. Techniques are necessary to detect, and remove, these *non-strategising* agents. Rational agents may learn from participation in a protocol to further refine their own strategy. *Irrational agents* do not follow an intended behaviour by the mechanism designer (Shneidman and Parkes, 2003).

Mechanism design can be used to create a good mechanism based on a set of pre-selected criteria. As we have seen in Section 2.6.1 these criteria are not the same for all peer-to-peer networks, and they are not the same for all tasks either. In a file sharing network maximising the downstream bandwidth usage may be the most important criterion during the transfer step. In a peer-to-peer information retrieval network this could instead be maximising the estimated relevance of search results during the search step and minimising the latency for obtaining those search results in the transfer step. Regardless, mechanism design provides a mental framework for thinking about these criteria and designing experiments in which peers optimise for them.

After this brief detour into economics, to which we will return later in Chapter 3, it is time to move on to the primary subject of this thesis: information retrieval in peer-to-peer systems.

2.6 INFORMATION RETRIEVAL

In an information retrieval peer-to-peer network the central task is *searching*: given a *query* return some list of document references: the *search results*. A query can originate from any peer in the network and has to be routed to other peers that can provide search results based on an index. The peers thus supply and consume results. A search result is a compact representation of a document that can contain text, image, audio, video or a mixture of these (Zeinalipour-Yazti et al., 2004). A search result, also called a *snippet*, at least includes a pointer to the full document and commonly additional metadata like: a title, a summary, the document size, et cetera. A concrete example: search results as displayed by modern search engines. Each displayed result links to the associated full document. The compact snippet provides a first filtering opportunity for users, enabling them to choose the links to follow.

Peer-to-peer information retrieval networks can be divided into two classes based on the location of the documents pointed to. Firstly, those with *internal* document references, where the documents have to be downloaded from other peers within the network, for example: digital libraries (Lu and Callan, 2006a; Di Buccio et al., 2009). Secondly, those with *external* document references, where obtaining the actual documents is outside of the scope of the peer-to-peer network, for example: a peer-to-peer web search engine (Bender et al., 2005b).

In the following subsections we compare peer-to-peer information retrieval networks with other applications and paradigms.

2.6.1 *Comparison with File Sharing Networks*

File sharing networks are used to search for, locate and download files that users of the peer-to-peer network share. The *searching* in such networks is similar to peer-to-peer information retrieval. A free text query is entered after which a list of files is returned. After searching, the user selects a file of interest to download that usually has some type of globally unique identifier, like a content-based hash. The next step is *locating* peers that have a copy of the file. It may then be either *transferred* from one specific peer or from several peers simultaneously in which case specific parts of the file are requested from each peer and stitched back together after the individual downloads complete.

The tasks of locating peers, and especially transferring content, are the primary application of file sharing networks and the focus of research and performance improvements. Searches in such networks are for known items, whereas in information retrieval networks the intent is more varied (Lu, 2007). While some information retrieval networks also provide locating and downloading operations, they typically focus on the search task. Besides this, there are at least three concrete differences.

Firstly, the search index for file sharing is usually based only on the names of the available files and not on their content as in information retrieval. Such a name index is smaller than a full document index (Suel et al., 2003). Hence, there are also fewer postings for each term, which makes it less costly to perform intersections of posting lists, an operation common in a distributed global index (Reynolds and Vahdat, 2003). Because of their small size the central approach scales well for name indices (Lu, 2007). However, they have become unpopular due to legal reasons.

Table 2.3: Differences between Locating in File Sharing and Searching in Information Retrieval Using a Two-Step Index

	File Sharing	Information Retrieval
Application	Locating	Searching
Index		
– Content	File identifiers	Document text
– Size	Small	Large
– Dominant Operation	Append	Update
– Document Location	Internal	External
– First Step Mapping	$fileid \rightarrow \{peer\}$	$term \rightarrow \{peer\}$
– Second Step Mapping	$fileid \rightarrow file$	$term \rightarrow \{document\}$
– Mapping Type	Exact lookup	Relevance ranking
– Result Fusion	Trivial	Difficult
Dominant Data Exchange		
– Unit	Files	Search results
– Size	Megabytes+ (large)	Kilobytes (small)
– Emphasis	High throughput	Low latency

Secondly, when a file is added to a file sharing index, it does not change. If an adjusted version is needed, it is simply added as a new file. Hence, index updates are not required. In contrast, in an information retrieval network, when the underlying document changes, the associated search results generated from that document have to change as well. Hence, the index needs to be updated so that the search results reflect the changes to the document pointed to.

Thirdly, since the emphasis in a file sharing network is on quickly downloading files, it is important to have *high throughput*. In contrast, in information retrieval the search task dominates, in which *low latency* is the most important (Reynolds and Vahdat, 2003). More concretely: it is acceptable if the network takes half a minute to locate the fastest peers for a download, whereas taking that long is not acceptable for obtaining quality search results. Table 2.3 summarises the differences assuming a two-step index and a peer-to-peer web search engine. For file sharing the index is the one used for *locating* a file, whereas for information retrieval it is the one used for *searching*. The first-step mapping is at the level of the whole network, whereas the second-step mapping is at a specific peer.

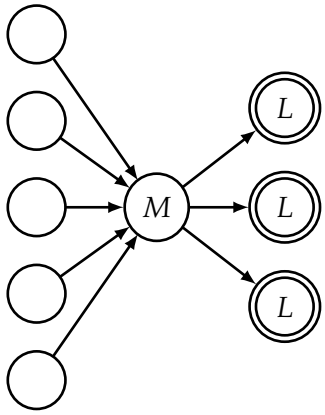


Figure 2.3: Schematic depiction of federated information retrieval. Each circle represents a peer in the network, those at the left are clients. Peers with double borders, at the right, are servers that maintain local indices marked with *L*. In between is the mediator node denoted with an *M*. The arrows indicate the origin of queries and the direction in which these flow through the system.

2.6.2 *Comparison with Federated Information Retrieval*

In federated information retrieval[1] there are three parties as depicted in Figure 2.3: *clients* that pose queries, one *mediator* and a set of search *servers* that each disclose a collection of documents: resembling strict local indices. The search process begins when a client issues a query to the mediator. The mediator has knowledge of a large number of search servers and contacts a subset of these appropriate for answering the query. Each search server then returns a set of search results for the query. The mediator merges these results into one list and returns this to the client (Callan, 2000).

Similarities

There are three challenges that form the pillars of federated information retrieval that it has in common with peer-to-peer information retrieval (Callan, 2000). Firstly, there is the *resource description problem*: the mediator either needs to receive an indication of the queries it can handle from

1 This is also referred to as distributed information retrieval. However, 'distributed' can be confused with general distributed systems such as server farms and grids. Hence, we stick to the now more popular term federated information retrieval.

each search server (Gravano et al., 1997), in the case of *cooperative* servers, or the mediator needs to discover this by probing the search servers if they are *uncooperative* (Du and Callan, 1998; Shokouhi and Zobel, 2007). In either case the end result is a resource description of the search server. These descriptions are typically kept small for efficiency reasons, as even large collections can be described with a relatively small amount of data (Tigelaar and Hiemstra, 2010b). The description can consist of, for example: summary statistics, collection size estimates and/or a representative document sample. In a peer-to-peer information retrieval network, the peers need to know to what other peers they can send a query. Hence, resource descriptions are also needed. The advantage of peer-to-peer networks is that peers can be cooperative and use a designed and agreed upon protocol, making exchange of resource descriptions easier. However, peers may have incentive to cheat about their content, which creates challenges unique to peer-to-peer networks.

Secondly, there is the *collection selection problem*: after acquiring resource descriptions, the next step is selecting a subset of search servers that can handle the query. When the mediator receives a new query from a client it can quickly score it locally against the acquired resource descriptions to determine the servers most likely to yield relevant search results for it. The algorithms for determining the best servers in federated information retrieval can be divided in two groups. Firstly, those that treat resource descriptions as big documents without considering individual documents within each resource: CORI, CVV and KL-Divergence based (Callan et al., 1995; Yuwono and Lee, 1997; Xu and Croft, 1999). Secondly, those that do consider the individual documents within each resource: GlOSS, DTF, ReDDE and CRCS (Gravano et al., 1999; Nottelmann and Fuhr, 2007; Si and Callan, 2003a; Shokouhi, 2007). Although considering individual documents gives better results, it also increases the complexity of resource descriptions and the communication costs. Additionally, most existing resource selection algorithms are designed for use by a single mediator party making them difficult to apply in a network with, for example, aggregated local indices. Resource selection according to the unique characteristics of peer-to-peer networks requires new algorithms (Lu, 2007).

Thirdly, there is the *result merging problem*: once the mediator has acquired results from several search servers these need to be merged into one coherent list. If all servers would use the same algorithm to rank

their results this would be easy. However, this is rarely the case and exact ranking scores are commonly not included. The first step in merging is to normalise the scores globally, so that they are resource independent. In federated information retrieval CORI or the SemiSupervised Learning (SSL) merging algorithm can be used for this (Si and Callan, 2003b). However, in peer-to-peer environments the indexed document collections often vary widely in their sizes, which makes CORI unlikely to work well. SSL requires a sample database which makes it undesirable in peer-to-peer networks cautious about bandwidth usage. An alternative approach is to recalculate document scores at the mediator as done by Kirsch's algorithm (Kirsch, 1997) which is quite accurate and has low communication costs by only requiring each resource to provide summary statistics. However, this also requires knowledge of global corpus statistics that is costly to obtain in peer-to-peer networks with local indices. Result merging in peer-to-peer information retrieval networks requires an algorithm that can work effectively with minimal additional training data and communication costs, for which none of the existing algorithms directly qualifies. Result merging in existing networks has so far relied on simple frequency-based methods, and has not provided any solution to relevance-based result integration (Lu, 2007).

Differences

The first noticeable difference with peer-to-peer information retrieval is the strict specialisation of the various parties. The clients only issue queries whereas the search servers only serve search results. This also determines the shape of the rigid overlay network that forms: a graph with clients on one side, servers on the other side and the mediator in the middle, as shown in Figure 2.3. Indeed, federated information retrieval is much closer to the conventional client-server paradigm and commonly involves machines that already 'know' each other. This contrasts with peer-to-peer networks where peers take on these roles as needed and frequently interact loosely with 'anonymous' other machines. Additionally, a peer-to-peer network is subject to significant churn, availability and heterogeneity problems that only mildly affect federated information retrieval networks due to the strict separation of concerns (Lu, 2007).

A second difference is the presence of the mediator party. To the clients the mediator appears as one entry point and forms a façade: clients are

never aware that multiple search servers exist at all. This has the implication that all communication is routed through the mediator which makes it a single point of failure. In practice a mediator can be a server farm to mitigate this. However, it still remains a single point of control, similar to completely centralised search systems, which can create legal and ethical difficulties. A peer-to-peer network with one central 'mediator' point for routing queries is conceptually close to a federated information retrieval network (Lu, 2007). However, most peer-to-peer networks lean towards distributing this mediation task, mapping queries to peers that can provide relevant results, over multiple peers.

2.6.3 *Challenges*

In Section 2.2 we have already seen some challenges that apply to peer-to-peer networks in general. In this subsection we discuss a subset of these aspects more important to peer-to-peer information retrieval.

Latency

In peer-to-peer information retrieval, latency is dominated by the number of peers involved in routing and processing queries. We have seen that local and global indices are suitable for different types of queries and that there are many optimisations that can be applied to reduce the cost of storing and transmitting index information. Nevertheless, the challenge of optimally combining these techniques, and finding new ones, to keep latency within acceptable bounds remains. The reason for this is primarily that there is no one good solution for all cases and that the increasing amount of information to index is leading to greater latency problems. For any search system, it is important to serve search results quickly *without* compromising too much on result quality. The technical causes of delays are irrelevant to users. After entering a query, the results should appear in at most 2 seconds, but ideally instantaneously in terms of perception, which means a delay of 0.1 seconds. Anything below that is unlikely to positively impact the user experience (Nah, 2004). Most existing solutions rightly focus on reducing the number of hops or using parallelisation to reduce latency. Efficient query routing is a challenge specific to peer-to-peer information retrieval and directly tied to latency. More research in temporal aspects of querying could lead to more optimal tailored solutions.

Freshness

Keeping the index fresh is a challenge for every search engine, which is commonly the responsibility of the engine's web crawling component. The index needs to be representative of the indexed websites, without incurring too much load on those sites to detect changes. Some web documents change quickly and some change rarely, and not every change that occurs is significant enough to warrant an index update (Risvik and Michelsen, 2002). In the ideal situation, websites participate cooperatively in a peer-to-peer network and signal significant changes to themselves to the network. This would remove the need for crawling. However, peer-to-peer web search engines will initially have to cope with the existing situation. Having peers perform their own crawl seems realistic, but introduces the same problems seen in conventional web crawling. Since many updates can occur due to changing documents, it is important that the index used has minimal mutation overhead. Separate indexing strategies could be used for fast and slow changing documents. A further challenge is caching of postings lists or search results. These mechanisms decrease latency, but do so at the expense of freshness.

Evaluation

Even though a simulation can fix many of the free variables of a peer-to-peer network, for rigorous comparison the same data needs to be used. There is a need for a common collection, a way to distribute this collection over multiple peers and a query set. There have been at least two attempts at establishing such a benchmark (Klampanos et al., 2005; Neumann et al., 2006), although they have not yet seen widespread adoption. Klampanos et al. (2005) state that evaluating peer-to-peer information retrieval systems is a demanding and neglected task. They establish a number of different document test beds for evaluating these systems. They state that evaluation of these networks is hard for several reasons. Firstly, they are assumed to be very large which makes simulation difficult. Secondly, they are subject to churn caused by normal peer on-off cycles and peers that crash or malfunction. Unfortunately, the impact of churn is not well investigated in peer-to-peer information retrieval experiments, as most assume an always-on environment (Zeinalipour-Yazti et al., 2004). Thirdly, documents are not likely to be randomly placed at peers, instead their distribution is influenced by previous location, prior

retrieval and replication. Lastly, simulating user behaviour is complex, for example: realistically simulating how both collections and query frequencies change over time. This is usually circumvented by reflecting behaviour in the document distribution. However, it is difficult to reflect the application scenario such that the results can be conclusive.

Different types of peer-to-peer information retrieval networks have different document distributions. Klampanos et al. (2005) present standardised distributions for three of these derived from the WT10g collection (Bailey et al., 2003) using about 1.7 million documents. Firstly, the web domain where the distribution of documents follows a power-law. Secondly, loosely controlled grid networks with a uniform distribution of documents that impose an equal load on each peer. Thirdly, digital libraries where the distribution also follows a power law, although less extreme than for the web domain. Additionally, digital libraries have fewer peers that each share a significantly larger amount of documents compared to the other cases. Replication is simulated in all cases by exploiting inter-domain links. The web and grid scenarios use about 11,680 peers, whereas 1,500 are used for the digital library case. Lu and Callan (2003) also present a test bed for digital libraries with 2,500 peers based on WT10g, and later also one with 25,000 peers based on the Gov2 collection (Lu, 2007; Clarke et al., 2004). Gov2 splits have also been used in the federated information retrieval setting (Fallen and Newby, 2006; Thomas and Hawking, 2007).

3

EXISTING RESEARCH & SYSTEMS

'There is a single light of science
and to brighten it anywhere is
to brighten it everywhere.'

Isaac Asimov

Peer-to-peer information retrieval has been an active research area for about a decade. In this chapter we first reveal its main focus, followed by an in-depth examination of optimisation techniques, an overview of existing information retrieval systems and its key challenges. This is followed by research on economics in peer-to-peer networks and the introduction of our own system architecture.

A practical view on the goal of peer-to-peer information retrieval is minimising the number of messages sent per query while maintaining high recall and precision (Zeinalipour-Yazti et al., 2004). There are several approaches to do this which represent trade-offs. Let us start with the two common strategies to partition indices over multiple machines: *partition-by-document* and *partition-by-keyword* (Li et al., 2003). In partition-by-document each peer is responsible for maintaining a local index over a specific set of documents: the postings for all terms of a particular document are located at one specific peer. In some cases the

This chapter is based on Tigelaar et al. (2012): *Peer-to-Peer Information Retrieval: An Overview*, that appeared in *ACM Transactions on Information Systems*, Volume 32, Issue 2 (May 2012). ©ACM, 2012. http://doi.acm.org/10.1145/2180868.2180871.

documents themselves are also stored at that peer, but they need not be. The strict and aggregated local indices architectures are commonly used in peer-to-peer networks that use this partitioning. In contrast, in partition-by-keyword each peer is responsible for storing the postings for some specific keywords in the index. A natural architecture for this is the distributed global index.

An early investigation into the feasibility of peer-to-peer web search was done by Li et al. (2003). They view partition-by-document as a more tractable starting point, but show that partition-by-keyword can get close to the performance of partition-by-document by applying various optimisations to a distributed global index. In contrast, Suel et al. (2003) conclude that partition-by-document approaches scale poorly, because document collections do not 'naturally' cluster in a way that allows query routing to a small fraction of peers and thus each query requires contacting nearly all peers. Perhaps due to this paper much of the research in peer-to-peer information retrieval has focused on partition-by-keyword using a distributed global index (Klampanos and Jose, 2004).

Unfortunately, a distributed global index is not without drawbacks since it is intended for performing efficient lookups, not for efficient search (Bawa et al., 2003). Firstly, a distributed hash table provides load balancing rather naively, by resorting to the uniformity of the hash function used (Triantafillou et al., 2003). As term posting lists differ in size, hotspots can emerge for popular terms, which debalances the load. Secondly, the intersection of term posting lists ignores the correlations between terms, which can lead to unsatisfactory search accuracy (Lu, 2007). Thirdly, the communication cost for an intersection grows proportionally with the number of query terms and the length of the inverted lists. Several optimisations have been proposed like storing multiterm query results for terms locally to avoid intersections and requiring peers to store additional information for terms strongly correlated with the terms they already store. The choice of resource descriptions in a distributed global index is thus limited by the high communication costs of index updates: full-text representations are unlikely to work well due to the massive network traffic required. Fourthly, skewed corpus statistics, as a result of term partitioning, may lead to globally incomparable ranking scores. Finally, distributed hash tables are vulnerable to various network attacks that compromise user security and privacy as well as the authenticity of data (Steiner et al., 2007).

Many authors fail to see a number of benefits unique to partition-by-document local indices, such as the low costs for finding popular items, advanced query processing, inexpensive index updates and high churn resilience. Admittedly, the primary challenge for such indices is routing the query to suitable peers. Our stance is that both approaches have their merit and complement each other. Recent research indeed confirms the effectiveness of using local indices for popular query terms and a global index for rare query terms (Rosenfeld et al., 2009).

Li et al. (2003) conclude that web-scale search is not possible with peer-to-peer technology. They indicate that compromises need to be made in either the quality of search results or the structure of the network. According to them, the overhead introduced by communication between peers is too large to offer reasonable query response times given the capacity of the Internet. However, much work, discussed in the next section, has been done since their paper and the nature and capacity of the Internet have changed significantly in the intervening time.

Yang et al. (2006) compare the performance of several peer-to-peer architectures for information retrieval combined with common optimisations. They test three approaches: a distributed global index augmented with Bloom filters and caching; aggregated local indices with query flooding; and strict local indices using random walks. All of these are one-step term-document indices. Interestingly, they all consume approximately *the same amount* of bandwidth during query processing, although the aggregated local indices are the most efficient. However, the distributed global index offers the lowest latency of these three approaches, closely followed by aggregated local indices and strict local indices being orders of magnitude slower. For all approaches the forwarding of queries in the network introduces the most latency, while answering queries is relatively inexpensive. Even though the distributed global index is really fast, its major drawback rears its ugly head at indexing and publishing time. Adding new documents to the network requires updating the posting lists, which takes six times as much bandwidth and nearly three times as much time compared to the aggregated local indices. Strict local indices resolve all this locally and incur no costs in terms of time or bandwidth for publishing documents. This study clearly shows that an architecture should achieve a balance between *retrieval speed* and *update frequency*.

3.1 OPTIMISATION TECHNIQUES

In this section we discuss several optimisation approaches. There are two reasons to use these techniques. One is to reduce bandwidth usage and latency, the other is to improve the quality and quantity of the search results returned. Most techniques discussed influence both of these aspects and offer trade-offs, for example: one could compromise on quantity to save bandwidth and on quality to reduce latency.

3.1.1 *Approximate Intersection of Posting Lists with Bloom Filters and Min-Wise Independent Permutations*

(Cuenca-Acuna et al., 2003; Reynolds and Vahdat, 2003; Suel et al., 2003; Zhang and Suel, 2005; Michel et al., 2005a, 2006)

When using a distributed global index, a multiterm query requires multiple lookups in the distributed hash table. The posting lists for all terms need to be intersected to find the documents that contain all query terms. Exchanging posting lists can be costly in terms of bandwidth, particularly for popular terms with many postings, thus smaller Bloom filters derived from these lists can be transferred instead. Reynolds and Vahdat (2003) were the first to use Bloom Filters in the context of peer-to-peer information retrieval.

A Bloom filter is an array of bits. Each bit is initially set to zero. Two operations can be carried out on a Bloom filter: inserting a new value and testing whether an existing value is already in the filter. In both cases k hash functions are first applied to the value. An insert operation, based on the outcome, sets k positions of the Bloom filter to one. Membership tests read the k positions from the Bloom filter. If all of them equal one the value *might be* in the data set. However, if one of the k positions equals zero the value *is certainly not* in the data set. Hence, false positives are possible, but false negatives never occur (Bloom, 1970; van Heerde, 2010, p. 82). Bloom filters are an attractive approach for distributed environments because they achieve smaller messages which leads to huge savings in network I/O (Zeinalipour-Yazti et al., 2004).

Consider an example in the peer-to-peer information retrieval context: peer Q poses a query consisting of terms a and b. We assume that term a has the longest posting list. Peer A holds the postings P_a for term a, derives a Bloom filter F_a from this and sends it to peer B that contains

the postings P_b for term b. Peer B can now test the membership of each document in P_b against the Bloom filter F_a and send back the intersected list $P_b \cap F_a$ to peer Q as final result. Since this can still contain false positives, the intersection can instead be sent back to peer A, that can remove false positives since it has the full postings P_a. The result is then $P_a \cap (P_b \cap F_a)$: the true intersection for terms a and b, which can be sent as result to peer Q. Bandwidth savings occur when sending the small F_a instead of the large P_a from peer A to B. However, this approach requires an extra step if one wants to remove the false positives (Reynolds and Vahdat, 2003).

False positives are the biggest drawback of Bloom filters: the fewer bits used, the higher the probability a false positive occurs. Large collections require more bits to be represented than smaller ones. Unfortunately, Bloom filters need to have the same size for the intersection and union operations to work. This makes them unsuitable for peer-to-peer networks in which the peers have collections that vary widely in the number of contained documents.

Bloom filters can be used to perform approximate intersection of posting lists. However, as a step prior to that it is also interesting to estimate what an additional posting list would do in terms of intersection to the lists already obtained. This task only requires cardinality estimates and not the actual result of an intersection. While Bloom filters can be used for this, several alternatives are explored by Michel et al. (2006). The most promising is Min-Wise Independent Permutations (MIPs). This requires a list of numeric document identifiers as input values. Firstly, this method applies k linear hash functions, with a random component, to the values each yielding a new list of values. Secondly, the resulting k lists are all sorted, yielding k permuted lists, and the minimum value of each of these lists is taken and added to a new list: the MIP vector of size k. The fundamental insight is that each element has the same probability of becoming the minimum element under a random permutation. The method estimates the intersection between two MIP vectors by taking the maximum of each position in the two vectors. The number of distinct values in the resulting vector divided by the size of that vector forms an estimate of the overlap between them. The advantage is that even if the input vectors are of unequal length, it is still possible to use only the first few positions to get a, less accurate, approximation. Michel et al. (2006) show that MIPs are much more accurate than Bloom filters for this type of estimation.

3.1.2 *Reducing the Length of Posting Lists with Highly Discriminative Keys*

(Skobeltsyn et al., 2009; Luu et al., 2006)

An alternative way of reducing the costs of posting list intersection for a distributed global index is by making the lists themselves shorter. To achieve this, instead of building an index over single terms, one can build one over entire multiterm queries. This is the idea behind highly discriminative keys. No longer are all terms posted in a global distributed index, but instead multiterm queries are generated from a document's content that discriminate that document well from others in the collection. The result: more postings in the index, but shorter posting lists. This offers a solution to one of the main drawbacks of using distributed hash tables: intersection of large posting lists.

3.1.3 *Limiting the Number of Results to Process with Top k Approaches*

(Tang et al., 2002; Cuenca-Acuna et al., 2003; Suel et al., 2003; Tang and Dwarkadas, 2004; Balke et al., 2005; Michel et al., 2005a; Zhang and Suel, 2005; Skobeltsyn and Aberer, 2006; Skobeltsyn et al., 2007a, 2009)

Processing only a subset of items during the search process can yield performance benefits: less data processing and lower latency. Various algorithms, discussed shortly, can be used to retrieve the top items for a particular query without having to calculate the scores for all the items. Retrieving top items makes sense as it has been shown that users of web search engines prefer quality over quantity with respect to search results: more precision and less recall (Oulasvirta et al., 2009). Top k approaches have been applied to various architectures and at various stages in peer-to-peer information retrieval:

Top k results requesting

A simple optimisation is requesting only the top results. Approaches that use local indices always apply a form of limited result requesting implicitly by bounding the number of hops made during flooding or by performing a random walk that terminates. However, that number can also be explicitly set to a constant as Cuenca-Acuna et al. (2003) do for their globally replicated index. They first obtain a list of k search results and keep contacting nodes as long as the chance of them contributing to this top k remains high. The top results stabilise after a few rounds.

Top k query processing

This approach has its roots in the database community, particularly in the work of Fagin et al. (2001). Several variations exist, all with the same basic idea: we can determine the top k documents given several input lists without having to examine these lists completely and while not adversely affecting performance. This is often used in cases where a distributed global index is used and posting lists have to be intersected. The *threshold algorithm* is the most popular top k processing approach (Michel et al., 2005a; Suel et al., 2003).

The threshold algorithm maintains two data structures: a queue with peers to contact for obtaining search results and a list with the current top k results. Peers in the queue are processed one by one, each returning a limited set of k search results of the form $(document, score)$ sorted by score in descending order. For a distributed global index these are the top items in the posting list for a particular term. The algorithm tracks two scores for each unique document: worst and best. The worst score is the sum of the scores for a document d found in all result lists in which d appeared. The best score is the worst score plus the lowest score (of some other document) encountered in the result lists in which d did not appear. Since all the result lists are truncated, this last score forms an upper bound of the best possible score that would be achievable for document d. The current top k is formed by the highest scoring documents seen so far based on their worst score. If the best score of a document is lower than the threshold, which is the worst score of the document at position k in the current top k results, it need not be considered for the top k. The algorithm bases the final intersection on only the top k results from each peer, that provably yields performance equivalent to 'sequentially' intersecting the entire lists. This saves both bandwidth and computational costs without negatively affecting result quality.

A drawback of the threshold algorithm is that looking up document scores requires random access to the result lists (Suel et al., 2003). Zhang and Suel (2005) later investigated the combination of top k query processing with several optimisation techniques. They draw the important conclusion that different optimisations may be appropriate for queries of different lengths. Balke et al. (2005) show that top k query processing can also be effective in peer-to-peer networks with aggregated local indices.

Top k result storing

One step further is only *storing* the top k results for a query, or term, in the index. Skobeltsyn and Aberer (2006) take this approach as a means to further reduce traffic consumption. Related to this is the work of Tang and Dwarkadas (2004) that store postings only for the top terms in a document. They state that while indexing only these top terms might degrade the quality of results, it likely does not matter since such documents would not rank high for queries for the other terms they contain anyway.

3.1.4 *Involving Fewer Peers During Index Look-ups by Global Replication*

(Cuenca-Acuna et al., 2003; Galanis et al., 2003)

Lookups to map queries to peers are expensive when they involve contacting other peers, regardless of the architecture used. What if a peer can do all lookups locally? The authors of the PlanetP system explore this approach (Cuenca-Acuna et al., 2003). They replicate a full global index at each peer: a list of all peers, their IP address, current network status and their Bloom filters for terms. This information is spread using *gossip*. If something changes at a peer it gossips the change randomly to each of its neighbours until enough neighbouring peers indicate that they already know about the rumour. Each peer that receives rumours also spreads it in the same way. There is the possibility that a peer misses out on a gossip; to cope with this the authors periodically let peers exchange their full directory and they also piggyback information about past rumours on top of new ones. Whilst this is an interesting way to propagate index information, it is unfortunately also slow: it takes in the order of hundreds of seconds for a network of several thousand peers to replicate the full index information at each peer. This approach is not widely used and is best suited to networks with a small number of peers due to scalability issues (Zeinalipour-Yazti et al., 2004).

Although we prefer to label this approach as a (replicated) global index, it can also be viewed as a very extreme form of aggregation where each peer holds aggregate data on every other peer in the network. Note that this approach differs from a single-hop distributed hash-table, since it uses no hashing and no distributed key space. Hence, the topology of the network is not determined by a key space.

3.1.5 *Reducing Processing Load by Search Result Caching*

(Reynolds and Vahdat, 2003; Skobeltsyn and Aberer, 2006; Skobeltsyn et al., 2007a; Zimmer et al., 2008; Skobeltsyn et al., 2007b, 2009; Tigelaar and Hiemstra, 2011; Tigelaar et al., 2011)

It makes little sense to reconstruct the search result set for the same query over and over again if it does not really change. Performance can be increased significantly by caching search results. The effectiveness of exploiting usage data to boost performance by caching for centralised search engines has been shown previously (Fagni et al., 2006; Baeza-Yates et al., 2007b; Lempel and Moran, 2003). Skobeltsyn and Aberer (2006) use a distributed hash table to keep track of peers that have cached relevant search results for specific terms. Initially this table is empty, and each (multiterm) query is first broadcast through the entire peer-to-peer network, using a shower broadcast[1] with costs $\mathcal{O}(n)$ for a network of n peers. After this step the peer that obtained the search results registers itself as caching in the distributed hash table for each term in the query. This allows for query subsumption: returning search results for subsets of the query terms in the absence of a full match. The authors base the content of the index on the queries posed within the network, an approach they term *query-driven indexing*. This significantly reduces network traffic for popular queries while maintaining a global result cache that adapts in real-time to submitted queries.

3.1.6 *Involving fewer Peers during Query Processing by Clustering*

(Bawa et al., 2003; Sripanidkulchai et al., 2003; Crespo and Garcia-Molina, 2004; Klampanos and Jose, 2004; Akavipat et al., 2006; Klampanos and Jose, 2007; Lu and Callan, 2007; Nguyen et al., 2008; Lele et al., 2009; Papapetrou et al., 2010; Tirado et al., 2010)

When using local indices, keeping peers with similar content close to each other can make query processing more efficient. Instead of sending a query to all peers it can be sent to a cluster of peers that covers the query's topic. This reduces the total number of peers that need to be contacted for a particular query. Unfortunately, content-based clustering does not occur naturally in peer-to-peer networks (Suel et al., 2003). Hence, Bawa et al. (2003) organise peer-to-peer networks by using topic segmentation.

1 A broadcasting method that visits each peer only once, described in Datta et al. (2005)

They arrange the peers in the network in such a way that only a small subset of peers, that contain matching relevant documents, need to be consulted for a given query. Clustering peers is performed based on either document vectors or full collection vectors. They then use a two-step process to route queries based on the topic they match. They first find the cluster of peers responsible for a specific topic and forward the query there. After this the query is flooded within the topical cluster to obtain matches. They conclude that their architecture provides good performance, both in terms of retrieval quality and in terms of latency and bandwidth. Unfortunately, their system requires a central directory for finding an initial topic cluster for query routing. Akavipat et al. (2006) show how to do clustering without a central directory.

Klampanos and Jose (2007) evaluate cluster-based approaches for large-scale peer-to-peer information retrieval, focusing on single-pass clustering with both a variable and fixed number of clusters. They find that the predominantly small size of web documents makes them hard to relate to other documents thereby leading to poor clustering. Clustering mechanisms fail to discover the structure of the underlying document distribution, leading to the situation where not enough relevant sources can be contacted to route a query to. This is due to the loss of information inherent in the creation of cluster centroids. They propose two solutions. Firstly, replicating documents part of popular clusters on multiple peers, leading to a significant improvement in effectiveness. Although this does not solve the problem for unpopular topics, it could work sufficiently well for most users. Secondly, assuming a relevance feedback mechanism exists and using this to alter the centroids of the global topic clusters. The weight of each term in a cluster is then determined by the relevance of that cluster to the query based on the feedback. They show the usefulness of both replication and relevance feedback which lead to better query routing and higher precision, emphasising relevance feedback as a promising evolution in peer-to-peer information retrieval.

Interest-based clustering works either by shortening the path lengths between peers with similar interest, meaning: peers that pose similar queries, or by bringing peers with a particular interest closer to peers with matching content. Although not exactly the same, both aim to reduce the number of hops needed for obtaining relevant content. In the first case by leveraging cached information present at peers with similar interests: caches at other consuming peers, while the second case brings one closer

to the origin of information: providing peers that contain original content (Sripanidkulchai et al., 2003; Akavipat et al., 2006). The two clustering approaches: by content and by interest, can also be combined.

3.1.7 *Reducing Latency and Improving Recall using Random Walks*

(Lv et al., 2002; Yang et al., 2006)

Peer-to-peer systems with local indices are conventionally searched with query flooding. That approach is theoretically exhaustive, but because of tractability it is applied in a non-exhaustive way by bounding the number of hops. Lv et al. (2002) propose an alternative to this by using random walks. Instead of searching in a breadth-first manner by forwarding the queries to all neighbours, we search depth-first by forwarding the query only to one neighbour. Such a walk originates from the querying peer and randomly steps forward through the network. Peers that have relevant results send these back directly to the originating peer. Peers participating in the walk occasionally check the satisfaction of the originating peer with respect to the number of results obtained so far and terminate the walk based on this. Yang et al. (2006) find that this approach is slow, but multiple walks can be started in parallel to decrease the latency. Similar to random walks Kalogeraki et al. (2002) propose to forward query messages to a randomly selected fraction of neighbouring peers. However, this still increases messaging costs exponentially when increasing the fraction, whereas for random walkers this remains linear.

3.1.8 *Reducing Latency and Improving Recall using Directed Walks*

(Adamic et al., 2001; Joseph, 2002; Kalogeraki et al., 2002; Yang and Garcia-Molina, 2002; Tsoumakos and Roussopoulos, 2003; Zhong et al., 2003; Zeinalipour-Yazti et al., 2004; Li et al., 2009; Song et al., 2010)

Adamic et al. (2001) route query messages via high-degree nodes, those with high connectivity, and show that this both decreases search time and increases the network penetration. Yang and Garcia-Molina (2002) forward query messages to peers that previously returned the most query results. In a similar vein Tsoumakos and Roussopoulos (2003) introduce adaptive probabilistic search where each peer maintains a probabilistic routing table for each query that either originated from it or travelled through it. The initial peer that submits a query broadcasts it to all its neighbours, but from there on the query message is forwarded only to

Table 3.1: Selection of Peer-to-Peer Information Retrieval Systems Literature

Name	References
DCT / ALVIS	Buntine et al. (2005); Luu et al. (2006)
	Skobeltsyn and Aberer (2006); Skobeltsyn et al. (2007a, 2009)
DHI / SPINA	Di Buccio et al. (2009)
DTF	Fuhr (1999); Nottelmann and Fuhr (2007)
pSearch / eSearch	Tang et al. (2002); Tang and Dwarkadas (2004)
MINERVA	Bender et al. (2005b); Chernov et al. (2005); Bender et al. (2006)
	Michel et al. (2006); Bender et al. (2007); Zimmer et al. (2008)
NeuroGrid	Joseph (2002)
ODISSEA	Suel et al. (2003); Zhang and Suel (2005)
PHIRST	Rosenfeld et al. (2009)
PlanetP	Cuenca-Acuna et al. (2003)
SETS	Bawa et al. (2003)
Sixearch	Akavipat et al. (2006); Menczer et al. (2008); Lele et al. (2009)

the neighbour that has the highest probability of obtaining results based on past feedback. Zeinalipour-Yazti et al. (2004) build upon this and propose a mechanism where peers actively build profiles of neighbouring peers based on the most recent queries they responded to, similar to Joseph (2002). A peer scores incoming queries against the profiles of its neighbouring peers, ranking them both qualitatively: based on their cosine similarity, and quantitatively: based on the number of previously returned results. This outperforms basic flooding, random forwarding (Kalogeraki et al., 2002) and pure quantitative directed routing (Yang and Garcia-Molina, 2002).

3.2 INFORMATION RETRIEVAL SYSTEMS

Many peer-to-peer information retrieval systems have been developed for various applications. These systems often borrow elements from file sharing networks and federated information retrieval with various levels of success. Most research systems focus on either the domain of computing grids, digital libraries or the web.

Table 3.1 lists references to literature that describes major research systems developed. Figure 3.1 shows a classification breakdown of these peer-to-peer information retrieval systems and other discussed systems.

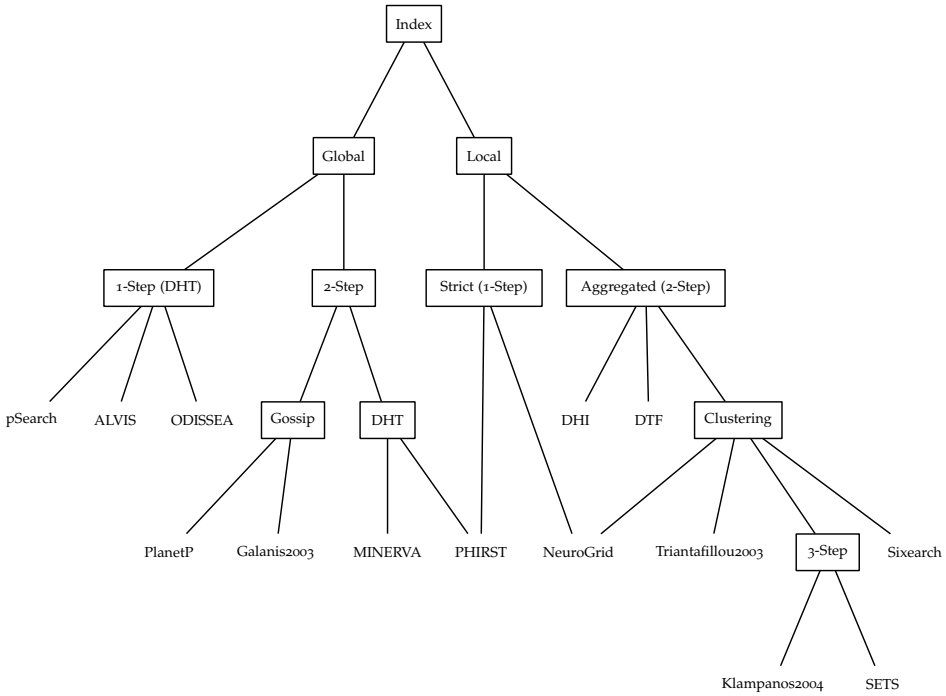


Figure 3.1: Classification of peer-to-peer information retrieval research systems. See Section 2.4 and Section 3.1 for an explanation of the rectangular distinctions. Clustering in this diagram means explicit interest-based or content-based clustering and not the random clusters that can occur naturally when using aggregated local indices.

3.2.1 *Scientific Systems*

Although many research systems exist, we restrict ourselves to a subset of them in this section. We discuss systems that stand out because of their pioneering nature or by use of an interesting mix of techniques.

Sixearch

One of the first peer-to-peer information retrieval systems was the Infrasearch project, that later became JXTASearch (Waterhouse et al., 2002; Klampanos and Jose, 2004) which is also the basis for Sixearch (Akavipat et al., 2006; Lele et al., 2009). Sixearch consists of several components: a topical crawler, a document indexing system, a retrieval engine, JXTA for peer-to-peer network communication and a contextual learning system. They use an XML-based architecture and assume that the query consists of a structured customisable set of fields. A book collection could, for example, have the fields: title, author, et cetera. This approach is not geared well towards full-text retrieval since it is based on the structure of queries rather than that of the content shared (Klampanos and Jose, 2004). Supplier peers publish, for each query field, a set of keywords for which they believe they can provide relevant results: their resource description. Consumer peers pose structured queries that are routed to appropriate supplier peers using hubs. The query routing bases itself on content profiles of neighbouring peers that are continually improved using reinforcement learning based on past interactions. Fusion of search results returned by multiple peers uses a simple voting algorithm. The authors want to improve their system by focusing on contextual learning and social collaboration. They intend to extend their design with a reputation system as a security component to distinguish spammers from honest peers (Menczer et al., 2008).

ODISSEA

Suel et al. (2003) introduce the ODISSEA peer-to-peer architecture. Their system consists of two tiers. The lower tier consists of peers that maintain a distributed global index. The postings for a term are always located at a single peer. The upper tier consists of update peers that insert or update documents in the system, like a crawler or a web server, and query peers that use the lower tier to answer queries. The novelty in their

approach is both in the specialisation of peers as well as in their usage of a distributed global term-document index. The specialisations make that in their system the peers responsible for storing, constructing and querying the index are in fact disjunct. This resembles the completely centralised approach, commonly used by modern search engines, where some machines just store documents in the index, some crawl to keep the index fresh and (external) others only query. In contrast to those systems, ODISSEA offers an open indexing and searching infrastructure in which every machine that 'speaks' the protocol can participate as peer.

When handling multiterm queries, the posting list intersections are conducted in ascending order of term posting list size: from small to large, as this greatly reduces the amount of data that needs to be transferred. Furthermore, they apply top k query processing to minimise bandwidth usage. The authors suggest optimisation of query execution, compression and pruning techniques as important future work. Furthermore, they state that web-scale information retrieval is a much more challenging application than file sharing.

MINERVA

Bender et al. (2005b) assume that each peer performs its own crawls and builds a local index. A peer first searches its own local index to find relevant search results. If these results are unsatisfactory, the peer can consult a global distributed index that contains for each term in the network a list of peers that have relevant documents: a two-step index. This global index also contains statistics regarding the local indices maintained by each peer. The authors show that properly estimating the overlap between search results can reduce the number of peers that need to be contacted for complete recall by more than 60 percent. However, the lookups in a distributed hash table remain expensive. Bender et al. (2006) propose to use correlations among individual terms in a query to reduce the number of lookups: the peer that handles the first term in the query also, adaptively, stores what peers to contact for the remaining query terms. This significantly reduces the number of involved peers, while sustaining the same level of recall. Nevertheless, popular terms can cause severe load imbalances if a single peer bears responsibility for storing all postings for one term. Michel et al. (2005b) propose creating one-step term-document indices in MINERVA for popular terms to reduce response times. Since

posting lists are usually scanned sequentially, from the best to worst scoring document for a particular term, they use an order-preserving hash function[2] to store the postings for a term sorted by descending score over multiple peers. The authors apply top k query processing to further reduce load. This can be further optimised by applying search result caching (Zimmer et al., 2008): storing cached search results for each complete query on peers that store the postings for one of the query terms. These results contain meta information that helps in judging whether they are still fresh enough and whom to contact for refreshed results. The authors show that cycling out the least frequently used item is the best cache management strategy for a bounded cache. They experiment with both exact caching: matching a multiterm query exactly, and approximate caching: matching term subsets of a multiterm query. They find that both approaches save valuable network resources without compromising result quality.

In later work Bender et al. (2005a) consider the *novelty* of additional results, in addition to the quality, using a modified federated search collection selection algorithm. This also appears in Michel et al. (2006) who focus on further optimising query routing. Furthermore, they experiment with Bloom filters and Min-Wise Independent Permutations, showing the latter is better for obtaining result set size estimates.

ALVIS

Buntine et al. (2005) and Luu et al. (2006) introduce the ALVIS Peers system: a distributed global index approach, with several innovations. During final result fusion each peer that generated an index entry is contacted and asked to recompute the document score based on global and local statistics, thereby generating globally comparable scores. Instead of storing postings for individual terms, the authors use highly discriminative keys. This introduces the problem of having to store many more keys than in a conventional term-peer index. To mitigate this, in later work (Skobeltsyn et al., 2007a, 2009) they combine their approach with query-driven indexing storing only popular keys in the index and apply top k result storing. While this has a penalty for less popular, long-tail queries, Shokouhi et al. (2007b) showed that query logs can be used to prune irrelevant keys from an index without much performance loss.

2 Such a function guarantees that if $a > b$, then $\text{hash}(a) > \text{hash}(b)$.

PHIRST

The differences between global and local indices give rise to a difficult trade-off. We have to choose between fast, but costly and inflexible exact search or slow, but cheap and flexible approximate search. Rosenfeld et al. (2009) present an approach to peer-to-peer full-text search that combines global and local indices for different types of terms. They keep only the low-frequency terms in a hash table, while estimating the counts for common terms via limited query flooding. Newly added documents likely contain more well-known highly frequent terms and less new low-frequency terms. Because of this effect they claim that their approach leads to a proportionally smaller index as the number of indexed documents and peers increases compared to a full index kept in a distributed hash table. Loo et al. (2004) and Huebsch et al. (2005) already showed that this hybrid approach improves recall and response times and incurs less bandwidth overhead for search in file sharing.

Klampanos2004

Klampanos and Jose (2004) attempt to apply standard information retrieval approaches in a peer-to-peer environment. They combine aggregated local indices with content clustering. They assume that each peer indexes its own documents and finds content clusters in its own collection. At the network level each peer joins one or more content-aware groups based on its local clusters. The content-aware groups are potentially overlapping clusters of peers. Each super peer, referred to as hub in their paper, stores the descriptors of these groups and given a query can score it against them. The descriptors are simple term frequency vectors. A simplified version of Dempster-Shafer theory, a way to combine evidence from multiple sources into one common belief, is used to fuse results from multiple peers at the super peers. This seems to perform well, while contacting few peers: usually one or two, sometimes three and rarely six. The overall system offers recall and precision levels that substantially exceed centralised search. They experimented with 560,000 documents, from the TREC 6 and 7 ad-hoc track, divided over 1,500 peers and used the Managing Gigabytes (MG) system for their experiments (Witten et al., 1999).

NeuroGrid

Joseph (2002) introduces an adaptive decentralised search system called NeuroGrid that uses hybrid indexing: initially all the peers have their own local document index, but when they join the network they create a peer index of their neighbouring peers. This closely resembles aggregated local indices, but with each peer functioning as a super peer. Initially a NeuroGrid network is a simple message flooding network. The novelty in the approach is in the adaptive routing of queries. User responses to search results, the absence of positive feedback or explicit negative feedback, are recorded. When NeuroGrid has to select a subset of peers to forward a query to, it tries to maximise the chance of receiving positive feedback for the returned results based on these previous experiences. In case of positive feedback the querying peer establishes a direct link to the responding peer in the overlay network. This type of clustering gradually increases connectivity and makes all peers become more knowledgeable concerning the content of their neighbours. This approach also reduces the length of the path that queries need to travel over time. The system prefers reliable peers: those that respond to queries and supply on-topic results of interest to the user. Well-connected peers have more influence on the statistical learning process.

Galanis2003

Galanis et al. (2003) propose organising all the data sources on the Internet in a large peer-to-peer network where queries are answered by relevant sites. They assume that each peer is essentially an XML search engine that maintains a local index. When a peer joins the network, it sends other peers a summary of its data: a small set of selected tags and keywords representative for its content. A join thus generates a wave of messages, making their approach geared towards networks with low churn. Alternatively, such information can also be piggybacked when sending queries as in Di Buccio et al. (2009). Peers initially acquire content summaries of others peers in the system via neighbouring peers and maintain their own peer index. The authors examine the effect of replicating summaries to every peer and to peer subsets of various sizes. They experiment with networks of 100 and 200 peers based on crawled eBay XML data. Their results suggest that using replication to every peer outperforms that of using subsets, although using large subsets can approach

this performance level. They compare full replication aggregated indices with a strict local indices approach and show aggregation increases query throughput with 2,071 percent and offers 72 times faster response times.

Triantafillou2003

Triantafillou et al. (2003) focus on enforcing fair load distribution among peers. They cluster documents into semantic categories and cluster peers based on the categories of documents they offer. The authors emphasise the need to impose some logical system structure to achieve high performance in a peer-to-peer information retrieval network. Peers maintain a document index that maps document identifiers to categories, a cluster index that maps categories to cluster identifiers, and a peer index that maps cluster identifiers to peers. The terms in a query are first mapped to categories, then to clusters and finally to a random peer within the relevant clusters. This random peer tries to satisfy the query with its own local results, but if too few are available it forwards the query to neighbouring nodes in the same cluster. This repeats until there are sufficient results. Since selection is random, each peer in a cluster is equally likely to be picked, which achieves load balancing among peers within the same cluster. Peers are assigned to clusters based on the categories of documents they share. For load balancing among clusters the authors introduce the *fairness index* and a greedy algorithm to maximise this, called MaxFair, which also compensates for peers with different processing power, content distribution and storage capacities. The most powerful peer in each cluster is designated as leader that participates in the MaxFair algorithm. Categories may be dynamically reassigned to a different cluster to improve fairness based on the load of each cluster. They show that their approach is capable of maintaining fairness even when peers and document collections change.

3.2.2 *Non-Scientific Systems*

Various developed systems exist that do not have direct scientific roots. In this section we list several of the better known systems. Although we attempt to give some details about the underlying technology used, it is often a bit harder to classify these systems as operational details are sometimes missing or not well documented.

YaCy

`www.yacy.net`

In YaCy each peer crawls parts of the web and builds a local index. When connected to the larger YaCy network the local index entries are injected into a distributed global index with a high level of redundancy to ensure continuous availability. YaCy uses no centralised servers, but relies on a select set of machines that maintain seed lists for bootstrapping peers into the network. To protect user privacy it does not index deep web pages by default. However, these parameters can be changed. YaCy is an open project with transparent protocols and positions itself as a counter-movement against the increasing influence of, and dependence on, large-scale proprietary search engines. As of July 2011 it consists of about 600 peers, that indexed 1.4 billion documents and serve results for about 130,000 queries every day.

Seeks

`www.seeks-project.info`

This project aims to design and develop an open peer-to-peer network that provides a social search overlay: clustering users with similar queries so they can share both query results, similar to interest-based clustering, but also their experiences with those results: making it a social network. It aims to make real-time, decentralised, web search a reality. To protect the privacy of users the queries are hashed. Seeks performs no crawling, instead relying solely on users to push content into the network. Although it is initially populated with search results from major search engines. Seeks uses a distributed global index and is usable and under active development as of 2012.

Faroo

`www.faroo.com`

This is a proprietary peer-to-peer search engine that uses a distributed global index and aims to 'democratise search'. They perform distributed crawling and ranking. Faroo encrypts queries and results for privacy protection. They claim to be the largest peer-to-peer search engine with as many as 2 million peers.

3.3 KEY FOCUS AREAS

We believe there are several key areas that are important to focus on today to be able to create the peer-to-peer information retrieval systems of tomorrow. The following list is based on the existing research discussed in this chapter and our own insights:

- Combining the strengths of *global* and *local* indices and developing algorithms to easily shift appropriate content from one to the other based on term or query popularity. Many existing systems do not scale well because they are solely based on either flooding the network with queries or because they require some form of global knowledge (Zeinalipour-Yazti et al., 2004).

- No architecture exists that offers the best solution for all peer-to-peer information retrieval problems, different architectures apply to different situations.

- Although good scalability properties are inherent to the peer-to-peer paradigm, systems that wish to support web-scale search need to focus on effectively *distributing their load* over a high number of peers: hundreds of thousands to millions (Triantafillou et al., 2003). An important reason for this is that peers are heterogeneous in terms of capacity and connectivity and are not dedicated server machines: they have to perform many other tasks as well.

- Focusing on *search results* instead of documents. This means shifting attention to networks that provide access to external documents emphasising the *search* task: the core of peer-to-peer information retrieval.

- Investigating and improving the performance of *search result caches*. It is important to achieve a good balance between providing results that are sufficiently fresh and not taxing the network for updating those results. This also should depend on the, predicted or provided, mutation frequency of the resources pointed to.

- Improving handling of peer *heterogeneity* in web search. A few peers have a lot of documents, whereas many peers have much smaller collections. These smaller collections are often specialised, making them appropriate for more specific queries.

- Applying interest-based and/or content-based *clustering* as it simplifies both the construction of resource descriptions and query routing, resulting in reduced latency.

- Improving both *topology* and *query routing*, particularly for avoiding and routing around hotspots in networks. A good topology favours both effectiveness and efficiency, by making it possible for a query to reach a relevant target peer in few steps.

- Focusing on *precision over recall*. Achieving a hundred percent recall in peer-to-peer systems would involve searching in all indices and is far too costly (Klampanos et al., 2005). It is also unnecessary if the quality of the returned results is high enough. Although this requires better result fusion techniques. One should realise that web search users do not browse beyond the first search result page, but instead engage in query reformulation.

- Developing *real-time distributed relevance feedback mechanisms* (Klampanos and Jose, 2007). Ideally, search result quality continually improves based on user feedback as is common for centralised search engines. The emerging trend of coupling this to social networks could be further explored.

- Creating a number of large standardised *test collections* that apply to different types of peer-to-peer information retrieval networks. The work of Klampanos et al. (2005) provides a good start, but is still somewhat conservative with respect to scale.

- Focusing on *tangible benefits* of peer-to-peer networks rather than ethics or 'coolness', giving users a proper incentive to search using such networks over other solutions. This is particularly important for peer-to-peer web search engines.

- Realising that any web search service is a form of *adversarial information retrieval*: companies and people, suppliers of information, have an incentive to appear high up in rankings (Craswell and Hawking, 2009). Use this fact to improve the quality of service for the end users.

3.4 ECONOMIC SYSTEMS

There is a large body of scientific work that applies economics, game theory or mechanism design to peer-to-peer networks. These approaches try to give peers incentives to participate in the peer-to-peer network, or disincentives for unwanted behaviour, and thereby encourage them to actually also supply resources instead of only consuming them. They particularly aim to mitigate the problem of free-riding and discourage certain malicious behaviours, see Section 2.2 for definitions.

In this section we discuss a broad selection of related work in order to get a better understanding of what incentive mechanisms can be used for what purpose and the impact these have. They could be used to address some of the challenges in the key focus areas presented in the previous section. We do not restrict ourselves to information retrieval alone, as there is little information retrieval research that applies game theory directly. However, as we will see in Section 3.5, this may be a possible starting point, particularly when networks can not be assumed to be fully cooperative.

3.4.1 *Optimising Query Routing*

One problem in uncooperative networks is how to incentivise peers to participate in query routing. This is particularly important in networks that use either strict or aggregated local indices. Let us look at some reward systems that can be used for this situation.

Zhong et al. (2003) consider the problem of message forwarding in mobile ad-hoc networks. They focus on selfish nodes that need incentive to forward messages they receive to other nodes. A selfish node is defined as an economically rational node whose objective is to maximise its own welfare: the benefit of its actions minus the cost of its actions. They introduce a simple, cheat-proof, credit-based system for mobile ad-hoc networks with selfish nodes: Sprite. This system relies on a centralised credit clearance service that keeps track of the credit balance of nodes. Nodes are charged for sending messages and may obtain credit by either buying it using real money or preferably by forwarding messages from other peers. The authors model the transactions as a receipt-submission game and prove the correctness of their approach. Their prototype shows that their system adds very little processing overhead. Li et al. (2009) take

this work as inspiration and focus on query message routing in peer-to-peer networks that use strict local indices. When a peer issues a query it offers a reward for the results. Neighbouring peers are promised this premium as payment when relevant search results are returned via them. Peers may choose to which other peers they forward a query and do so in return for a part of the premium offered to them. Finally, when a peer is discovered via query routing that has relevant search results, these are passed back along the path to the peer that initiated the query. Along the way each peer is given the promised reward. This reward currency can be used to issue new queries by each peer and thus encourages participation in routing. The authors show their approach is better in utilising the peer-to-peer network than both query flooding and random walks. This set-up could also be useful in networks with aggregated local indices.

Sun and Garcia-Molina (2004) present a Selfish Link-based InCentive (SLIC) for peer-to-peer networks with strict local indices. Their focus is on servicing query messages in such networks. The flooding based search mechanism allows neighbouring peers to control each other's access to the rest of the network: a mutual access control relationship. SLIC exploits this relationship by allowing each peer to rate its neighbours and use the rating to control how many queries from each neighbour to process and forward. They point out two advantages of their approach: each peer is greedy and trying to maximise its utility and each peer needs to keep statistics only about its direct neighbours. SLIC operates in periods of, for example, one minute. During each period, a peer can use its capacity to service queries from neighbouring peers in the overlay network. They show a peer has several options to increase its reputation and thus its utility obtained from the network: by sharing more data and thus increasing its answering power, by increasing the number of edges (the connectivity) or by increasing the capacity used to service a neighbour's queries. The authors use the total and average number of hits to measure utility. They consider the initial rating that should be used for newly joining peers and show that using (functions of) the average rating of peers in the network is a good approach.

Yang et al. (2005) discuss non-cooperative behaviour in file-sharing peer-to-peer networks that use strict local indices. In their set-up, peers compete for the right to handle a query and deliver search results. Because of the nature of the network, peers may choose not to forward queries to potential competing peers. To solve this, the authors propose

an economic protocol that encourages peers to cooperate even when there is competition. A peer can buy the Right To Respond (RTR) for a query. Peers have incentive to buy this right, since it brings in business from a peer's perspective. When a query originates at a peer, an RTR is offered to each neighbour which consists of the reputation of the querying peer, a timestamp, the query and a price. The reputation is based on previous interactions of neighbouring peers with the requesting peer. Peers can choose the price at which they buy and (re)sell RTR's. To prevent useless offers, peers can indicate in a profile what categories of queries they can handle. These filters enable more intelligent routing of queries. The authors also allow peers to adapt the network to be close to other peers that have information that interests them, which reduces routing overhead. They make the assumption that there is some central entity that keeps track of the balance of each peer and assumes it is not easy for peers to change their identity. Their goal is to maximise the quality of the results and minimise the number of messages needed to obtain these results. They show that RTR has the same performance, in terms of quality and message overhead, in an uncooperative network as normal query flooding has in a cooperative network.

Karakaya et al. (2007) address the problem of free-riding in peer-to-peer file sharing networks. They assume a network with a topology like Gnutella: queries and search results are routed using the peer-to-peer overlay network, while file transfers are performed directly between peers. They show that by keeping track of the behaviour of neighbouring peers in the overlay network, a peer can determine whether a neighbour is free-riding. Tracking is done over time, so that possible counter actions are taken only when sufficient evidence has been gathered. They identify three types of free-riders: those that do not contribute any files but do route messages: non-contributors, those that share very little files and route messages: consumers, and those that do not contribute any files and also do not route any messages: droppers. They show that, depending on the type of free-riding, different counter actions can be appropriate. To counter, peers can either reduce the time to live of messages originating from the free-rider or drop originating message entirely. Whilst the latter seems very effective, it also affects genuine contributors, since the free-rider detection also yields false positives. An other problem is the fact that the actual downloads are performed between peers directly. This enables peers to lie and promise to service a download via query hit

messages, but to actually not deliver the 'goods' when the direct download is initiated, negatively affecting performance. To detect malicious behaviour the authors propose to broadcast notification messages about such events back over the path where the query hit messages originated from. Finally, they show their approach is resilient to a variety of other attacks, including collusion among free-riders.

3.4.2 *Levelling Supply and Demand*

While peers in a peer-to-peer network supply as well as consume resources, keeping this balance equal is a challenge. It may be beneficial for a peer to only take and not give anything back. There is a need for mechanisms that equalise the supply and demand, several of which have been explored.

Vishnumurthy et al. (2003) propose an economic framework for resource sharing called KARMA. In their system each peer has a certain amount of KARMA: a form of currency. The goal of KARMA is to discourage free-riding and achieve a balance between the resources consumed and supplied by peers. A distributed hash table is used for keeping track of which peer shares which file. To obtain a file, a peer pays some KARMA to the supplier peer that can provide the file. Groups of special nodes, termed *bank-sets*, keep track of the KARMA balance of each peer. The bank-set is distributed and replicated. Transactions involve two steps. Firstly, transferring the KARMA via the bank-sets. Secondly, transferring the actual file between the supplying and consuming peer. All transactions are zero-sum, encrypted and verifiable. The bank-set acts as mediator when conflicts arise. Their framework forms an interesting starting point for equalising contributions in peer-to-peer networks, although maintaining a monetary system seems cumbersome.

Ranganathan et al. (2003) claim the performance of incentive schemes in peer-to-peer networks is not well understood. They model two incentives schemes as a multi-person prisoner's dilemma: a token exchange (pricing) scheme where peers pay each other for files requiring a currency mechanism, and a peer approved (soft incentive) scheme based on reputations, requiring a reliable and secure mechanism for reputation tracking. They consider two classes of users: those that share all their files and those that share only a subset. With no incentive mechanism in place, the pay-offs are always higher for partial sharers. The peer approved system

encourages more peers to perform full sharing depending on the benefit function used. Token exchange leads to a higher number of files shared compared with the peer approved scheme when the files are distributed uniformly. However, when files are assumed to have a Zipf distribution, a few peers serving most of the files, the peer approved scheme converges to the same optimum as token exchange, albeit at a slower rate. This suggests that for peer-to-peer networks where the resources to be obtained are not uniformly distributed, like search results in a peer-to-peer web search engine, a reputation scheme may form a better starting point than a monetary scheme.

Ngan et al. (2003) investigate the usefulness of *auditing* as a mechanism for enforcing fair sharing behaviour in peer-to-peer networks. They believe that peers should not be placed in a position of permanent authority over other peers. If all nodes in the peer-to-peer system publish their digitally signed resource usage records and other nodes audit those records, each node has a natural incentive to publish accurate resource usage records. Peers in the network can still collude with others to lie collectively about their resource usage or they may even bribe other peers to condone false data. To prevent this, the audit process uses a challenge mechanism: If a peer claims to have a file, and thus claims to have less storage available, an other auditing peer asks for the hash codes of several random blocks of that file and compares this to those of the same blocks stored at other peers. Audits are performed anonymously by using one or more intermediate peers. Machines that hold a copy of a file perform normal audits periodically on their origin peers, whereas all peers perform random audits. Once a cheating peer has been discovered, its usage file is effectively a *signed confession* of its misbehaviour. The authors conclude that auditing provides incentives and increased resistance to collusion and bribery attacks.

3.4.3 *Establishing Trust Relationships*

One step further than directly levelling the supply and demand in a peer-to-peer system is making sure that peers in the system can be trusted. This commonly involves building ones reputation over time, the primary goal of this is increasing trust between peers. The secondary goals may vary from levelling contributions to fending of attacks, as we will see.

Kamvar et al. (2003) use reputation as a means to decrease the number of inauthentic files on a peer-to-peer file sharing network. Each peer in their system rates the other peers based on successful transactions. The challenge in a distributed environment is how to aggregate these local ratings, trust values, without centralised storage. The authors present EigenTrust that uses the notion of transitive trust: peers have a high opinion of peers that have provided them with authentic files and are likely to trust the opinions of those peers. EigenTrust, which uses basic linear algebra, eventually converges to the global trust value for each peer, meaning: the trust the system, as a whole, places in a particular peer. The authors run simulations with a large number of different threats and show that EigenTrust remains highly effective even when up to 70 percent of the peers in a network form a malicious collective. In that specific case, only 10 percent of downloads are inauthentic, versus 96 percent when EigenTrust is not used.

Grothoff (2003) describes the GNUnet system: a peer-to-peer network for anonymous distributed file sharing. He employs an economic trust model to perform resource allocation and to discourage free-riding. The author considers trust to be local and non-transitive: not communicated between peers, in contrast with Kamvar et al. (2003). There are no special nodes in his system. Each peer keeps track of transactions performed with other peers in the past and learn which ones behave well. As in any peer-to-peer network, a node can never be guaranteed that a service that it has provided in the past will pay off in the future. If a peer has a resource shortage it may choose to drop requests from other peers that have earned low trust: consumers compete for the resources at suppliers with the trust they earned. The author states that simple supply-demand schemes are inadequate for peer-to-peer networks since supply usually exceeds demand: most computing resources are available in excess most of the time and massive resource consumption only occurs during brief peak periods. Hence, they conclude that freeloading is harmless as long as excess resources are available. A number of attack scenarios are considered, but ultimately it appears that the damage a peer can inflict is limited by its own bandwidth. The authors briefly consider the problem of fake content and propose user feedback as a solution. However, in an anonymous network, the feedback itself can be either a valuable contribution or a malicious deception. Finally, the author states it is difficult to evaluate if the algorithms they present optimise for the right goal

which is to allocate resources for each peer proportional to the amount of resources provided. The problem with this definition is that it does not capture protocol overhead and variations in resource value over time. Hence evaluation remains an open issue.

Andrade et al. (2004a) attempt to reduce free-riding in a peer-to-peer grid system using autonomous, local, reputations. They state economies based on currencies are too complex to implement, but do recognise that in the absence of incentives for donation, a large proportion of peers only consume resources: free-riders. In their CPU-sharing grid, termed OurGrid, the peers are assumed to be eager consumers: a peer gains positive benefit from whatever resources it obtains. A peer calculates the reputation of an other peer locally based on the favours it has received from and given to that other peer, in terms of donated processing power. This build up of autonomous local reputations uses no information on interactions that did not directly involve the peer assessing the reputation, which reduces the number of ways in which malicious peers can distort the reputation system. Techniques such as lying about the behaviour of third parties can not be applied. The authors run experiments with a small network of hundred peers. Each peer can be either consuming, not consuming and donating their resources: collaborate, or not consuming and staying idle: free ride. They show their approach reduces the number of transactions with free-riders over time, depending on the number of free-riders in the network and the total number of consuming peers. However, this convergence takes some time which might not work very well in dynamic resource-sharing networks with many joining and departing peers. Additionally, they show that if free-riders can change their identity, their method does not converge if the reputation system can be both positive and negative. However, when using only positive reputations the network is far less sensitive to identity changing free-riders, regardless of the total number of consumers in the network. In follow-up work (Andrade et al., 2004b) the authors drop the eagerness assumption: consuming peers have a limit on the amount of resources they can use with positive utility. They do assume that the value of giving and receiving a donation does not vary much between peers, which seems realistic for grid computing. Simulation reveals that even when there is a famine of donations, their approach encourages cooperative behaviour among peers. The elegance of their approach is that it does not require central coordination or cryptography.

3.4.4 *Using Behavioural Histories*

An alternative way to assess the contribution of other peers in the network is looking at their behavioural histories: a more direct method than using trust or reputations. We look at two such approaches.

Feldman et al. (2004) investigate incentives for peers to cooperate in peer-to-peer networks. They attempt to prevent the so called 'tragedy of the commons' : the situation where the result of each peer attempting to maximise its own utility, lowers the overall utility of the system. They claim using conventional strategies, like tit-for-tat, does not work well in a peer-to-peer context in contrast to Cohen (2003). Their starting point is the generalised prisoner's dilemma. A peer can either defect or cooperate and peers can reciprocate towards each other. To decide whether to cooperate with a peer in the future, peers can keep a private history where they record the behaviour of previously contacted peers. However, since peer-to-peer networks are large, the likelihood of multiple transactions between the same peers is low. Hence, the authors propose a global shared history, stored in a distributed hash table, where the behaviours of all peers are tracked: how many times a peer defected or cooperated and the transactions that have taken place. The shared history introduces new peer behaviours that can have a negative effect on cooperation in the network. Peers can collude to report false statistics, for example: being overly positive by indicating that a certain peer always cooperates, even if it did not. The authors use the Maxflow algorithm to counter this when not too many peers are lying about the shared history statistics. A complete discussion of Maxflow is beyond the scope of this thesis. One final issue is that of zero-cost identities. A peer can easily disconnect from the network and reconnect to whitewash its own history. To handle this, global statistics are kept on the behaviour of these 'strangers'. Peers decide what their behaviour towards strangers is based on these statistics. The authors show that this stranger-adaptive approach is effective both when using private and shared histories. Finally, the authors show that traitors, peers with good reputation that defect, can be countered by looking only at the short-term history.

Friedman and Resnick (2001) treat the problem of cheap pseudonyms prevalent on the Internet. Peer-to-peer networks are a specific instance of this problem, since peers can easily depart and rejoin the peer-to-peer network and, by doing so, clear their reputation. The authors consider

several scenarios based on a repeated prisoner's dilemma that rewards mutual cooperation, keeps mutual defection costs neutral and gives an added bonus for defecting if the opponent chooses to cooperate. Players playing their first game are termed newcomers and players that played in previous rounds are referred to as veterans. Players are randomly paired each round. If players can not change their identity, then a localised punishment strategy yields a sustainable equilibrium: cooperate with those who cooperated in the previous round and defect with those that defected. If players can change their identity, then the public grim trigger, every player defects if there has been a defection in the previous round, is an equilibrium. However, this is highly inefficient, as a simple network error can paralyse the entire game. An alternative is to punish newcomers by having the veterans always defect when they are paired with a newcomer. The authors show that this entry-fee for newcomers yields an equilibrium, and that there is no alternative strategy that yields a higher pay-off. There appears to be no other way than to punish newcomers. They consider letting newcomers pay a fixed amount to all veterans in the network, but conclude that this can cause players to stay around for longer than they can contribute anything useful and discourages poor peers from participating. Finally, they introduce an alternative way to discourage identity changing: issuing regular and once-in-a-lifetime identities. This approach gives participants an option to commit to not changing identities. In equilibrium no one would use the regular identifiers.

3.4.5 *Explaining Free Riding*

So far we have seen that many of the approaches discussed have the purpose of reducing or eliminating free-riding. However, is it really necessary to do this? Why do networks with free-riding peers remain operational? We look at two studies that examine this.

Krishnan et al. (2002) attempt to explain why free-riding can be tolerated in peer-to-peer file sharing networks. Ideally, the fact that once users download files they, by default, also share it immediately, leads to the situation where the provision of content on the network scales to match the level of demand for the content. The authors state that files on peer-to-peer networks have the economic properties of public and club goods: non-rivalry in demand and non-excludability in supply. Tradi-

tional economic theory predicts that free-riders cause inefficient private provision of public goods and calls for central intervention to remedy this problem. However, this does not explain the high levels of sharing seen in peer-to-peer networks. While sharing implies a cost by reducing the user's private bandwidth, it also reduces traffic other peers place on other nodes in the network which increases the user's utility. The authors model the problem of free-riders using game theory and conclude that peer-to-peer networks can tolerate a certain amount of free-riding. Above this threshold the network would collapse, the risk of which encourages newly joining nodes to share instead of free-ride, since no one benefits from a defunct peer-to-peer network. Free-riding appears to be part of a sustainable equilibrium.

Jian and MacKie-Mason (2008) investigate the incentives for file sharing in peer-to-peer networks. They first consider the so-called offload effect: sharing redistributes network traffic to the advantage of the sharing peer. However, in large networks, where highly sought after files are highly replicated already, the benefit of offloading by sharing appears to be quite minimal. The authors conclude that the offload effect alone is not sufficient to motivate the amount of sharing seen on successful peer-to-peer networks. They turn to indirect incentive for contributing to the public good, also known as *generalised reciprocity*: 'I will do this for you without expecting anything back from you, in the confident expectation that someone else will do something for me down the road'. They show that, even in the presence of free-riders, stable equilibria emerge where only some peers share content. The reason for this appears to be that peers care a lot about fulfilling their own demands. Since the cost of sharing is relatively low, free-riding can be tolerated to a certain extent.

3.4.6 *Conclusion*

We have seen various ways in which economics can be applied to peer-to-peer networks. We briefly summarise the main findings:

- Rewarding peers is an effective way to incentivise them to participate in query routing, and is necessary in uncooperative environments that use local indices. This can be implemented using credits, mutual ratings, response rights or behavioural tracking (Zhong et al., 2003; Sun and Garcia-Molina, 2004; Yang et al., 2005; Karakaya et al., 2007).

- Reputation systems can be used to level supply and demand in peer-to-peer systems. This can be based done using some type of currency, although monetary schemes seem somewhat impractical. An alternative is rating or auditing peers. A reputation system should be robust to malicious collectives of peers that conspire to subvert the system (Vishnumurthy et al., 2003; Buragohain et al., 2003; Ranganathan et al., 2003; Ngan et al., 2003; Kamvar et al., 2003; Karakaya et al., 2007).

- Systems that build trust by tracking contributions over a long time period remain effective when under attack. Trust can be used as a currency for resource consumption, to reduce free-riding and to stimulate cooperation among peers (Kamvar et al., 2003; Grothoff, 2003; Andrade et al., 2004a,b; Suryanarayana and Taylor, 2004).

- Behavioural histories give insight into how the tragedy of the commons can be prevented. They can also be used to determine entry fees for newcomers to the network, as this discourages them from whitewashing their reputation by disconnecting and reconnecting to the network (Friedman and Resnick, 2001; Kamvar et al., 2003; Feldman et al., 2004).

- While free-riding is unwanted behaviour, and many systems focus on discouraging it, it seems that free-riding is also part of a sustainable equilibrium: a certain amount of free-riding can be tolerated (Krishnan et al., 2002; Jian and MacKie-Mason, 2008).

- Simplified games can be used to model and reason about resource exchange in peer-to-peer networks. However, very basic supply-demand schemes are inadequate, since supply usually exceeds demand (Friedman and Resnick, 2001; Grothoff, 2003; Wang and Li, 2003; Ranganathan et al., 2003; Zhong et al., 2003; Feldman et al., 2004).

3.5 OUR NETWORK ARCHITECTURE

Armed with an understanding of peer-to-peer information retrieval, background knowledge of existing systems and challenges as well as economic systems and incentive mechanisms, we are ready to explain our architecture in this section. We use this as the basis for our experiments in Chapter 4, 5 and 6.

The most popular peer-to-peer file sharing technology to date is BitTorrent (Cohen, 2003). Torrent traffic is a mainstay of residential broadband users (Maier et al., 2009) and accounts for a substantial amount of traffic on the Internet (Pouwelse et al., 2005). We use it as inspiration and starting point for our own architecture and experiments. While we have briefly touched on BitTorrent in Section 2.3, we describe it more elaborately here and draw an analogy with peer-to-peer information retrieval. We start with a brief overview in Section 3.5.1, followed by a translation to information retrieval in Section 3.5.2. We close with a selection of literature specifically about BitTorrent in Section 3.5.3.

3.5.1 *BitTorrent Overview*

BitTorrent (Cohen, 2003) is based on the observation that the upload capacity of peers that are downloading largely goes unused. To solve this, BitTorrent cuts files into small blocks that are downloadable not only from the original peer that shared the file, but also from all still downloading peers that have obtained a subset of those blocks. This way, peers that are downloading can saturate their download link by transferring blocks from many different peers, not only the peer that shared the original file. Although BitTorrent was originally released in 2001, it did not really take off until 2003 when it was used for spreading Linux distributions and other content. Crucial to its adoption was the emergence of websites that allowed users to search for files to download.

Instead of downloading the actual file from a website, users obtain a small meta-data file: a torrent. Besides a filename and size, this file contains two important pieces of information needed for locating and transferring. Firstly, hash values for each block of the file to be downloaded. Secondly, the address of a tracker that keeps track of the peers that are also sharing (parts of) the file. Peers that have complete copies of the file are termed *seeders*, whereas peers that have not yet downloaded all blocks

are termed *leechers*. When a peer wants to initiate transfer of a particular file it contacts the tracker to obtain a list of seeders and leechers, this is the locating task termed *scraping* in BitTorrent. The strength of BitTorrent is handling situations where only a few machines have the complete file and many of them are still downloading it. The tracker has the crucial tasks of helping peers, that are interested in obtaining the same file, find each other. By default trackers return a limited random set of peers that share the same file. All peers that share (parts of) the same file, and upload to and download from each other, are collectively called a *swarm*.

BitTorrent uses no central resource allocation for the transfer task. Instead each peer tries to maximise its own download rate. Leechers download from whichever peer has the current block they want. However, those peers may refuse to upload, this is termed *choking*. A peer uploads in parallel only to a limited number of peers, termed the *active set*, providing them all with an equal share of its upload bandwidth. The rest of the peers are choked. The size of the active set typically depends on the total available upload capacity. Peers prefer to unchoke peers that give them a high download rate (for other blocks of the file that they are still downloading). That rate is the average measured over a fixed time window: typically the last twenty seconds. Each uploading peer redecides every ten seconds which peers it unchokes. To find better connections a peer also optimistically unchokes a random peer every thirty seconds, regardless of the download rate it gets from that peer. If a downloading peer does not retrieve a block it has requested from a peer it is connected to for over a minute, it assumes that peer has 'snubbed' him and will not upload to that peer again, except as an optimistic unchoke. After a peer has downloaded the file completely, and is thus a seed, it simply prefers to upload to peers to which it has the best upload rates.

The ideas behind BitTorrent's resource allocation closely resemble the *tit-for-tat* strategy in game theory: if you give me a good download rate, I will upload to you as reward. But, if you do not upload to me for some time, I will stop uploading to you as well. Although after a while I might contact you again and see if you reciprocate if I upload something to you. The initial strategy is to cooperate and to defect only if the other party defects as well. A party's previous behaviour is held against him only once: peers do not keep a history of past behaviour of other peers.

Recent years have seen a move away from centralised components in BitTorrent. A limited form of this is bypassing the tracker and obtaining

knowledge of additional peers in the swarm by exchanging information with other peers. A stronger change has been doing away with centralised trackers altogether: trackers are no longer dedicated server-like machines, instead a distributed global index is used to provide the same functionality. BitTorrent uses a derivative of Kademlia as distributed hash table[3] (Maymounkov and Mazières, 2002). Effectively, all peers collectively provide the locating service, removing a single point of failure. Furthermore, the global index is not only used for locating suitable peers, but also for the step that precedes it: downloading the torrent meta-data file. Instead of a torrent file, websites only provide a so-called magnet link, which is a hash over a part of the content of the torrent file. Peers can download the full torrent file by performing a look-up of this hash value in a distributed global index to see which peers provide it[4]. Hence, the locating and transferring steps are first carried out for the torrent file, and later followed by the same steps for transferring blocks of the actual file the torrent file describes.

3.5.2 *BitTorrent and Information Retrieval*

BitTorrent's success makes it an attractive model for other peer-to-peer applications, but how do we apply the principles set forth by BitTorrent to information retrieval? We have already seen the differences between file sharing and information retrieval at an abstract level in Section 2.6.1. Let us take a closer look at how we can translate some of BitTorrent's ideas to peer-to-peer information retrieval. The following paragraphs describe the architecture that we will use in the remainder of the thesis.

We take the point of view that peers are capable of providing relevant *search results* instead of the conventional view that peers hold static document collections. This distinction makes it easier to abstract away from who actually holds the document, as we do not focus on actually obtaining those: the transfer task is considered to be outside of the scope of the peer-to-peer network. A search result, which we also refer to as *snippet*, consists of at least a location (URL), usually a title and a query-dependent summary. This is treated in more detail in Chapter 4.

Starting with the tracker, we have to recognise that the task of the tracker in a file sharing network is merely *locating* appropriate peers,

3 `http://www.bittorrent.org/beps/bep_0005.html` (Retrieved June 25th 2012)
4 `http://www.bittorrent.org/beps/bep_0009.html` (Retrieved June 25th 2012)

whereas the task of a tracker in information retrieval is *searching* for appropriate peers. We can view this as the tracker giving *advice* to peers for where they can obtain relevant search results for their queries. Essentially this makes the network form a two-step index: one step to map a query to appropriate peers and a second step to send that query to those peers and obtain the actual search results. This has something in common with federated information retrieval's mediator party, except that the tracker is not a façade and does not perform actual search result retrieval or merging: it is more lightweight (see Section 2.6.2).

For estimating which peers have relevant content for a given query, the tracker needs representations of each content bearing peer: each *supplier* in the network. Hence, when peers bootstrap into the network they either need to cooperate and provide a resource description, explored in Chapter 5, or the tracker needs to sample their content, examined in Chapter 4. We assume the tracker stores the resource descriptions in a centralised global index. As we have seen in Section 3.5.1, this starting assumption also makes it easier to decentralise the tracker by switching from a central global index to a distributed global index later on, or perhaps even to aggregated local indices. Keeping resource descriptions up to date is an open challenge we briefly discuss in Chapter 6.

We can define a swarm in our system as the collection of all peers that have search results for a particular query. A search result set could conceptually be viewed as a (very small) file, and blocks could be seen as either individual search results or subsets of the full search result set. The active set can be viewed as the maximum number of queries a peer is willing (or can) process per unit of time. But, what is a seeder and what is a leecher? In an information retrieval network we can distinguish between suppliers that provide search results and consumers that pose queries. Thus suppliers may seem like seeds and consumers may seem like leechers. However, we run into a fundamental difference here: in file sharing the leechers are all interested in obtaining the same file *at the same time*. This works because the time it takes to obtain a file is typically long. Contrast this with search results that can be transferred in the blink of an eye. Hence, in an information retrieval network, peers will rarely have a symmetric interest at exactly the same point in time. How do we deal with this? In Chapters 6 we explore search results caching as solution for this asymmetry. Consuming peers are expected to cache the search results for the queries they posed, so that others may obtain those results

directly from them at a later point in time. This reduces the load on the supplier peers, just as leechers reduce the request load on the seeders in BitTorrent: search result caching also exploits the storage and upload capacity of other peers in the network. There is one small problem with this analogy. As soon as the consumer peers obtain search results for a query they directly become suppliers themselves. Hence, true leechers in the BitTorrent sense do not really exist in our peer-to-peer information retrieval network. Instead we make the conceptual distinction between origin suppliers: those that generated the initial search results for a query, and other suppliers that hold cached results. This leads us to solving a problem that BitTorrent itself does not actually address: how to keep the seeders around?

Whenever a file transfer completes in BitTorrent, and the peer becomes a seeder for the file, there is a huge incentive to disconnect from the swarm. Ideally nodes would stay in the swarm for as long as needed to give back as much bandwidth as was used for the download. Equivalently, we would like consumer peers that have obtained a search result set for a query to keep it around so that others may use it. Thus, this same problem exists in our architecture and this is where we can extend some of the mechanics BitTorrent uses during file transfers. BitTorrent's tit-for-tat can be viewed as a very simple reputation system with a short-term memory. One way to implement this idea in a peer-to-peer information retrieval system would be using reputations as incentive for keeping search result caches. A peer's reputation is lowered in response to cache misses. Other peers are less willing to share their cached items with peers with a low reputation, and hence can choke them by refusing to serve cached results for queries. This forces the low reputation peer to obtain results from other peers which increases latency. Of course: peers can also optimistically unchoke a peer so it can gain reputation. Furthermore, peers with low reputation are less likely to be selected as origin suppliers by the tracker. This makes it more difficult to share unique content and makes sense when you consider that low reputation peers likely serve spam web pages or are link farms.

To summarise: our architecture uses a tracker that contains resource descriptions of all supplier peers in the network. Peers in the network are expected and incentivised to cache search results they obtain for queries in order to evenly balance the load. The focus is thus explicitly on search results and not on the documents they are derived from. We assume

the tracker is implemented as a global index, which could be either a centralised or distributed global index, as described in Section 2.4. Furthermore, the tracker maps queries to peers, the peers themselves map the queries to search results. Hence, the architecture is a global two-step index as shown in Figure 3.1.

3.5.3 *Studies of BitTorrent*

The incentive of giving upload bandwidth resulting in download reciprocation is said to build robustness in BitTorrent (Cohen, 2003). However, this claim is somewhat disputed. In this section we give a brief overview of existing studies of BitTorrent to show strengths and points of improvement. We focus particularly on its incentive mechanics. At the end of the section we briefly summarise what the observations in these studies mean for the information retrieval analogy.

Pouwelse et al. (2005) perform a detailed study of the performance of BitTorrent in combination with SuprNova. BitTorrent is a system that has stood the test of intensive daily use by a very large user community. The authors stress that it is *only* a download protocol and that it needs to be combined with global components for finding files. Although the lack of global components increases *availability*, their presence increases *data integrity*. The authors identify three global components: a website offering a searchable archive of torrent files, mirrors of that archive and central trackers that keep track of who is actively downloading a particular file. The torrents that appear at the website are user moderated to reduce the number of fake and corrupted files. This moderation appears to be very effective. A central tracker provides peers with a list of some of the other peers that share the same file, with whom the user can establish direct connections to barter for chunks of the file. Unfortunately, the trackers, as well as the other global components, are a frequent target of denial-of-service attacks and they are costly to operate due to gigabytes of daily bandwidth consumption. The authors conclude there is an obvious need to decentralise these global components. While peers with high upload rates will, with high probability, also be able to download with high speeds, the resources of peers that share a file initially, the seeders, are heavily taxed. The author see this as a flaw in BitTorrent's design and state that peers should be rewarded for seeding instead of being punished. Additionally, peers should also be given incentives to lengthen their up-

time, as most of the peers remain on-line only for a very short time. Their study shows that the arrival and departure process of downloaders does not follow a Poisson model as suggested by Qiu and Srikant (2004), rather it is dominated by *flash crowds*: large groups of peers that form temporarily to download a specific file and then disappear once done. BitTorrent can handle large flash crowds.

Bharambe et al. (2005) investigate the performance of BitTorrent using a simplified implementation of the protocol in a discrete-event simulator. They make several interesting observations about the protocol. Firstly, preserving seed bandwidth is crucial when there are only a few seeds. Initially serving only unique blocks, also known as *superseeding*, is an excellent way to do this. Also, prioritising the rarest blocks for download is important to prevent the situation where almost all blocks have been downloaded except one rare block, as this increases download times and, at worst, makes it impossible to complete a download.

What about reciprocation? The authors find that tit-for-tat fails to prevent unfairness across nodes in terms of volume of content served. This is primarily due to heterogeneity in the bandwidth capacity of participating machines: high bandwidth peers that connect to low bandwidth peers and vice verso. The authors propose applying tit-for-tat at the level of individual blocks, instead of average transfer rates. However, changing the protocol is infeasible as so many peers are already using the unfairness in the existing protocol and thus have no incentive to switch to a fairer protocol at all.

In general it seems better to design or adjust a system such that the most selfish behaviours lead to an optimal outcome, rather than actually relying on altruism. Piatek et al. (2007) share this view and ask the question: can a strategic peer 'game' BitTorrent to significantly improve its download performance at the same level of upload contribution? The authors show that the performance of BitTorrent is not so much due to its tit-for-tat bandwidth sharing, but can be attributed to the fact that many people use default, suboptimal, settings and to BitTorrent clients that do not strategise. Whilst Qiu and Srikant (2004) mathematically showed that an equilibrium can emerge, Piatek et al. (2007) conclude that it rarely does in practice. BitTorrent, they claim, relies on significant upload altruism provided by peers with high bandwidth capacity. Such high capacity peers are often forced to pair with low capacity peers until they happen to run in to an equal peer by optimistic unchokes.

The authors verify their claims experimentally by using a modified strategising BitTorrent client with the telling name *BitTyrant*. Three strategies are used in BitTyrant to improve performance. Firstly, finding peers with large reciprocation bandwidth. Secondly, expanding the active set dynamically until the marginal benefit of one more unchoked peer is outweighed by the cost of reduced reciprocation of other peers. Thirdly, instead of dividing the upload bandwidth equally among the peers in the active set, the client lowers its upload contribution to a peer as long as the peer continues to reciprocate. This allows for a larger active set. To summarise: BitTyrant differs from the reference BitTorrent implementation by dynamically shaping its active set and varying the upload rate per connection. The rationale is that the best peers reciprocate most for the least number of bytes contributed back to them.

BitTyrant was evaluated both in the real-world and in a controlled wide-area test bed environment known as PlanetLab[5]. Evaluation shows that BitTyrant offers a median 70 percent performance gain for a 1 Mb/s client on live Internet swarms. Of course, in such swarms it is the only BitTyrant client among peers using non-strategising clients. However, even in swarms that consist of only BitTyrant clients, it still offers better performance over the reference implementation, since the strategies make more optimal usage of available bandwidth.

Recent implementations of BitTorrent use a distributed global index instead of a centralised tracker. Historically two distributed hash tables have come into use: the Main-Line DHT (MDHT) and the Azureus DHT (ADHT), both of these are Kademlia derivatives that make different choices with respect to functionality and parameters. Crosby and Wallach (2007) examine both implementations and find that they fail to implement important aspects of Kademlia, like extra routing tables and data migration. This leads to erroneous behaviour like lookups that never return, which happens for 20 percent of the lookups in the MDHT, rejections of key store attempts and long lookup delays, in the order of minutes, caused by time-outs of dead peers. Furthermore, they point out that distributed hash tables raise security concerns that are difficult to address. Falkner et al. (2007) find that the maintenance overhead of the ADHT is high and makes up 81 percent of the network traffic. The length of lookups are the result of conservative time-outs. They suggest the ADHT

5 `http://www.planet-lab.org` (Retrieved June 25th 2012)

can be adapted to reduce lookup latency and conclude that median response times of 281 milliseconds are possible. They also propose aggressively refreshing routing tables when there is not much network demand or churn.

Tribler is a social peer-to-peer network optimised for streaming media introduced by Pouwelse et al. (2008a). Conventional peer-to-peer file sharing systems focus exclusively on technical issues and are unable to leverage the power of social phenomena. The authors believe that peers in a peer-to-peer network had best be viewed as social partners rather than solitary rational agents. They identify five key research challenges for peer-to-peer file sharing: decentralisation, availability, integrity, providing incentives and network transparency. The prime social phenomenon that is exploited in Tribler is that of 'kinship fosters cooperation'. They have implemented the ability to distinguish friend from foe and newcomer. Virtually all peer-to-peer systems lack persistent memory about previous activity in the network about, for example, social relations, altruism levels, peer uptimes and taste similarities. Tribler uses a number of megacaches: the friends list with information on social networks, the peer cache with information on the peers in the network, the metadata cache with file metadata and the preference cache with the preferences of other peers. The megacaches are kept up-to-date by using Bloom filters to reduce bandwidth overhead. They manage to reduce the metadata cache to a size of several hundreds of kilobytes which allows it to be cheaply replicated among all peers. This reduces the problem of content discovery from network-based keyword searching to a local search. To locate content, people with similar tastes, taste buddies, are connected with each other. To more effectively connect people, geolocation is used. Tribler is bootstrapped by first connecting to known super peers and thereafter participating in the global Tribler overlay swarm that is solely used for content and peer discovery. A fundamental limitation of file sharing systems is that of the session boundary: once the session ends, all context information is lost. To solve this, the authors introduce permanent and unique peer identifiers using public-key cryptography. For communication among peers the epidemic BuddyCast algorithm is used (Pouwelse et al., 2008b). For the actual downloads they use the 2Fast protocol that divides peers into collectors, that want to obtain a resource, and helpers, that are willing to assist in this for free. The authors conclude that the idea of exploiting the naturally occurring social connections between humans

behind the computers in large-scale peer-to-peer networks is starting to become a major research topic. They also mention reputation systems as important future work.

With respect to improvements to BitTorrent's mechanics we can conclude that seeds should be rewarded and peers should be incentivised to remain part of the network for a longer amount of time. In an information retrieval set-up this translates to offloading peers that provide original search results and incentivising peers that have caches to keep them. Whilst saturating the upload links of peers is not a goal of peer-to-peer information retrieval, the more general goal of using those links to serve cached results is similar. To superseed would be to make sure that results for rare queries are replicated to a sufficient degree, so that they will not disappear if the origin supplier goes off-line. This makes sense for information retrieval to an extent, results for rare queries should remain available. On the other hand, filling caches with results for queries that are rarely asked seems wasteful. A possible solution for this would be to anticipate the demand for such queries. However, this deserves a study of its own, making a good trade-off is non-trivial and not subject of this thesis. A starting point from a centralised perspective is given by Fagni et al. (2006) and Lempel and Moran (2003).

4

REPRESENTING UNCOOPERATIVE PEERS

'The biscuit degenerates
outside the culture!'

(randomly generated)

A peer-to-peer information retrieval network may initially contain few documents, and can thus answer few queries: there is a bootstrap problem. In this chapter we examine how to represent uncooperative peers and external search servers, so that queries for which no cooperative peers qualify can be directed to them.

A fundamental problem in a peer-to-peer web information retrieval system is how to seed the network with sufficient content. A small network with few peers is unlikely to attract a large following. Hence, we need to overcome this content bootstrap problem: the network must be able to provide useful results for a large range of queries from the start. There are two ways in which this can be achieved. Firstly, there can be *uncooperative* peers within the network: they may provide only a partial or legacy interface for searching their content. This contrasts with *cooperative* peers that can provide any information about their document collection that is required. Secondly, there are search servers *external* to the peer-to-peer network, that may also provide only a basic search interface. In this chapter we consider these cases to be equivalent and interchangeably use the terms uncooperative peer and search server. The work discussed in this chapter applies to peer-to-peer information retrieval that uses a

Parts of this chapter appeared previously in Tigelaar and Hiemstra (2009, 2010a,b).

tracker, as well as to federated information retrieval that uses a mediator. The way in which the representations obtained are used for retrieving results differs between these two scenarios as described in Section 2.6.2.

A search server minimally provides an interface that can accept queries and return search results. The tracker can fall back to redirecting a query to these search servers in case there are no or few cooperative peers in the network capable of providing relevant search results for that particular query. However, there is one problem with this approach: since search servers are external to the peer-to-peer network, and like uncooperative peers do not offer more than a minimalistic search interface, they can not provide the tracker with a resource description. Hence, we have to turn to other techniques to represent the content of these external servers, so that relevant queries can be directed to them. In this chapter we look at various ways of improving an established technique for obtaining such representations: query-based sampling.

Query-based sampling can be used to discover the content available at a search server (Callan and Connell, 2001). This method makes only a minimal assumption: the server should be able to receive and process queries and send back a list of search results. Each search result, also referred to as *snippet*, consists of a location, title and a brief summary. No other functionality is necessary beyond what end users of a search engine would use directly via its interface. Conventionally, random queries, consisting of a single term, are sent to this search engine and the documents the returned search results point to are fully downloaded from their source locations by the tracker, or mediator in federated information retrieval, to create a resource description of the server. This resource description is a language model: terms with occurrence counts. For example: $[(apple, 16), (pear, 23)]$. The vocabulary is defined as the terms without occurrence counts: $[apple, pear]$. The first query sent is conventionally a frequently occurring word in the (English) language obtained from an external resource, for example: a dictionary. The purpose of this first query is to retrieve some initial search results. Subsequent queries are terms drawn *randomly* from the vocabulary of the language model constructed so far. Thus, conventionally only the query sent in the first iteration is based on an external resource. The process of sending queries, downloading documents and updating the language model, iterates until a stopping criterion is reached. For example, when three hundred documents have been downloaded or after hundred iterations.

In this chapter we will look at two research questions with respect to query-based sampling:

1. Can query-based sampling using full documents be improved by using an alternative term selection strategy?
2. How does query-based sampling using *only snippets* compare to downloading full documents in terms of the learned language model?

4.1 DATA SETS

All experiments in this chapter rely on a subset of the following datasets:

OANC: The Open American National Corpus 1.1: A heterogeneous collection. We use it exclusively for selecting bootstrap terms (Ide and Suderman, 2010).

TREC123: A heterogeneous collection consisting of TREC Volumes 1–3. Contains: short newspaper and magazine articles, scientific abstracts and government documents (Harman, 1994). Used in previous experiments by Callan and Connell (2001).

WT2G: Web Track 2G: A small subset of the Very Large Corpus web crawl conducted in 1997 (Hawking et al., 2000).

WIKIL: The large Memory Alpha Wiki.
`http://memory-alpha.org`

WIKIM: The medium sized Fallout Wiki.
`http://fallout.wikia.com`

The first experiment in Section 4.3, about alternative term selection strategies, relies on the first three sets only. The second experiment, in Section 4.4, uses all of them.

The OANC is used as external resource to select a bootstrap term on the first query-based sampling iteration: we pick a random term out of the top 25 most frequent terms, excluding stop words. TREC123 is used for comparison with Callan and Connell (2001). WT2G is a representative subset of the web. It has some deficiencies, such as missing inter-server links (Bailey et al., 2003). However, since we use only the page data, this is not a major problem for this experiment.

Table 4.1: Properties of the Data Sets Used

Name	Raw	Index	#Docs	#Terms	#Unique
OANC	97M	117M	8,824	14,567,719	176,691
TREC123	2.6G	3.5G	1,078,166	432,134,562	969,061
WT2G	1.6G	2.1G	247,413	247,833,426	1,545,707
WIKIL	163M	84M	30,006	9,507,759	108,712
WIKIM	58M	25M	6,821	3,003,418	56,330

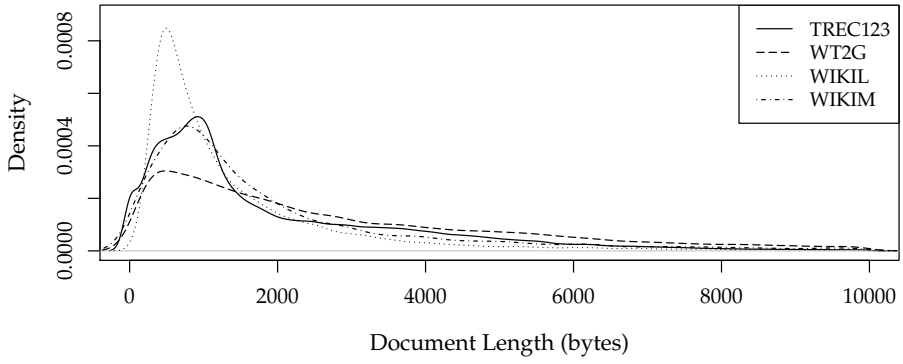


Figure 4.1: Kernel density plot of document lengths up to 10 kilobytes for each collection.

We have included two additional Wiki collections for use in the snippet experiment: WIKIL and WIKIM, to examine the effect on recent, self-contained search servers. All Wiki collections were downloaded from Wikia, on October 5th 2009. Wikis contain many pages in addition to normal content pages. However, we index *only* the content pages.

Table 4.1 shows some properties of the data sets. We have also included Figure 4.1 that shows a kernel density plot of the size distributions of the collections (document mark-up removed) (Venables and Smith, 2012). WT2G has a more gradual distribution of document lengths, whereas TREC123 shows a sharper decline near two kilobytes. Both collections consist primarily of many small documents. This is also true for the Wiki collections, especially WIKIL has many very small documents.

4.2 METRICS

Evaluation is done by comparing the complete remote language model of the search server with the subset local language model at the tracker each iteration. We discard stop words and compare terms unstemmed. Various metrics exist to conduct this comparison. For comparability with earlier work we use two metrics, the CTF ratio and KLD, and introduce one new metric in this context: the Jensen-Shannon Divergence, which we believe is a better choice than the others for reasons outlined later.

We first discuss the Collection Term Frequency (CTF) ratio. This metric expresses the coverage of the terms of the locally learned language model as a ratio of the terms of the actual remote model. It is defined as follows (Callan et al., 1999):

$$\mathrm{CTF}_{ratio}\left(\mathcal{T}, \hat{\mathcal{T}}\right) = \frac{1}{\alpha} \cdot \sum_{t \in \hat{\mathcal{T}}} \mathrm{CTF}\left(t, \mathcal{T}\right), \tag{4.1}$$

where $\mathcal{T}$ is the actual model and $\hat{\mathcal{T}}$ the learned model. The CTF function returns the number of times a term t occurs in the given model. The symbol α represents the sum of the CTF of all terms in the actual model $\mathcal{T}$, which is simply the number of tokens in $\mathcal{T}$. The higher the CTF ratio, the more of the terms have been found.

The Kullback-Leibler Divergence (KLD) gives an indication of the extent to which two probability models, in this case our local and remote language models, will produce the same predictions. The output is the number of additional bits it would take to encode one model into the other. It is defined as follows (Manning et al., 2008, p. 231):

$$\mathrm{KLD}\left(\mathcal{T} \parallel \hat{\mathcal{T}}\right) = \sum_{t \in \mathcal{T}} P\left(t \mid \mathcal{T}\right) \cdot \log_2 \frac{P\left(t|\mathcal{T}\right)}{P\left(t|\hat{\mathcal{T}}\right)}, \tag{4.2}$$

where $\hat{\mathcal{T}}$ is the learned model and $\mathcal{T}$ the actual model. KLD has several disadvantages. Firstly, if a term occurs in one model, but not in the other it will produce zero or infinite numbers. Therefore, we apply Laplace smoothing, that simply adds one to all counts of the learned model $\hat{\mathcal{T}}$. This ensures that each term in the remote model exists at least once in the local model, thereby avoiding divisions by zero (Baillie et al., 2006a). Secondly, the KLD is asymmetric, which is expressed using the double bar notation. Manning and Schütze (1999, p. 304) reason that using

Jensen-Shannon Divergence (JSD) solves both problems. It is defined in terms of the KLD as (Dagan et al., 1997):

$$\mathrm{JSD}\left(\mathcal{T}, \hat{\mathcal{T}}\right) = \mathrm{KLD}\left(\mathcal{T} \middle\| \frac{\mathcal{T}+\hat{\mathcal{T}}}{2}\right) + \mathrm{KLD}\left(\hat{\mathcal{T}} \middle\| \frac{\mathcal{T}+\hat{\mathcal{T}}}{2}\right). \tag{4.3}$$

The Jensen-Shannon Divergence (JSD) expresses how much information is lost if we describe two distributions with their average distribution. This distribution is formed by summing the counts for each term that occurs in either model and taking the average by dividing this by two. Using the average is a form of smoothing that avoids changing the original counts in contrast with the Laplace smoothing used by the KLD. Other differences with the KLD are that the JSD is symmetric and finite. Conveniently, when using a logarithm of base 2 in the underlying KLD, the JSD ranges from 0.0 for identical distributions to 2.0 for maximally different distributions.

4.3 CAN WE DO BETTER THAN RANDOM?

In query-based sampling conventionally random single-term queries are used to obtain a document sample of each search server. In this section we explore if better resource descriptions can be obtained by using alternative single-term query construction strategies. We motivate our research from the perspective of search servers that offer access to information not easily reached by conventional crawling: the deep web.

The surface web consists of static, easily indexable, pages. The deep web consists of content that is generated dynamically by search servers in response to user queries. These queries are sent through a search form or interface. The simplest incarnation of a search interface presents a single free-text search field. These interfaces commonly provide access to an underlying database, for example: a database with descriptions of films or books. The number of deep web pages that provide search services like this is estimated by one study to be five hundred times larger with respect to the amount of surface web pages (Bergman, 2001).

The deep web is similar to the surface web: they both grow fast and offer a diverse range of information. However, there are some notable differences: the deep web is mostly structured and difficult to crawl. Some

deep web sites expose direct links to the underlying content, making indexing easier for traditional search engines. Despite this, the major search engines, Google, Yahoo and Bing, have trouble indexing the deep web. A 2004 study shows that they typically index only 37 percent of the available deep content (Chang et al., 2004). It has been suggested that a database-centred, discover-and-forward access model would be a better approach for enabling search of the deep web (He et al., 2007). Peer-to-peer information retrieval provides a model for this. However, it still requires a resource description of each server or uncooperative peer.

Recall that conventional query-based sampling revolves around sending random single-term queries to a search server to discover its language model. This section focuses on drawing terms in iterations that *follow* the first. We explore several strategies to choose single-term queries based on the constructed language model. Our research question is:

> 'Can query-based sampling be improved by using an alternative term selection strategy?'

The foundational work for acquiring resource descriptions via query-based sampling was done by Du and Callan (1998), Callan et al. (1999) and Callan and Connell (2001). They showed that a small sample of several hundred documents can be used for obtaining a good quality resource description of large collections consisting of hundreds of thousands of documents. They used uniform random term selection, meaning each term in the vocabulary has an equal probability of being chosen, but suggest that query terms could be selected from the learned language model using other criteria. This is what this section focuses on. The test collection used in their research, TREC-123, is not a web data collection. However, Monroe et al. (2002) showed that the query-based sampling approach also works very well for web data when using random queries.

Conventionally we compare the language model of the sample with that of the collection to assess the resource description quality. Sampling stops when a certain amount of documents has been obtained. Baillie et al. (2006a) propose using the predictive likelihood as a stopping criterion for sampling, instead of a fixed number of documents. They use a set of reference queries that represent typical information needs. Performance is measured with respect to this set of reference user queries. They thus shift focus with respect to the foundational work, which looked at

similarities between language models, to the expected real-world performance given a representative query set.

A problem with sampling is that some documents are more likely to be sampled than others. For example: longer documents are more likely to be sampled than shorter ones. This problem is commonly referred to as *sampling bias*. Bar-Yossef and Gurevich (2006) introduce two methods for obtaining an unbiased sample from a search system. The novelty of their approach is that they take into account the probability that a document is sampled. They use stochastic simulation techniques to produce nearly unbiased samples. Their goal is to produce realistic estimates of the sizes of search engine indices. The question is whether we really need unbiased samples for building resource descriptions (Baillie et al., 2006a). We do not further investigate this issue. However, we do use their rejection sampling technique for the query cardinality strategy presented later.

Ipeirotis and Gravano (2002) introduce a variant of query-based sampling called focused probing. They apply machine learning methods to learn the categorisation of a remote server based on the returned content. For example, a server that returns documents containing 'hepatitis' and 'MRSA', would be placed in the 'Health' category. This information can be used by a central system to narrow down the server selection for a user query. For example, all queries typically associated with health are forwarded to sites in this category, the returned results are merged and then presented to the user. In follow-up work Ipeirotis and Gravano (2008) improve their category-based approach with shrinkage. This approach exploits the hierarchy inherent in the categorisation used. This works by first traversing higher up the category hierarchy and smoothing the language models of servers, so that their representations are more similar. This is particularly effective for small servers, as their representations can augment each other this way. The negative effects of this, like introducing terms in a server's resource description that the server does not contain or underestimating term frequencies, are offset by the overall improvement in performance.

Ntoulas et al. (2005) have the goal of crawling as much content as possible from a hidden web resource and present an adaptive approach for term selection: they try to identify those terms that are most likely to return the most additional documents. This outperforms other approaches in terms of the number of documents seen. However, the goal of crawling differs from merely representing a resource. Madhavan et al. (2008) have

a similar goal and use tf.idf on the documents obtained so far to select keywords for subsequent iterations.

Other approaches go deeper into the functionality of the search interface itself. For example, Bergholz and Chidlovskii (2003) investigate how to find out what type of complex queries a free-text search field supports, for example: if it supports Boolean queries and what operators it supports for such queries. Chang et al. (2005) assume that the page on which the search interface resides contains clues about the underlying content. This is suitable for search forms that contain additional interface elements to narrow a search. In our research we restrict ourselves to the basic assumption that there is a single search field devoid of semantic clues. We send only single-term queries via this interface.

4.3.1 *Methodology*

In our experimental set-up we have a single remote server whose content we wish to estimate by sampling. This server provides a minimal search interface: it can only take queries and return search results consisting of a list of documents. Each document in this list is downloaded and used to build a resource description in the form of a vocabulary with frequency information, also called a language model (Callan and Connell, 2001). The act of submitting a query to the remote server, obtaining search results, downloading the documents, updating the local language model and calculating values for the evaluation metrics is called an *iteration*. An iteration consists of the following steps:

1. Pick a one-term query.
    a) In the first iteration our local language model is empty and has no terms. Thus, for bootstrapping, we pick a random term from an external resource.
    b) In subsequent iterations we pick a term based on a *term selection strategy*. We only pick terms that we have not yet submitted previously as query.
2. Send the query to the remote search server, requesting a maximum number of results ($n = 10$).
3. Download all the returned documents if $1 \leq n \leq 10$ or, if $n = 0$, go to step 5.

4. Update the resource description using the content of the returned documents.

5. Evaluate the iteration by comparing the remote language model with the local model (see metrics described in Section 4.2)

6. Terminate if a stopping criterion has been reached, otherwise go to step 1.

This set-up is similar to that of Callan and Connell (2001). However, there are several differences. During evaluation, step five, we compare all word forms in the index *unstemmed* which can lead to a slower performance increase. The database indices used by Callan and Connell contain only stemmed forms. Additionally, the underlying system uses fewer stop words. Callan and Connell use a list of 418 stop words. We use a conservative subset of this list consisting of 33 stop words that is distributed with Apache Lucene. We examine no more than 10 documents per query, as this is the number of search results commonly returned by search engines on the initial page nowadays. Callan concluded that the influence of the number of documents examined per query is small on average. Examining more documents appears to result in faster learning, although with more variation (Callan et al., 1999; Callan and Connell, 2001).

4.3.2 *Strategies*

In conventional query-based sampling, terms in iterations after the first one are selected at random from the language model constructed so far. This means that if we obtain 100 terms after the first iteration, each term in the language model has an equal probability of $1/100 = 0.01$ of being selected for the next iteration. The probability of an individual term being selected decreases as the vocabulary size increases. This is termed uniform random selection.

Vocabulary Frequency-Based Measures

A simple adjustment to selecting a random query term is selecting a term based on frequency information of the terms in the documents retrieved so far. Since a language model is more than just a vocabulary: it also contains term occurrence counts. We devised several selection strategies that exploit this. To illustrate we use the following example:

$[(lychee, 6), (okra, 3), (rambutan, 1)]$. This means that 60 percent of the terms seen so far are *lychee*, 30 percent are *okra* and the remaining 10 percent are *rambutan*. In a uniform random selection scenario, that disregards the frequencies, all terms are equally likely to be selected with a probability of approximately 33 percent. Choosing between them would be like rolling a three-sided dice. Let us investigate several alternatives to uniform random selection:

BIASED-RANDOM-COLLECTION Terms with a high frequency in the language model are more likely to be selected. This simply uses the frequency information to alter the selection probability. In this case the probability for *lychee* would be 0.6, for *okra* 0.3 and for *rambutan* 0.1.

LEAST-FREQUENT Select the term with the lowest frequency in the language model. In the example this would be *rambutan,* since: $0.1 < 0.3 < 0.6$.

MOST-FREQUENT Select the term with the highest frequency in the language model. In the example this would be *lychee*, since: $0.6 > 0.3 > 0.1$.

For these last two approaches, it is possible that there are multiple terms with the same lowest or highest frequency. For example: $[(apple, 0.1), (pear, 0.1), (banana, 0.1)]$. Selection among such terms, with the same frequency, is random. Since the frequency of terms in a vocabulary follows a Zipf distribution (Manning et al., 2008, p. 82), such random selection is more likely to occur among low frequency terms.

Document Frequency Based Measures

In each iteration we obtain a sample of documents. In the previous section we considered all documents together as one language model. The approaches in this section use the individual language models of each obtained document.

BIASED-RANDOM-DOCUMENT Each iteration we throw an n-faced dice where n is the number of terms in our local language model. However, the probability of a term being selected is proportional to its document frequency. The higher the document frequency, the more likely the term

will be selected. This is equivalent to biased-random-collection, but using document frequencies: in how many documents a term appears, instead of collection frequencies.

DOCUMENT INFORMATION RADIUS Each iteration we can use the statistical properties of the documents obtained so far. One way is to adapt the idea behind one of the metrics we use: Jensen-Shannon Divergence (JSD), explained in Section 4.2. We term this approach *document information radius* to prevent confusion with the metric. The intuition behind this method is: from all documents we first select the one that has the most terms in its language model that diverge from the language model of all sampled documents. From this document we select the term that diverges *least* from the combined language model as query.

Recall that the JSD calculates the divergence between two language models. In this approach, we compare the current local language model to a pooled language model of a group of documents. We define that $\mathcal{S}$ represents the entire sample of documents. The steps are as follows:

1. Determine document scores:
   a) Each document $d \in \mathcal{S}$ has an initial score of zero.
   b) For each term t in the vocabulary of $\mathcal{S}$ consider the pool of documents $\mathcal{S}_t \subseteq \mathcal{S}$ that contain that term at least once.
   c) Increase the score of each document $d \in \mathcal{S}_t$ with the JSD between the language model defined over the document pool $\mathcal{S}_t$ for each term t and the model of the entire sample $\mathcal{S}$. The document score is thus a sum of JSD values.
2. Select the document with the highest score.
3. From this document select the term t whose $\text{JSD}(\mathcal{S}_t, \mathcal{S})$ contributed least to the document score.

For example: if we have document A with score 0.5 and B with 1.0, we would select document B since it has a higher score. Thereafter we would select from the individual terms in document B the one with the lowest Jensen-Shannon Divergence.

DOCUMENT POTENTIAL Assume that it is preferable to use terms from many different documents as queries to obtain a good sample. Given this we need some way to avoid selecting query terms from one document too often. We need to determine the *document potential*. To do this we use appearance counts. Each iteration a document appears in the search results, its count is incremented by one. Thereafter, a query term is selected randomly from the document with the *lowest* appearance count. So, if we have document A with count 1 and document B with count 2, a term will be drawn randomly from the vocabulary of document A. If there are multiple documents with the same, lowest, appearance count, a document is first selected randomly and then a term from that document's vocabulary. So, if A and B both have count 1, we first select either document A or B with a fifty percent probability and then randomly select a term from the selected document as query.

This approach continually attempts to 'harvest' terms from documents that either appeared in more recent iterations or were neglected before. It indirectly also penalises long documents that have a higher probability of appearing in search results often.

Controlled Query Generation

Frequency information can also be used in more complex ways. Controlled query generation was proposed as a means of evaluating blind relevance feedback algorithms (Jordan et al., 2006). In this approach, queries with high discriminative power are generated from the documents seen so far. Kullback-Leibler Divergence (KLD), also called relative entropy, forms the basis for this. The KLD for each term is calculated between all the documents it appears in and the entire collection:

$$\text{score}\left(t, \mathcal{S}_t, \mathcal{S}\right) = P\left(t \mid \mathcal{S}_t\right) \cdot \log_2 \frac{P\left(t \mid \mathcal{S}_t\right)}{P\left(t \mid \mathcal{S}\right)}, \tag{4.4}$$

where t represents a single term, $\mathcal{S}_t$ the subset of documents of the sample in which t occurs and $\mathcal{S}$ the sample of the collection seen so far. Hence, $\mathcal{S}_t \subseteq \mathcal{S}$. The resulting score represents the power of a term to discriminate a subset of documents with respect to all other terms.

The highest scoring terms are used for querying. This might appear counter-intuitive, since terms with high discriminative power imply that these terms also return fewer and more specific documents. We will see

later that returning fewer documents each iteration does not necessarily yield poor modelling performance.

Query Cardinality

Bar-Yossef and Gurevich (2006, 2008b) present several approaches to obtain a uniform random sample from the index of large search engines to compare their coverage of the web. An important point they make is that of query overflow and underflow. A query that underflows is one that returns less than the number of desired results, whereas one that overflows returns more. For example: assume that we want each query to yield 10 documents, if only 5 are returned by a query it is a query that underflows. We call the number of results that a query yields the *cardinality* of the query.

The problem of underflow is relevant to query-based sampling. Each iteration we can send one query and get back results. Ideally, we would always want each query to return as many document as we request, since processing a query is a costly operation. If one query yields less than the amount of requested documents, we are partially wasting an iteration. This problem is not addressed in the foundational query-based sampling papers (Callan et al., 1999; Callan and Connell, 2001).

To avoid underflow we adopt the rejection sampling method that is illustrated in the pool-based sampler of Bar-Yossef and Gurevich (2006). To determine which query to send we adopt the following procedure:

1. Select a random term t from the set of terms $\mathcal{T}$ seen so far (the vocabulary of the local language model).

2. Count the number of documents $|\mathcal{S}_t|$ in our sample $\mathcal{S}$ that contain t. Use this count as an estimate of the number of results that will be returned.

3. If $|\mathcal{S}_t|$ is exactly the number of desired documents n, then accept and use this term. Otherwise, with probability $1 - |\mathcal{S}_t| / n$, reject the term and return to step 1, and with probability $|\mathcal{S}_t| / n$ accept and use this term.

Terms that are present in few documents in the sample obtained so far have a lower probability of being selected for obtaining more documents.

4.3.3 *Results*

In this section we report the results of our experiments. Because the queries are chosen randomly, we repeated each experiment 30 times. We derived the regression plots from scatter plots. Based on the shape of the data we fitted regression lines using $y = \log(x) + c$. The graphs show results for 100 iterations. We verified that the trends shown continue beyond the graph limit up to 125 iterations. In each iteration a variable number of documents is returned. We show the average number of returned results per iteration in separate graphs.

Figure 4.2 shows results on TREC-123 using the basic strategies: random, biased-random-collection, biased-random-doc, least-frequent and most-frequent. We see that the baseline, random term selection, performs quite well. While the biased approaches both perform worse than random, although using document frequencies instead of collection frequencies appears to be more optimal. Realise that document frequencies are in fact coarse collection frequencies. The only strategy that actually performs better than random is least-frequent. The opposite strategy, most-frequent, performs worst. Apparently, if we try to sample from a large underlying collection, using the least frequent terms as queries is the most optimal approach.

If we regard the region between the least-frequent and most-frequent strategies in each graph as an upper and lower limit, we can explain why the random strategy already works well. Since there are just a few high frequency terms and many low frequency terms, the probability of randomly selecting a less frequent term is quite large. This is the reason that random is so close to least-frequent. However, there is still a probability of selecting a frequently occurring term, which explains why random is slightly less optimal than always selecting the least frequent term. The number of returned results per iteration decreases more rapidly for least-frequent, but this apparently does not affect the strategy's capability of optimally sampling the underlying collection. The most-frequent strategy always returns ten results. However, these results are likely to be largely the same after each iteration. This redundancy in the results explains most-frequent's relatively poor performance.

Figure 4.3 shows the results for the advanced strategies. The random baseline is repeated in this graph for comparison. The document potential strategy performs quite poorly. Even though it manages to consistently

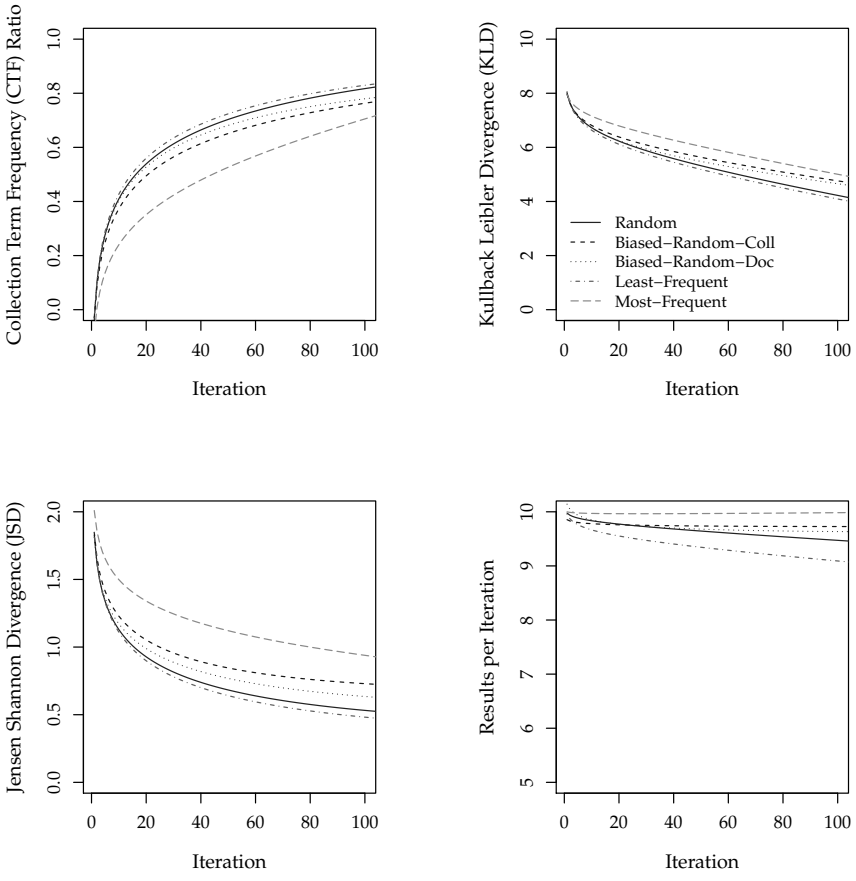


Figure 4.2: Results for TREC-123 for the basic frequency strategies. Legend in the top right graph. For CTF higher is better in terms of modelling performance, whereas for KLD and JSD lower is better.

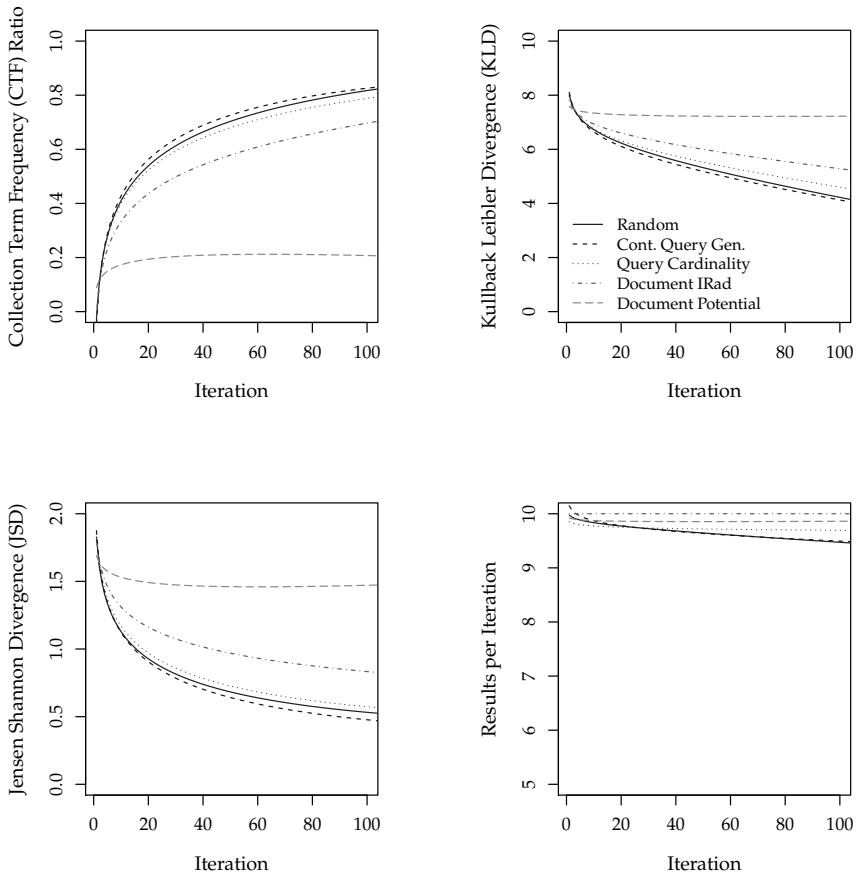


Figure 4.3: Results for TREC-123 for the advanced strategies. Legend in the top right graph.

select terms as query that retrieve a relatively stable number of results, these queries retrieve documents that poorly model the collection. The original scatter plots suggest that this strategy gets stuck on using query terms from one particular document in early iterations. The document information radius strategy always retrieves ten results, but similarly does not perform so well. Query cardinality performs just a little bit worse than random. In contrast, controlled query generation performs a bit better than random, comparable to least-frequent select.

Figure 4.4 and Figure 4.5 show the results for the strategies for WT2G. This collection is more representative for the Web. The pattern for the basic strategies, in Figure 4.4, is mostly the same as for TREC-123. The only difference is the results per iteration that varies much more for WT2G. The more heterogeneous nature of the corpus likely causes this higher variation. Indeed, the performance of least-frequent is quite good considering how few documents it retrieves in later iterations. The advanced strategies in Figure 4.5 show more difference. Controlled query generation, a strategy that performed better than random for TREC-123, performs poorly here. It appears to get stuck on terms that often retrieve the same documents. Apart from this, the result is similar to TREC-123 with query cardinality performing close to random.

To give further insight into why least-frequent select performs better, Figure 4.6 shows scatter plots of the JSD against the number of iterations and against bandwidth. These scatter plots show about 1000 samples of the source data per graph. Specifically, the two top graphs show the basis for the regression lines in the bottom-left graph of Figure 4.4. We can see from these graphs that random has more outliers than least-frequent select. We believe that the outliers for random are frequently occurring terms. Least-frequent never selects these terms. As a result its performance has less variance, which explains the better regression line.

We also plotted the JSD against the bandwidth consumption, shown as the bottom graphs in Figure 4.6. The horizontal axis in these graphs is not the number of iterations, but the combined size in kilobytes of the sample. It appears that the least-frequent strategy also shows more stable performance when plotted against bandwidth, suggesting that it retrieves higher quality documents in terms of representativeness measured by the JSD. For random, the quality of retrieved documents seems to vary more compared with least-frequent. This implies that least-frequent provides, on average, better results per unit of bandwidth used.

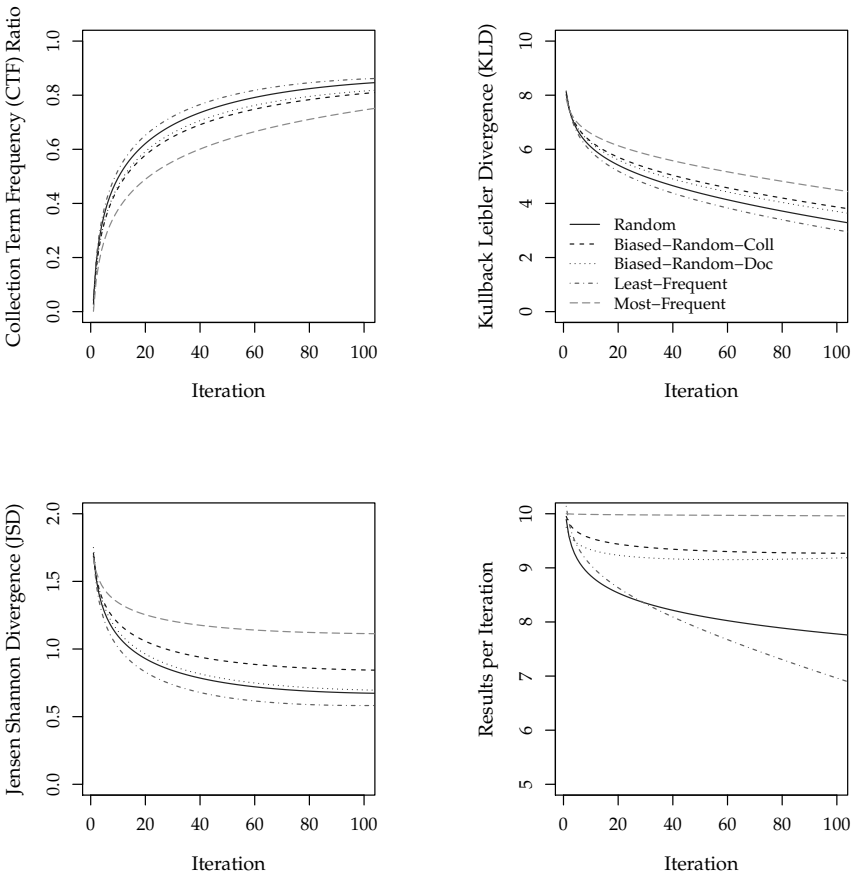


Figure 4.4: Results for WT2G for the basic frequency strategies. Legend in the top right graph.

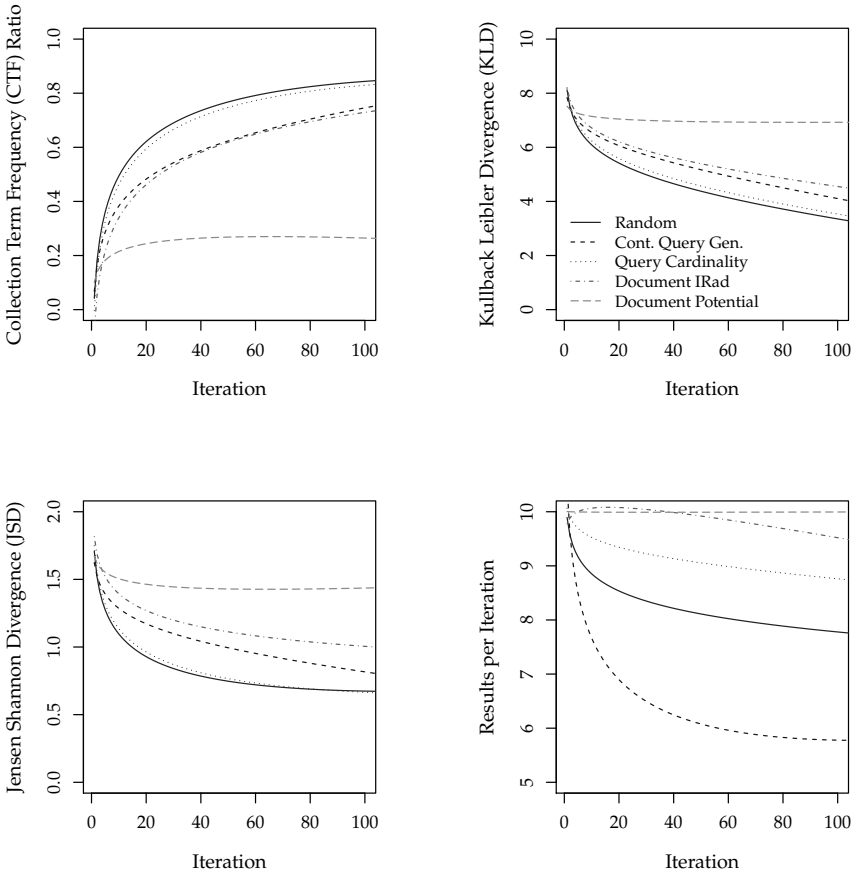


Figure 4.5: Results for WT2G for the advanced strategies. Legend in the top right graph.

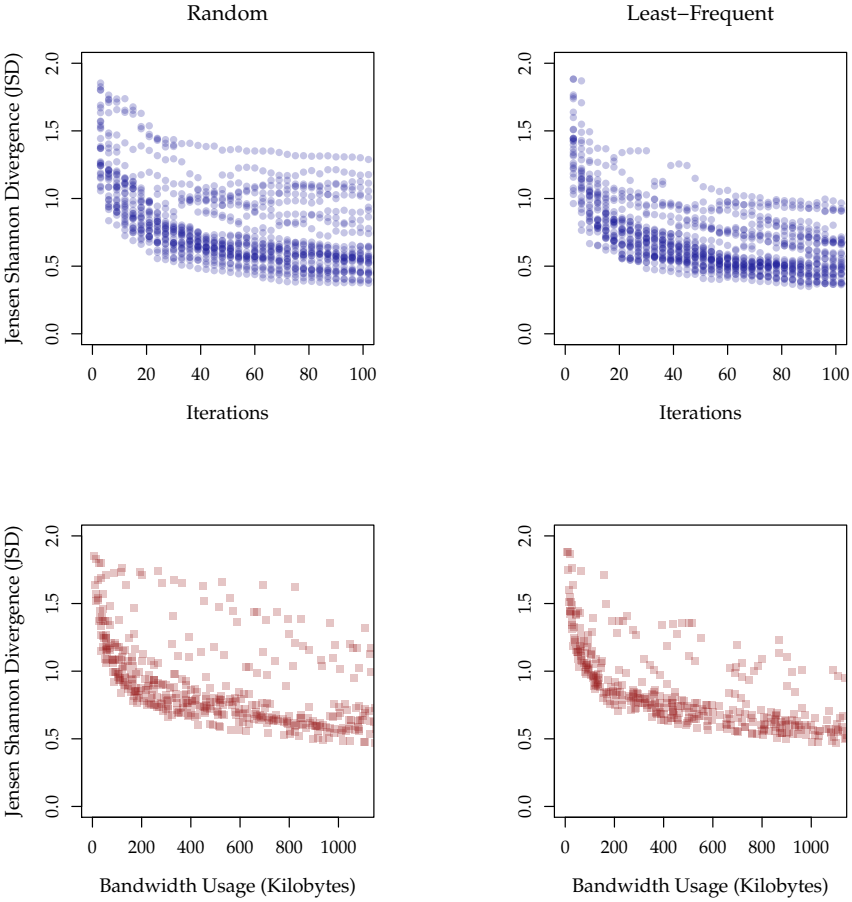


Figure 4.6: Scatter plots of the Jensen Shannon Divergence (JSD) against the number of iterations (top) and the bandwidth consumption (bottom) for WT2G. The left graphs are for the random strategy, the right graphs for least-frequent. Each graph is based on approximately 1000 samples.

4.3.4 *Conclusion*

In conventional query-based sampling, remote servers are sampled by sending random terms as queries, retrieving the results and using these to build a resource description. We presented several alternative approaches to using random terms. Selecting the least-frequent term in the language model as query yields slightly better performance than selecting a random term. Most other presented strategies did not show consistent improvement for the two test collections used.

The least-frequent approach outperforms other methods, while downloading less documents per iteration on average: it saves more bandwidth as the number of iterations increase. This suggests that we need to look beyond the quantity of data, the number of documents, used to build resource descriptions and pay more attention to quality and representativeness of those documents.

Nevertheless, the results also show that random selection is quite optimal and difficult to improve upon when relying just on the data provided by the search server. Furthermore, despite the good results for least-frequent, we did not study the influence of the size of the underlying database on the performance of the querying strategy. As such the result of this research should be seen as strictly applying to sites that index a large underlying collection in the same, or higher, order as the collections used in this research. The rationale behind this: using a least-frequent strategy on small collections may result in many iterations with a low number of search results, since rarely occurring terms might appear only in one specific document. For a large collection this problem is less pronounced. Low-frequency terms likely occur in at least some other documents. This also increases the chance that new documents are retrieved, which leads to a more accurate model in fewer iterations than the random approach.

4.3.5 *Future work*

We regard investigating the exact influence of the size of the underlying collection as important future work. Combining this with collection size estimation enables faster and less costly construction of resource descriptions tailored to collections of specific sizes. The research presented in this section is admittedly limited in scope. We would need to test on a higher number of recent collections with varying sizes to produce robust

experimental results. Other strategies for term selection could also be explored. However, we believe that further improvements, when using just the sampled language model, will be difficult.

We wish to emphasise that while a resource description needs to represent the underlying resource well, it need not necessarily be unbiased. In fact, some bias might make resource selection easier (Baillie et al., 2006a). The costs of term selection and query construction is an other aspect that should be more deeply evaluated.

Work that explores beyond the usage of the sampled language model has so far focused on using reference queries (Baillie et al., 2006a) and the beforementioned hierarchical topic-based content classification with shrinkage (Ipeirotis and Gravano, 2002, 2008). This may offer a promising basis for follow-up research.

Future work could also explore the construction of multi-term queries, which possibly return a more diverse set of documents each iteration (Jordan et al., 2006). Additionally, querying web forms that have more search fields, possibly not all of them free text, could be further explored (Senellart et al., 2008).

4.4 QUERY-BASED SAMPLING USING SNIPPETS

As we have learned in this chapter so far: query-based sampling is a technique that exploits only the native search functionality provided by servers to construct a resource description. This description is based on the downloaded content of a small subset of documents the server returns in response to queries (Callan et al., 1999). In this section we present an approach that requires no additional downloading beyond the returned results, but instead relies solely on information returned *as part* of the results: the snippets.

In conventional query-based sampling, the first step is sending a query to a server. The server returns a ranked list of results, of which the top n most relevant documents are downloaded and used to build a resource description. Queries are randomly chosen, the first from an external resource and subsequent queries from the description built so far. This repeats until a stopping criterion is reached (Callan et al., 1999; Callan and Connell, 2001). Disadvantages of downloading entire documents are that it consumes more bandwidth, is impossible if servers do not return full documents and does not work when the full documents themselves

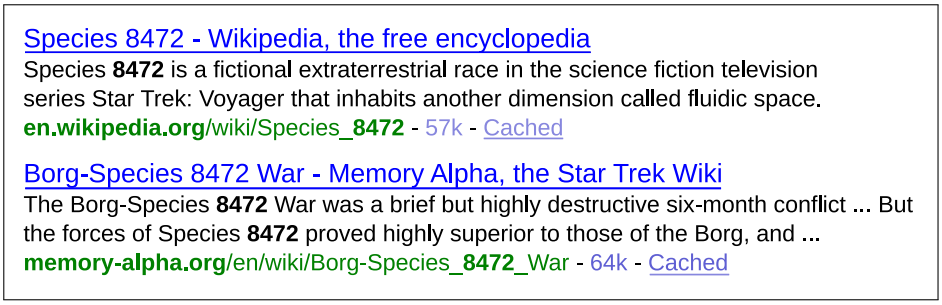


Figure 4.7: Example snippets. From top to bottom: each snippet consists of an underlined title, a two line summary and a link.

are non-text: multimedia with short summary descriptions. In contrast, some data always comes along 'for free' in the returned search results: the snippets. A snippet is a short piece of text consisting of a document title, a short summary and a link as shown in Figure 4.7. A summary can be either dynamically generated in response to a query or is statically defined (Manning et al., 2008, p. 157). We postulate these snippets can also be used for query-based sampling to build a language model. This way we can avoid downloading entire documents and thus reduce bandwidth usage and cope with servers that return only search results or contain multimedia content. However, since snippets are small, we need to see many of them. This means sending more queries compared with the full document approach. While this increases the query load on the remote servers, it is an advantage for live systems that need to sample from collections that change over time, since it allows continuously updating the language model, based on results of live queries.

Whether the documents returned in response to random queries are a truly random part of the underlying collection is doubtful. Servers have a propensity to return documents that users indicate as important and the number of in-links has a substantial correlation with this (Azzopardi et al., 2007). This may not be a problem, as it is preferable to know only the language model represented by important documents, since the user is likely to look for those (Baillie et al., 2006a). Bar-Yossef and Gurevich (2008b) focus on obtaining uniform random samples from large search engines, in order to estimate their size and overlap. Thomas and Hawking (2007) evaluated this in the context of obtaining resource descriptions and found it does not consistently work well across collections.

The approach we take has some similarities with prior research by Paltoglou et al. (2007). They show that downloading only a part of a document can also yield good modelling performance. However, they download the first two to three kilobytes of each document in the result list, whereas we use small snippets and thus avoid any extra downloading beyond the search results.

Our main research question is:

> 'How does query-based sampling using *only snippets* compare to downloading full documents in terms of the learned language model?'

We show that query-based sampling using snippets offers similar performance compared with using full documents. However, using snippets uses less bandwidth and enables continuously updating the resource description at no extra cost. Additionally, we introduce a method to establish the homogeneity of a corpus, which can be useful for determining a corpus-specific sampling strategy.

4.4.1 *Methodology*

In our experimental set-up we have one remote search server which content we wish to estimate by sampling. This server can only take queries and return search results. For each document a title, snippet and download link is returned. These results are used to locally build a resource description in the form of a vocabulary with frequency information at the tracker. The act of submitting a query to the remote server, obtaining search results, updating the local language model and calculating values for the evaluation metrics is called an *iteration*. An iteration consists of the following steps:

1. Pick a one-term query.
    a) In the first iteration our local language model is empty and has no terms. In this case we pick a random term from an external resource as query.
    b) In subsequent iterations we pick a random term from our local language model that we have not yet submitted previously as query.

2. Send the query to the remote server, requesting a maximum number of results ($n = 10$). In our set-up, the maximum length of the document summaries may be no more than 2 fragments of 90 characters each ($|s| \leq 2 \cdot 90$).
3. Update the resource description using the results ($1 \leq n \leq 10$).
   a) For the full document strategy: download all the returned documents and use all their content to update the local language model.
   b) For the snippet strategy: use the title and summary of each document in the search results to update the local language model. If the same document appears multiple times in search results, do not use its title and use its summary only if it differs from previously seen summaries of that document.
4. Evaluate the iteration by comparing the unstemmed language model of the remote server with the local model (see metrics described in Section 4.2).
5. Terminate if a stopping criterion has been reached, otherwise go to step 1.

Since the snippet approach uses the title and summary of each document returned in the search results, the way in which the summary is generated affects performance. We use Apache Lucene that generates keyword-in-context document summaries (Manning et al., 2008, p. 158). These summaries are constructed by using words surrounding a query term in a document, without keeping into account sentence boundaries. For all experiments, the summaries consisted of two keyword-in-context fragments of maximally ninety characters. This length is similar to what is used by modern web search engines for summary generation. One might be tempted to believe that snippets are biased due to the fact that they commonly also contain the query terms. However, in full-document sampling the returned documents *also* contain the query and have a similar bias, although mitigated by document length.

4.4.2 *Results*

In this section we report the results of our experiments. Since the queries are chosen randomly, we repeated each experiment 30 times.

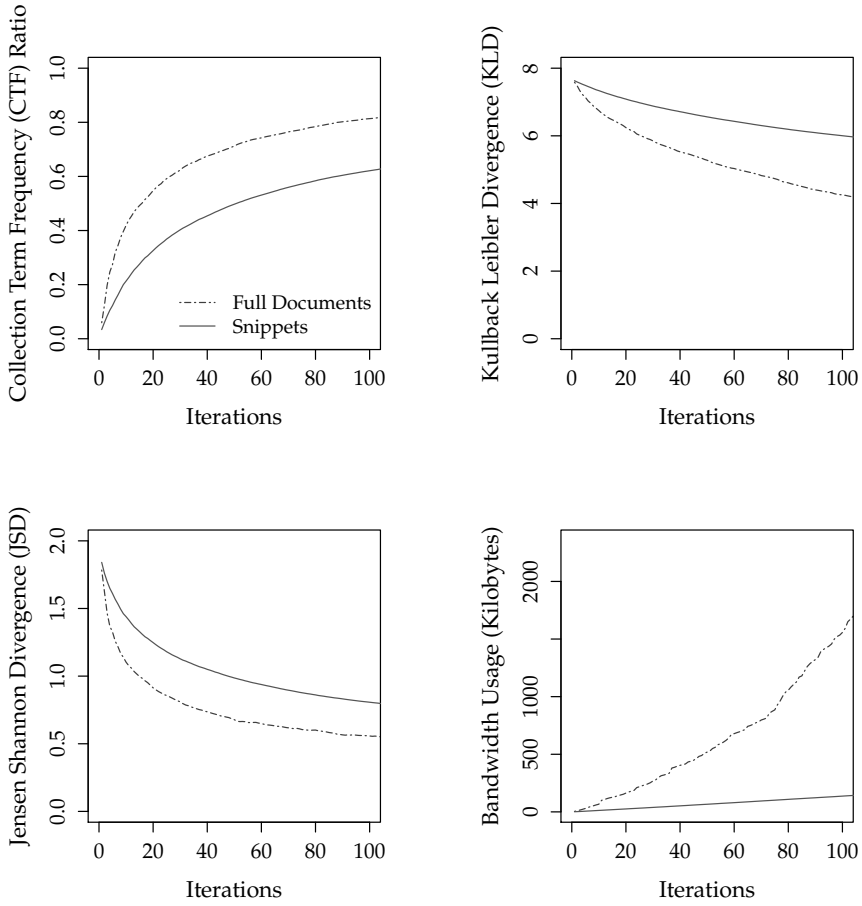


Figure 4.8: Results for TREC123. The graphs show the CTF, KLD, JSD and bandwidth usage, plotted against the number of iterations, both for the full document and snippet-based approach. The legend is shown in the top left graph.

Figure 4.8 shows our results on TREC123 in the conventional way for query-based sampling: a metric against the number of iterations on the horizontal axis (Callan and Connell, 2001). We have omitted graphs for WT2G and the Wikia collections as they are highly similar in shape.

As the bottom right graph shows, the amount of bandwidth consumed when using full documents is much larger compared with using snippets. Full documents downloads each of the ten documents in the search results, that can be potentially large. Downloading all documents also uses many connections to the server: one for the search results plus ten for the documents, whereas the snippet approach uses only one connection for transferring the results and performs no additional downloads.

The fact that the full documents approach downloads a lot of extra information, results in it outperforming the snippet approach for the defined metrics as shown in the other graphs of Figure 4.8. However, comparing this way is unfair. Full document sampling performs better, simply because it acquires more data in fewer iterations. A more interesting question is: how effectively do the approaches use bandwidth?

Bandwidth

Figures 4.9 and 4.10 show the metrics plotted against bandwidth usage. The graphs are 41-point interpolated plots based on experiment data. These plots are generated in a similar same way as recall-precision graphs, but they contain more points: 41 instead of 11, one every 25 kilobytes. Additionally, the recall-precision graphs, as frequently used in TREC, use the maximum value at each point (Harman, 1993). We use linear interpolation instead that uses averages.

Figure 4.9 shows that snippets outperform the full document approach for all metrics. This seems to be more pronounced for WT2G. The underlying data reveals that snippets yield much more stable performance increments per unit of bandwidth. Partially, this is due to a larger quantity of queries. The poorer performance of full documents is caused by variations in document length and quality. Downloading a long document that poorly represents the underlying collection is heavily penalised. The snippet approach never makes very large 'mistakes', because its document length is bound to the maximum summary size.

TREC123 and WT2G are very large heterogeneous test collections as we will show later. The WIKI collections are more homogeneous and have

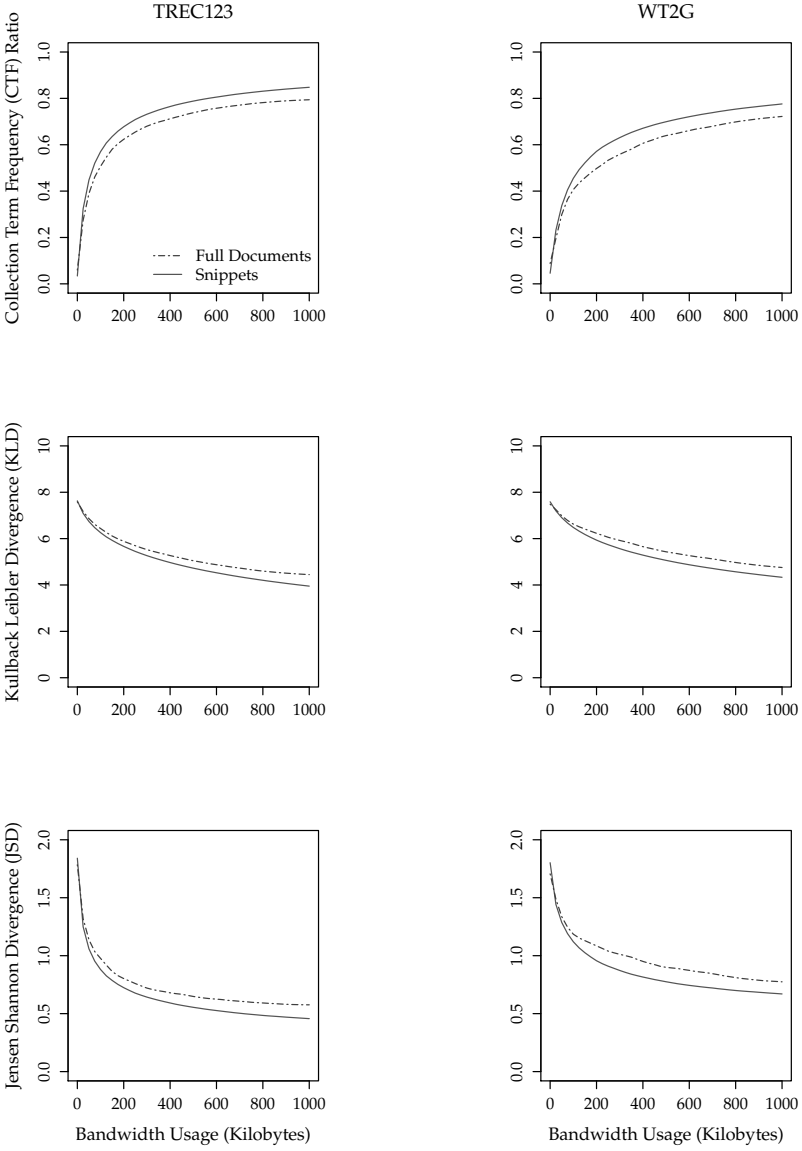


Figure 4.9: Interpolated plots for all metrics against bandwidth usage up to 1000 kilobytes. The left graphs show results for TREC123, the right for WT2G. Axis titles are shown on the left and bottom graphs, the legend in the top left graph.

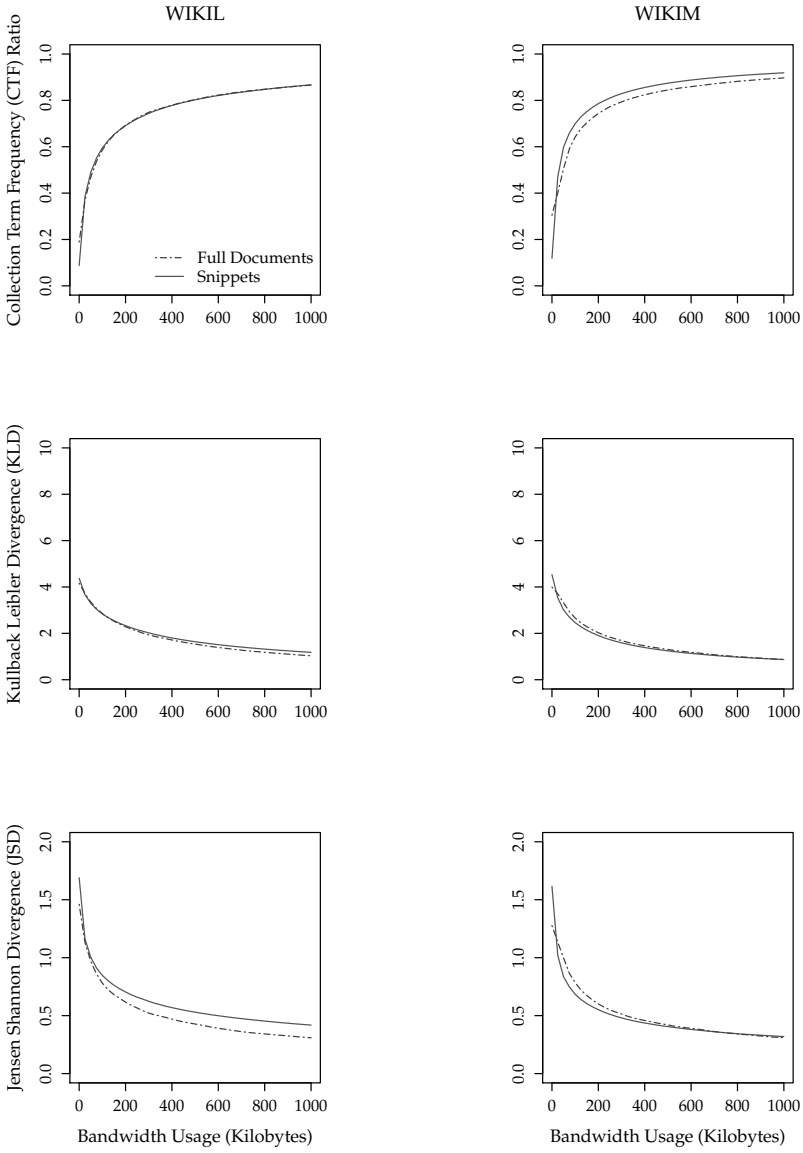


Figure 4.10: Interpolated plots for all metrics against bandwidth usage up to 1000 kilobytes. The left graphs show results for WIKIL, the right for WIKIM. Axis titles are shown on the left and bottom graphs, the legend in the top left graph.

different document length distribution characteristics. In Figure 4.10 we see that the performance of snippets on the WIKIL corpus is worse for the JSD, but undecided for the other metrics. For WIKIM performance measured with CTF is slightly better and undecided for the other metrics. Why this difference? We conducted tests on several other large size Wiki collections to verify our results. The results suggest that there is some relation between the distribution of document lengths and the performance of query-based sampling using snippets. In Figure 4.1 we see a peak at the low end of documents lengths for WIKIL. Collections that exhibit this type of peak all showed similar performance to WIKIL: snippets performing slightly worse especially for the JSD. In contrast, collections that have a distribution like WIKIM, also show similar performance: slightly better for CTF. Collections that have a less pronounced peak at higher document lengths, or a more gradual distribution, appear to perform at least as good or better using snippets compared to full documents.

The reason for this is that as the document size decreases and approaches the snippet summary size, the full document strategy is less heavily penalised by mistakes. It can no longer download large unrepresentative documents, only small ones. However, this advantage is offset if the document sizes equal the summary size. In that case, the full document approach would use twice the bandwidth with no advantage: once to obtain the search results, with summaries, and once again to get the entire documents which are identical to the snippet summaries.

Homogeneity

While WIKIM has a fairly smooth document length distribution, the performance increase of snippets over full documents with regard to the JSD and KLD metrics is not the same as that obtained with TREC123 and WT2G. This might be caused by the homogeneous nature of the collection. Consider that if a collection is highly homogeneous, only a few samples are needed to obtain a good representation. Every additional sample can only slightly improve such a model. In contrast, for a heterogeneous collection, each new sample can improve the model significantly.

So, how homogeneous are the collections that we used? We adopt the approach of Kilgarriff and Rose (1998), that split the corpus into parts and compare those, with some slight adjustments. As metric we use the Jensen-Shannon Divergence (JSD), explained in Section 4.2, also used by

Table 4.2: Collection Homogeneity Expressed as Jensen-Shannon Divergence

Collection name	μ JSD
TREC123	1.11
WT2G	1.04
WIKIL	0.97
WIKIM	0.85

Lower scores indicate more homogeneity ($n = 100, \sigma = 0.01$).

Eiron and McCurley (2003) for the same task. The exact procedure we used is as follows:

1. Select a random sample $\mathcal{S}$ of 5000 documents from a collection.

2. Randomly divide the documents in the sample $\mathcal{S}$ into ten bins: $s_1 \ldots s_{10}$. Each bin contains approximately 500 documents.

3. For each bin s_i calculate the Jensen-Shannon Divergence (JSD) between the *bigram* language model defined by the documents in bin s_i and the language model defined by the documents in the remaining nine bins. Meaning: the language model of documents in s_1 would be compared to that of those in $s_2 \ldots s_{10}$, et cetera. This is known as a leave-one-out test.

4. Average the ten JSD scores obtained in step 3. The outcome represents the homogeneity. The lower the number, the more self similarity within the corpus, thus the more homogeneous the corpus.

Because we select documents from the collection randomly in step 1, we repeated the experiment ten times for each collection. The averaged results in table 4.2 show that the large collections we used, TREC123 and WT2G, are more heterogeneous compared with the smaller collections WIKIL and WIKIM. It appears that WIKIL is more heterogeneous than WIKIM, yet snippet-based sampling performs better on WIKIM. This could be caused by the difference in document length distributions discussed earlier. It seems as if query-based sampling using snippets is better suited towards heterogeneous collections with a smooth distribution of document lengths. However, this needs to be further investigated.

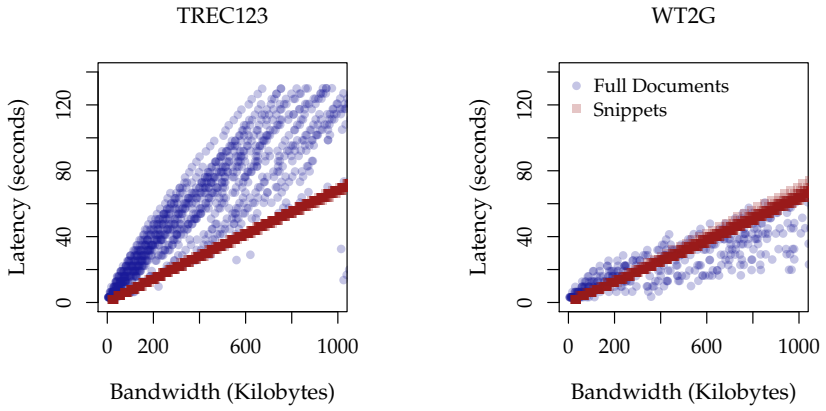


Figure 4.11: Latency of full documents and snippets, lower is better. The thick diagonal bar made up of squares shows the performance of snippets. The more thinly spread out circles the performance of full documents.

Latency

Now that we have plotted everything against bandwidth, we are ignoring an other important factor: latency. Sending a query and returning results takes time. Since the snippet approach needs many iterations to consume the same amount of bandwidth as full documents, it might impose additional delay. We wonder: at the same bandwidth usage, does the snippet approach incur more latency compared with full documents?

An argument in favour of snippet query-based sampling is that nothing extra needs to be downloaded: it relies solely on the snippets. In a real-world system these snippets come along 'for free' with the results returned in response to each query posed to the system. We assume a set-up where obtaining search results costs 100 milliseconds (ms), downloading a document costs 100 ms for setting up the connection and 1 ms for each kilobyte downloaded (assuming an 8 megabit connection). The same server that provides the search service also stores all documents. Each iteration the full document strategy costs at least 1050 ms: 100 ms for the search results and 950 ms for downloading the, on average, 9.5 documents. Snippets costs only 100 ms for obtaining the search results.

Figure 4.11 shows the comparison in scatter plots. The performance

variation of the full document approach is large. TREC123 contains many small documents which may explain its consistent poor performance with full documents: it would need to initiate more downloads. However, this requires further investigation. The snippet approach performs significantly better. However, for WT2G the full document approach is more optimal. WT2G has a more even document length distribution with fewer small documents as shown in Figure 4.1. The result thus depends on the characteristics of the documents that make up the collection. Selecting different costs also leads to different performance. For example, with a transfer rate of one kilobyte per 50 ms the snippet approach always has the lowest latency for both data sets. Since the full document approach requires initiating more connections and also requires more bandwidth, it is more sensitive to the characteristics of the network connection.

4.4.3 *Conclusion and Future Work*

We have shown that query-based sampling using snippets is a viable alternative for conventional query-based sampling using full documents. This opens the way for distributed search systems that do not need to download documents at all, but instead solely operate by exchanging queries and search results. Few adjustments are needed to existing operational federated information retrieval systems, as the remote search engines and the mediator already exchange snippets. For peer-to-peer information retrieval systems, peers could occasionally submit snippet batches to the tracker, thereby helping to keep resource descriptions up-to-date. Alternatively, they could first perform resource selection locally based on the snippets seen, thereby offloading the tracker. Our research implies that the significant overhead incurred by downloading documents in today's prototype federated information retrieval systems can be completely eliminated. This also enables modelling servers from which full documents can not be obtained and those that index multimedia content. Furthermore, the central server can continuously use the search result data to keep its resource descriptions up to date without imposing additional overhead, naturally coping with changes in document collections that occur over time. This also provides the extra iterations that snippet query-based sampling requires without extra latency.

Compared with the conventional query-based sampling approach, our snippet approach shows equal or better performance per unit of band-

width consumed for most of the test collections. The performance also appears to be more stable per unit of bandwidth used. Factors that influence the performance are the document length distribution and the homogeneity of the data. Snippet query-based sampling performs best when document lengths are smoothly distributed, without a large peak at the low-end of document sizes, and when the data is heterogeneous.

Even though the performance of snippet query-based sampling depends on the underlying collection, the information that is used always comes along 'for free' with search results. No extra bandwidth, connections or operations are required beyond simply sending a query and obtaining a list of search results. Herein lies the strength of the approach.

We believe that the performance gains seen in the various metrics leads to improved selection and merging performance. However, this is something that could be further explored. A measure for how representative the resource descriptions obtained by sampling are for real-world usage would be very useful. This remains an open problem, also for full document sampling, even though some attempts have been made to solve it (Baillie et al., 2009).

An other research direction is the snippets themselves. Firstly, how snippet generation affects modelling performance. Secondly, how a query can be generated from the snippets seen so far in more sophisticated ways. This could be done by attaching a different priority to different words in a snippet. Finally, the influence of the ratio of snippet to document size could be further investigated.

5

SELECTING COOPERATIVE PEERS

'If you want to be incrementally better: be competitive. If you want to be exponentially better: be cooperative.'

Anonymous

A peer-to-peer information retrieval network essentially consists of many small cooperative search engines. Given a query we need to direct it to one or more peers with relevant results. In this chapter, we explore how cooperative peers can be represented for this selection, what the effect is in terms of search result relevance and what the costs of representations are in terms of bandwidth and storage space.

An open challenge for peer-to-peer information retrieval networks is routing a query to one or more peers that have sufficient relevant search results. In this chapter, we focus on the special party that facilitates query routing: the tracker. This can either be a single machine, a set of special peers in the network or, at the other extreme: all peers. The sole purpose of the logical tracker in our peer-to-peer information retrieval network is providing advice for query routing. When a peer has a query it first sends it to the tracker to obtain a list of candidate peers from which it can obtain search results. However, the tracker needs some way to match queries to candidate peers. The central question this chapter tries to answers is:

> 'What is the best representation for a peer in terms of retrieval effectiveness and costs?'

The effectiveness is measured by the number of relevant documents returned. Since we consider a web search environment in this thesis, we focus primarily on precision at early ranks. The costs are expressed in terms of bandwidth and storage. Transferring more data between the peer and the tracker incurs a higher cost.

Lu and Callan (2003) explore three approaches for representing individual peers as well as groups of peers: based on the titles of documents, on vocabularies (match-based) and on vocabularies with frequency information: language models (content-based). They assume a super peer network where queries are first routed between super peers and then to leaf peers. Assuming that a leaf peer always forwards a query to the super peer, there are three retrieval steps. Firstly, selecting alternative super peers to forward the query to, this is evaluated using both simple flooding as well as content-based selection. Secondly, selecting appropriate leaf nodes for a query, evaluated using all three approaches described. Finally, the actual document retrieval itself, performed using the title and content-based approaches. Conceptually, the role of the super peers is equivalent to that of the tracker in our design, although with the additional challenge of forwarding queries between super peers. Lu and Callan (2003) evaluate their approach on a host-based split of WT10g (Bailey et al., 2003) with about 1.4 million documents divided over 2,500 peers. They conclude that using content-based resource selection, for all three steps, is substantially better than using other alternatives when considering recall, precision and the communication overhead. This is closely followed by using flooding between super peers combined with content-based selection for leaf peers and document retrieval. However, requiring a match for all query terms, when both leaf peer selection and document selection use the title-based approach, offered comparable precision. Additionally, the vocabulary (match-based) approach also offered only slightly lower precision, and may be a good alternative if content-based representation is not possible.

Witschel (2008) investigates peer representations in peer-to-peer information retrieval networks. He emphasises that it is important for resource descriptions to remain compact as they need to be passed around the network frequently. He experiments with pruning resource descriptions, adjusting them based on the query stream, as well as based on query expansion. The datasets used are CiteSeer and the German medical corpora Ohsumed and GIRT-4. Explored resource selection techniques are:

random, based on unpruned CORI representations and based on CORI representations pruned to a fixed selection of the top n terms, where n varies between 10 and 640. His experiments use selections ranging from 1 to 100 peers. He concludes that resource descriptions can be pruned without much performance loss in terms of mean average precision: about -0.05 when using the top 10 terms. Unfortunately, results for precision at specific cut-offs are not provided and the paper relies on mean average precision and relative precision as main metrics, which makes it difficult to compare results with our work. Using query expansion does not seem to improve results and in fact produces significantly worse results most of the time. In contrast, learning and adapting the representations based on an actual query log improves performance with at least 5 percent or more consistently.

Klampanos and Jose (2004) let each peer cluster its own documents and, based on these locally created document clusters, let each peer join one or more matching content-aware groups. These are groups of peers that cover a particular topic: clusters at the network level. Each super peer stores descriptors of these groups and matches queries against them. The group descriptors are term frequency vectors, derived from the local document clusters of all associated peers. The authors used the ad-hoc TREC collections of TREC 6 and 7 for evaluation, a network with 560,000 documents divided over 1,500 peers. Their approach offers substantially better performance, in terms of precision and recall, compared with centralised search. Furthermore, the number of contacted peers is low: usually one or two, sometimes three and rarely six.

Resource representation and collection selection have been extensively studied in the context of federated information retrieval (Callan, 2000). In federated information retrieval, a mediator holds representations of all resources, search servers, and scores incoming queries against these representations to determine what servers should actually perform retrieval. In contrast with the peer-to-peer tracker, the mediator also forwards the queries to the servers and merges the results from multiple servers into one result list. The mediator is essentially a façade: client machines that issue queries are not aware of the many search servers 'behind' the mediator, see Section 2.6.2 for more details.

Early federated information retrieval approaches represented each resource as a single large document. These documents were stored in an index and queries scored against this index directly yield a ranking

of appropriate servers. This method was used by CORI, CVV and KL-Divergence (Callan et al., 1995; Yuwono and Lee, 1997; Xu and Croft, 1999). Later approaches considered the individual documents in each collection: GlOSS, DTF, ReDDE and CRCS (Gravano et al., 1999; Nottelmann and Fuhr, 2007; Si and Callan, 2003a; Shokouhi, 2007). In ReDDE a set of sample documents is retrieved from each server, these are stored in a central sample index. When a query is scored against this index, the result is a list of documents. The intuition behind ReDDE is that each document in the result set is essentially a vote for the resource the document belongs to. Votes are accumulated by going down the ranked list and counting votes for each resource. The weight of the vote is proportional to the size of the resource. Hence, there is a bias towards large collections.

The work discussed so far considers only cooperative resources. In contrast, when resources are uncooperative one can resort to query-based sampling (Callan et al., 1999), discussed in Chapter 4, and resource size estimations (Shokouhi et al., 2006b). However, in this chapter we assume a cooperative scenario.

5.1 APPROACH

In this chapter we describe and test a variety of peer selection methods. We assume the scenario described previously: a tracker whose sole purpose is to make a best estimate as to what query should be handled by which peer. We assume a cooperative environment, where each peer can provide its own representation, also termed a *resource description*. The purpose of this exploration is to get an impression of the performance of each method in terms of relevance and the associated bandwidth and storage costs. In the following sections we introduce the methods that we will test in the experimental section and the costs they incur.

5.1.1 *Methods*

Random

When one does not know anything about peers, the simplest strategy is to randomly return a set of peers for a particular query. This strategy would not be a bad choice for a scenario where all the peers have identical indices, or when they are very large to begin with, as it would provide a natural way to perform load balancing: random selection does not require

any cooperation and also has zero storage-cost overhead for the tracker. However, in the scenario that we are considering, a heterogeneous peer-to-peer network, this is not true: for each query there are only a few peers that hold relevant documents.

In peer-to-peer networks that use a single layer of strict local indices only (see Section 2.4), which is a different architecture than the one we consider here, randomly forwarding the query throughout the overlay network has been found to be an effective strategy (Lv et al., 2002), see also Section 3.1.7. However, while the tracker in our experiment also performs query routing, forwarding through an overlay network is different. In a random walk the random selection at each forwarding step is over a small set of neighbouring peers, whereas in our case we select a random peer from the entire network.

Size-based

If we could obtain a single statistic with regard to a particular peer, what should it be? Perhaps its size: intuitively, larger collections hold more documents and thus also more relevant documents. We conduct a simple experiment, where the tracker returns a static list with the n largest peers. In a cooperative environment each peer could easily provide this information accurately at little cost. In uncooperative environments this can be estimated (Shokouhi et al., 2006b). The tracker needs to hold only one numeric value for each peer.

Vocabulary-based

Using the vocabulary of each peer is an established idea that has been used previously in peer-to-peer information retrieval experiments (Gravano et al., 1997, 1999; Lu and Callan, 2003). Using all terms in the vocabulary is wasteful from a bandwidth and storage perspective, and also unnecessary. It has been shown that removing all single term occurrences can work as well or better than using all terms (Lu and Callan, 2003). In our experiment we represent each peer by a limited selection of the top n terms in its vocabulary. We select the top n in three different ways: by count (tf), by local tf.idf and by global tf.idf. For the count approach we determine the score of each term as follows:

$$\text{termscore}\left(t, \mathcal{C}_p\right) = \sum_{d \in C_p} \text{tf}\left(t, d\right), \tag{5.1}$$

where t is a term, d is a document and $\mathcal{C}_p$ is a peer's document collection. The function $\text{tf}\left(t, d\right)$ gives the occurrence count of term t in document d. For the tf.idf approaches we use a slightly different formula:

$$\text{termscore}\left(t, \mathcal{C}\right) = \sum_{d \in \mathcal{C}} \text{tf}\left(t, d\right) \cdot \text{idf}\left(t, \mathcal{C}\right), \tag{5.2}$$

where the variables are identical to the previous formula, with one addition: the idf function returns the inverse document frequency for term t in collection $\mathcal{C}$. Its formal definition: $\text{idf}\left(t, \mathcal{C}\right) = \log\left(\left|\mathcal{C}\right| / \text{df}\left(t, \mathcal{C}\right)\right)$, where the df function gives the number of documents in which term t appears in collection $\mathcal{C}$ and $\left|\mathcal{C}\right|$ is the total number of documents in collection $\mathcal{C}$. For the local tf.idf variant, $\mathcal{C}$ is simply the peer's document collection $\mathcal{C}_p$, whereas for global tf.idf $\mathcal{C}$ represents the collection of all peer in the network $\mathcal{C}_g$.

The count-based approach is conceptually the least complex: we rank all terms in the resource by descending count and take the top n. In the local tf.idf approach the idf is based solely on the peer's own collection, whereas for the global tf.idf the idf is over the collections of all peers combined. Hence, the global tf.idf variant is expensive, because it is the only one that requires communication with other peers, or the tracker, to obtain global inverse document frequencies.

After we have selected the top n terms the actual count or tf.idf information is removed and the vocabulary is used as a flat list of terms for peer selection: a type of large document representation. The peers thus need to only transmit a list of terms to the tracker, no additional information is needed. The tracker stores the resource description of each peer in a single index and scores each incoming query against this index to determine the best servers for providing relevant search results for this query.

Term-weighted Vocabulary-based

In the previous approach we removed the term counts, but what happens if we do use these? In this approach we experiment with peer vocabularies that include frequency information. We store the top $n = 1{,}000$ terms

of each peer, including their frequencies. Each query is first split into separate terms and then matched against the representation of each peer. The scoring function is the same as that used for full language models and has the following formal definition (Lu and Callan, 2003):

$$R_p\left(\mathcal{Q}, \mathcal{L}_p, \mathcal{L}_g\right) = \sum_{t \in \mathcal{Q}} \log\left\{\lambda P\left(t \mid \mathcal{L}_p\right) + (1-\lambda)\, P\left(t \mid \mathcal{L}_g\right)\right\}, \qquad (5.3)$$

where $\mathcal{Q}$ is a query, t is a term within the query, $\mathcal{L}_p$ is the language model of peer p and $\mathcal{L}_g$ is the model of the entire collection: either ClueWeb09-B or Gov2 in our case. The influence of the smoothing background model $\mathcal{L}_g$ is controlled by the constant λ. Each peer is scored this way, and from the result we select the top 10 peers.

Query Log based

We start from the scenario where each peer submits its query to the tracker to obtain advice. Let us assume that the tracker actually uses this information to route future queries to the peers that submitted them previously. The underlying assumption is that those peers either already hold or have obtained relevant documents from other peers. The tracker thus needs to maintain a query history for each peer.

In this experiment, we simulate the query log using the approach of Dang and Croft (2010). The generated log is based on the anchors between peers. We use only unique anchors, so if one anchors occurs multiple times between two peers, it is used only once as a query. However, the count information is used for selection as follows: we select the top n queries by count to form the actual resource description at the tracker. Particularly for small peers this may lead to many ties in the ranking due to the low query counts. To resolve ties we prefer longer queries over smaller ones, as these are likely to be more descriptive. Note that, in contrast with the vocabulary-based approach, a query can consist of more than one term.

Document Sample based

An often used approach is taking a random sample of documents from each peer, storing these in a central sample index, scoring each query

against this index and deriving a ranking of peers from the resulting ranking of documents. This is often referred to as the small document approach, as each peer is represented by a small set of sampled documents. We test two algorithms that use a central sample index: ReDDE and CRCS (Si and Callan, 2003a; Shokouhi, 2007). We also adapt ReDDE for use with a simple reputation scheme, explained later. The costs for the tracker of maintaining document samples is difficult to predict precisely as document sizes vary and the sample is drawn randomly.

ReDDE uses the central sample index to estimate how many relevant documents it *expects* each peer to contain. This proceeds as follows: the query is first scored against the central sample index, this yields a ranking of the sample documents. ReDDE considers each document in the top ranking to represent m matching documents in the actual peer, where m is the size of the collection divided by the size of the sample. For example, if we have a sample $\mathcal{S}_p$ of 10 documents of one particular peer p, and the peer's collection $\mathcal{C}_p$ consists of 1,000 documents, then each matching sample document in the ranking represents the expectation of $m = 1{,}000/10 = 100$ matching documents at the peer. This is formally defined as (Shokouhi and Si, 2011; Si and Callan, 2003a):

$$R_p\left(\mathcal{Q}, \mathcal{S}_p, \mathcal{C}_p\right) = \sum_{d \in \mathcal{S}_p} R_d\left(d, \mathcal{Q}\right) \cdot \frac{\left|\mathcal{C}_p\right|}{\left|\mathcal{S}_p\right|}, \tag{5.4}$$

where $\mathcal{Q}$ is a query, $\mathcal{S}_p$ represents the sample documents of peer p in the central sample index, $\mathcal{C}_p$ is the document collection of peer p, d is a document in the sample $\mathcal{S}_p$ and $R_d\left(d, \mathcal{Q}\right)$ is 1 if document d is ranked high enough for query $\mathcal{Q}$ and 0 otherwise[1]. The last part of the formula $\left|\mathcal{C}_p\right| / \left|\mathcal{S}_p\right|$ is often referred to as the *scale factor*, where $\left|\mathcal{C}_p\right|$ is the number of documents at peer p and $\left|\mathcal{S}_p\right|$ is the number of sample documents of peer p in the central sample index.

ReDDE does not consider the entire ranking of documents, but only the top n. There are a number of ways in which n can be chosen. We used a fixed value of $n = 1{,}000$ documents, corresponding with ReDDE.top (Arguello et al., 2009a). With respect to the formula above, this means that $R_d\left(d, \mathcal{Q}\right)$ is 0 if the rank of sample document d is higher than 1,000.

1 There are also variants of ReDDE that substitute the score of the sample document here, these are termed *score-based* variants, we use a *count-based* approach.

In ReDDE, the weight of each document is essentially the same, irrespective of its rank. In contrast, CRCS *does* consider the rank of each document, which means that sample documents that are ranked higher have more influence on the selection process. Additionally, CRCS also normalises the scale factor with respect to the largest peer in the system. It is defined as follows (Shokouhi, 2007; Shokouhi and Si, 2011):

$$R_p\left(\mathcal{Q}, \mathcal{S}_p, \mathcal{C}_p, \mathcal{C}_{pmax}\right) = \sum_{d \in \mathcal{S}_p} \alpha \cdot \exp\left(-\beta \cdot R\left(d, \mathcal{Q}\right)\right) \cdot \frac{\left|\mathcal{C}_p\right|}{\left|\mathcal{C}_{pmax}\right| \cdot \left|\mathcal{S}_p\right|}, \quad (5.5)$$

where $\mathcal{Q}$ is a query, $\mathcal{S}_p$ represents the sample documents of peer p in the central sample index, $\mathcal{C}_p$ is the document collection of p, $\mathcal{C}_{pmax}$ is the largest document collection in the system, d is a document in sample $\mathcal{S}_p$ and $R\left(d, \mathcal{Q}\right)$ is the rank of sample document d in the relevance ranking for query $\mathcal{Q}$. For the constants we used $\alpha = 1.2$ and $\beta = 2.8$, and we consider the top $n = 50$ ranked sample documents. These parameter settings are identical to those used in the original paper that introduced the method (Shokouhi, 2007).

We can adapt ReDDE to weight a reputation score for each peer. We assume the reputation is a value between 0.0 and 1.0 and use the following adjusted version of the original ReDDE formula:

$$R_p\left(\mathcal{Q}, \mathcal{S}_p, \mathcal{C}_p, r_p\right) = \sum_{d \in \mathcal{S}_p} R_d\left(d, \mathcal{Q}\right) \cdot \frac{\left|\mathcal{C}_p\right|}{\left|\mathcal{S}_p\right|} \cdot r_p, \quad (5.6)$$

where $r_p \in [0, 1]$ is the reputation. The rest of the parameters are identical to the original ReDDE Formula 5.4. The intuition behind this formula is that the reputation affects how many relevant documents are expected to be returned. If a particular peer has a reputation of say 0.5, then only half its document are expected to be relevant. In our experiment we derive reputations from two sources: spam scores and pageranks, this is discussed in Subsection 5.3.7.

5.1.2 *Costs*

Table 5.1 shows an overview of the theoretical cost of each approach. Random selection has no costs, no information needs to be stored at, or transferred to, the tracker, apart from the peer identifiers. Storing the size

Table 5.1: Theoretical Costs of each Method

Method	Costs / Peer (Bytes)
Random	0
Size	4
Vocabulary	
- 10 terms	50
- 100 terms	500
- 1000 terms	5,000
Weighted Vocabulary (1000 terms)	9,000
Samples	5,000 – 1,500,000

would only costs as much as storing a single integer. A conservative estimate of four bytes is used here, as this would allow for peers with up to four billion documents: likely enough for contemporary collections. For the vocabulary-based approach, we assume the average term length to be five characters and also assume these to be mostly characters expressible in one byte. Here the storage requirements become larger depending on the number of terms used. For weighted vocabularies, assuming we use 1,000 terms for each peer, the costs are somewhat higher as we store a single precision floating point value for each term as well. Finally, using document samples has a variable costs that depends on the size of the sample documents. If we assume an average document length of 5,000 bytes, and a sample size of 1 to 300 documents, we get an estimate of as much as 1,500,000 bytes per peer. However, this is likely to be an overestimate and the real size is likely closer to the lower number predicted: 5,000 bytes per peer.

Apart from the costs in bytes shown in the table, there is also the additional cost of storing peer identifiers and/or host names. This is something that will likely add an additional overhead of several tens of bytes to each of the methods here. We will revisit this issue later when we present the actual costs as opposed to these theoretical costs.

5.2 DATA SETS

We use the ClueWeb 2009 Category B collection, consisting of fifty million documents, and the Gov2 collection consisting of about twenty five mil-

Table 5.2: Collection Statistics

	ClueWeb09-B	*Gov2*
Raw Size (GB)	1,500	400
Index Size (GB)	435	217
#Documents (K)	50,220	25,205
Avg Doc Length (B)	5,215	5,716
#Peers (K)	2,929	17
#Anchors (K)	404,060	2,973
#Terms (M)	143	69
#Queries	98	149

lion documents (Clarke et al., 2004, 2009). All mark-up is removed from the documents before they are added to the index, this results in some empty documents, and conversely some documentless peers, that we do not further consider.

Statistics are shown in Table 5.2. We use the TREC queries for these collections for evaluation. For ClueWeb there are two queries and for Gov2 there is one query for which there are no relevant search results. These were not included conforming to the official TREC evaluations. We experiment with almost 3 million peers for ClueWeb09-B and nearly 17 thousand for Gov2. To the best of our knowledge there is no previous work that explores resource selection in peer-to-peer networks at the scale of millions of peers. Gov2 has been previously used to create a 25 thousand peer network (Lu, 2007).

There are likely machines that hold the web pages for many hostnames, and conversely there are hostnames that may in reality map to many machines. For reasons of simplicity we ignore these complications and assume that each unique hostname represents one peer in the peer-to-peer system. This approach has been used before to create generic web retrieval (Hawking and Thomas, 2005) and peer-to-peer web retrieval test beds (Klampanos et al., 2005; Lu and Callan, 2003).

ClueWeb09-B and Gov2 are both subsets of the web, but have different characteristics. The peers in Gov2 are very large: each holds nearly 1,500 documents on average, whereas for ClueWeb09-B this is only about 17. Table 5.2 also lists the number of anchors. We define an anchor as a link, made using an HTML anchor tag, between two distinct hosts, these are so

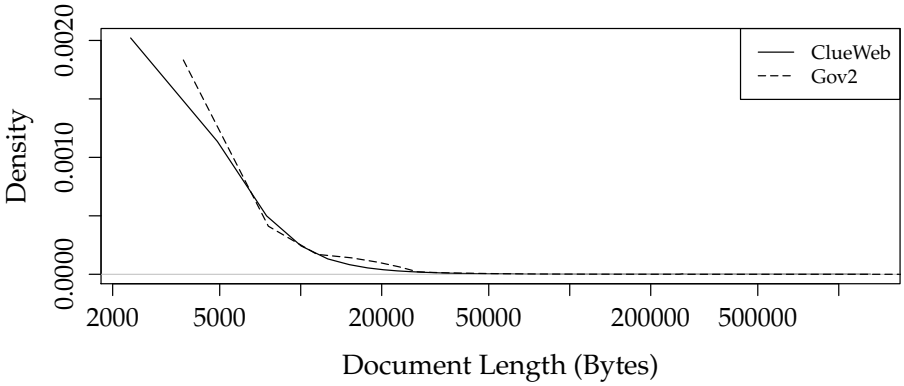


Figure 5.1: Document length distribution of ClueWeb09-B and Gov2 based on pre-processed documents.

Table 5.3: Largest Peers in ClueWeb

Peer	#Docs	#Anchors
en.wikipedia.org	5,996,421	4,197,241
dictionary.reference.com	34,686	223,468
dir.yahoo.com	30,428	76,877
www.aboutus.org	26,818	22,352
acronyms.thefreedictionary.com	25,849	351,828
en.wiktionary.org	22,067	83,211
www.answers.com	21,735	101,433
commons.wikimedia.org	21,450	214,888
www.scribd.com	20,476	31,872
www.iexplore.com	19,535	2,768

Table 5.4: Largest Peers in Gov2

Peer	#Docs	#Anchors
ghr.nlm.nih.gov	717,321	66
nih-library.nih.gov	709,105	46
wcca.wicourts.gov	694,505	11
cdaw.gsfc.nasa.gov	665,987	16
catalog.tempe.gov	650,208	366
www.catalog.kpl.gov	637,313	6
edc.usgs.gov	551,123	362
www.fs.usda.gov	492,416	18
gis.ca.gov	459,329	35
www.csm.ornl.gov	441,201	253

called *external links*. Hence, links within the same host are not considered and only unique anchors counts are shown. Since the number of peers in Gov2 is much lower, the number of interlinks, and thus anchors, is also much lower. Furthermore, the size distribution of these collections is different as well. Figure 5.1 shows the size distribution as a kernel density plot, and Table 5.3 and Table 5.4 list the largest peers and the number of documents they contain, as well as the number of incoming anchors. Even though the average document length of both collections is not that far apart, the distribution is somewhat different as can be seen in the kernel density plot. While ClueWeb09-B has many small documents, those in Gov2 are larger. ClueWeb09-B has a snapshot of the complete English Wikipedia, which results in one peer that dwarfs many of the others. We will revisit this during our experiments. For Gov2 the size distribution is more even, and the peers are larger. The number of anchors for Gov2 is also lower for the largest peers listed, for reasons explained before: there are simply not that many peers in Gov2, and they are larger and more self-contained.

Figure 5.2 and Figure 5.3 show, for each TREC query, the number of peers we would need to contact to achieve complete recall. The purpose of these graphs is to get an impression of the distribution of relevant documents over the peers. For ClueWeb09-B, TREC query number 1 we would need to contact 10 peers to achieve perfect recall. Each segment in the bar indicates how many peers there are that have a number of rele-

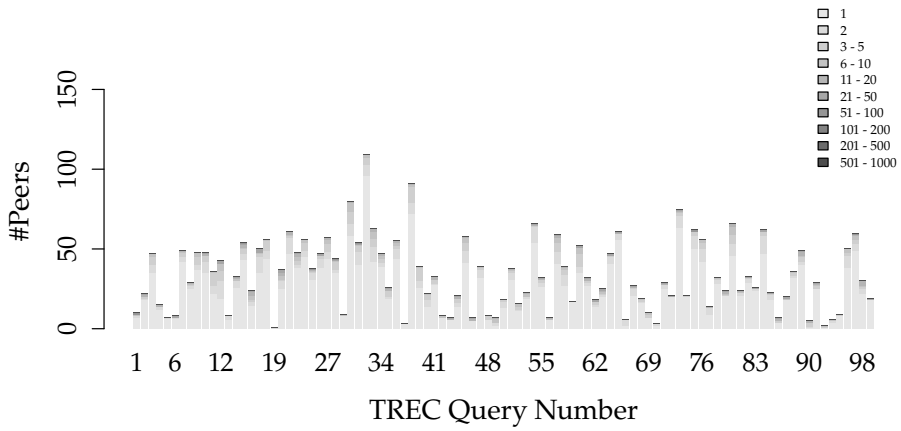


Figure 5.2: Distribution of relevant documents per peer for ClueWeb09-B.

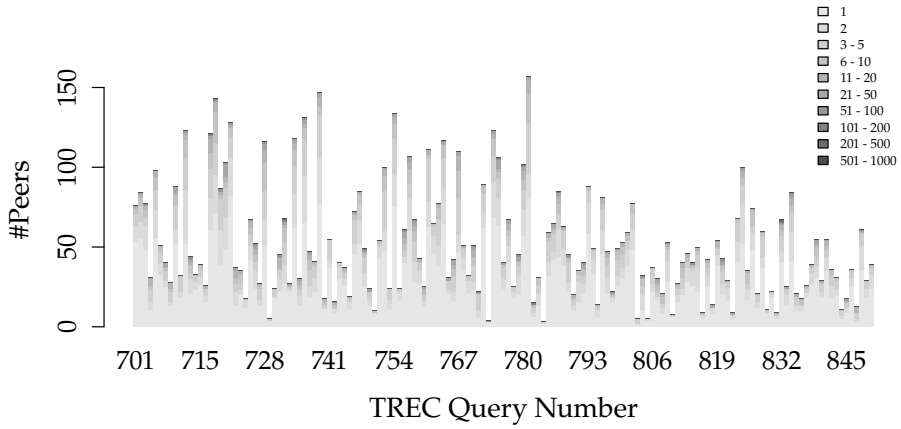


Figure 5.3: Distribution of relevant documents per peer for Gov2.

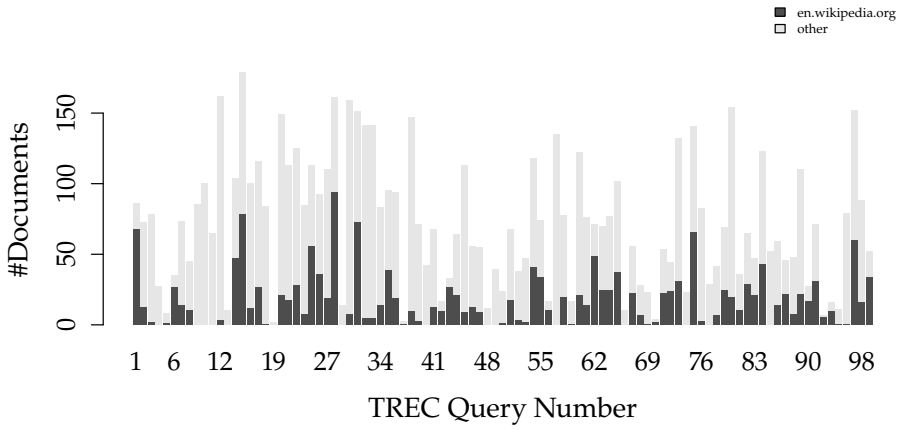


Figure 5.4: Distribution of relevant documents over Wikipedia and other peers for ClueWeb09-B.

vant documents in a specific range. For query 1 there are 7 peers with one relevant document and 3 peers with more than one: 2, 6–10 and 51–10, respectively. What we can conclude from these graphs is that, for both ClueWeb09-B and Gov2, there are many peers that hold a relevant document, but there are also a few peers that hold many relevant documents. The average number of peers to contact for full recall is 52 for Gov2 and 33 for ClueWeb09-B. We will get back to this in the experiment section.

Figure 5.4 shows the number of relevant documents that specifically reside at Wikipedia, and those that reside at other hosts. In contrast with the other graphs, we show the number of documents on the vertical axis. As stated, the Wikipedia peer is many times larger than others, we will see the effect this has on resource selection in the next section.

5.3 EXPERIMENT

In this section we evaluate each peer selection method taking the perspective of the tracker. We use the TREC queries to assess the performance in terms of relevance of each method. In most cases this is based on using a particular resource description at the tracker for peer selection, followed by performing the actual document retrieval at the peers.

For indexing and searching the collections, and derivatives thereof, we used Apache Lucene 3.4. By default we obtain maximally 1,000 search

Table 5.5: Centralised Baselines

Corpus	Part	P@5	P@10	P@30	P@100	MAP
CWB	Full	0.19	0.20	0.16	0.13	0.09
	Wp	0.21	0.21	0.15	0.10	0.06
	No-Wp	0.18	0.19	0.16	0.12	0.08
Gov2	Full	0.29	0.28	0.27	0.24	0.14

results per peer and perform a merge to obtain a final result list with maximally 1,000 results as well. In some experiments, we vary the number of selected peers and in others we keep this fixed. The merge we perform is perfect, meaning: we assume that peers have access to global idf statistics that would enable them to merge search results from multiple sources perfectly. In a real system this could be solved using Kirsch's algorithm (Kirsch, 1997; Lu, 2007). However, we do not further consider the result merging problem in this chapter.

For the experiments conducted in this chapter, we look strictly at relevance. However, we stress that a peer selection algorithm should take query processing overhead and load distribution into account.

5.3.1 *Baselines*

We first need to establish a baseline. We created four indices: one with all ClueWeb Category B documents, one with only the Wikipedia part of ClueWeb, one with everything but the Wikipedia part of ClueWeb and one with all Gov2 documents. Table 5.5 shows results for centralised baseline runs for the 98 TREC ClueWeb queries on the ClueWeb indices and the 149 TREC Gov2 queries on the Gov2 index.

The reason we created a Wikipedia-only index is to measure the effect of selecting Wikipedia as peer. We can see this effect is significant, as using only Wikipedia offers performance competitive with using the whole of ClueWeb Category B. This suggests that simply selecting Wikipedia as peer is a good strategy, since it contains so many relevant documents, which can also be seen in Figure 5.4. However, the results of the everything except Wikipedia run (No-Wp), shows that decent performance can also be obtained without Wikipedia. The fact that the Wikipedia portion of ClueWeb has different characteristics in terms of documents quality

Table 5.6: ClueWeb09-B Random Selection

#Peers	P@5	P@10	P@30	P@100	MAP
1	0.00	0.00	0.00	0.00	0.00
10	<0.01	<0.01	<0.01	<0.01	<0.01
100	<0.01	<0.01	<0.01	<0.01	<0.01
1000	<0.01	<0.01	<0.01	<0.01	<0.01

Table 5.7: Gov2 Random Selection

#Peers	P@5	P@10	P@30	P@100	MAP
1	<0.01	<0.01	<0.01	<0.01	<0.01
10	0.01	0.01	<0.01	<0.01	<0.01
100	0.05	0.04	0.02	0.01	<0.01
1000	0.22	0.18	0.12	0.06	0.02

has been observed before (Bendersky et al., 2011; He et al., 2009).

Experiments with the Gov2 collections partitioned by domain name have been performed before by Thomas and Hawking (2007). Earlier work on Gov2 shows that partitioning by domain name is particularly challenging for result merging, a problem we do not consider in this chapter (Fallen and Newby, 2006).

5.3.2 *Random*

Since there are thousands of peers in Gov2 and millions of peers in ClueWeb09-B, the odds of randomly selecting a peer with relevant documents are very low. In fact, for each Gov2 query there are on average 181, standard deviation ± 149, relevant documents distributed over 52 ± 35 peers. For each ClueWeb query there are on average 72 ± 46, relevant documents distributed over 33 ± 22 peers. Hence, the average probability of randomly selecting one peer with at least one relevant document is about 0.3 percent for Gov2 and about 0.001 percent for ClueWeb09-B. That probability increases as we randomly select more peers without replacement.

We verified the effect of random selection on both ClueWeb09-B and Gov2 by selecting 1, 10, 100 and 1000 peers randomly. These experiments

were repeated ten times and the results were averaged. For ClueWeb09-B, the final result was very close to zero for all metrics and for all selection sizes as shown in Table 5.6. The higher probability of selecting a good peer at random in Gov2 was confirmed by our experiments. In Table 5.7 we see that the P@5 is as high as 0.22 for random selection of 1,000 peers respectively.

For a large peer-to-peer network, like the one based on ClueWeb09-B, we can not perform random selection and need more information about peers in order to make an informed routing decision. A similar argument holds for Gov2, as selecting 1,000 peers is very taxing on the resources of a peer-to-peer network. However, for very small peer-to-peer information retrieval networks, with limited resources, a random selection strategy may not be the worst approach, particularly if relevant documents are uniformly distributed.

5.3.3 *Size-based*

Table 5.8 shows the results for size based selection on ClueWeb09-B. The largest server is Wikipedia: *en.wikipedia.org*, and in fact the results for selecting 1 peer are identical to the Wikipedia baseline run we saw before. The WP column shows the percentage of the returned results provided by Wikipedia. In ClueWeb, some of the Wikipedia pages are outside the official Wikipedia part of the corpus. Hence, there is a 1 percent error margin on the numbers reported in the WP columns in all tables. This is the reason why for 1 peer we see 99 percent instead of 100 percent Wikipedia results.

If we increase the number of selected peers, we see a dip around 100 peers. Further investigation reveals that performance actually increases quickly again after selecting 200 and 500 peers respectively, with 500 giving a top P@5 of 0.19, but a lower P@15 of 0.16 compared to the results for 1,000 peers. This suggests that there are a few large peers (around 100) that, while large, have fewer relevant documents, which negatively affects the precision. These observations are similar to those by Witschel (2008).

For ClueWeb09-B we also conducted this same experiment based on pageranks instead of size. Since large servers can accumulate a lot of pagerank, the results of this experiment were nearly identical to those shown in Table 5.8, the only difference being a P@5 of 0.17 instead of 0.18

Table 5.8: ClueWeb09-B Size-based Selection

#Peers	P@5	P@10	P@30	P@100	MAP	WP
1	0.21	0.21	0.15	0.10	0.05	99%
10	0.18	0.18	0.14	0.09	0.05	91%
100	0.16	0.16	0.12	0.09	0.04	63%
1000	0.18	0.18	0.12	0.09	0.04	34%

Table 5.9: Gov2 Size-based Selection

#Peers	P@5	P@10	P@30	P@100	MAP
1	0.00	0.00	0.00	0.00	0.00
10	0.05	0.04	0.02	0.01	<0.01
100	0.23	0.23	0.19	0.13	0.04
1000	0.27	0.27	0.26	0.22	0.10

when selecting the 1,000 peers with the highest pagerank.

Table 5.9 shows the results for size base selection on Gov2. We can see that selecting the largest peer is ineffective, as it has no relevant results for any of the queries. Selecting more peers gives increasingly better performance, suggesting that the smaller servers still contribute relevant documents. Gov2's peer size distribution is more uniform than ClueWeb09-B's, particularly at the low-end, and since there are fewer peers, we are selecting a much larger percentage of them: 1,000 peers is 5.88 percent of Gov2, whilst it is only 0.03 percent of ClueWeb09-B.

We can conclude that size does have influence on performance. This observation is in line with resource selection algorithms that have a size bias, like ReDDE (Si and Callan, 2003a). However, as is clear from the results for Gov2, just picking the largest peer, or a subset of large peers, does not always work very well. Further analysis reveals that the top ten servers in Gov2 hold only 0.73 percent of the relevant documents. This drastically bounds the maximum attainable performance. The results for ClueWeb09-B are distorted: 24.66 percent of the relevant documents resides at the largest peer: en.wikipedia.org. Using the top ten servers only marginally increases this to 25.21 percent. Hence, the fairly good performance for ClueWeb is really due to Wikipedia.

Statically selecting the largest peers in the networks is not a good

approach for two other reasons besides the results of the experiments. Firstly, it would not work well for long-tail queries. Secondly, it would miss highly relevant search results that reside at small peers. Nevertheless, knowing the size of peers does improve things with respect to random selection. Hence, knowing the size is better than nothing.

5.3.4 *Vocabulary-based*

In this section we base the selection on the vocabulary of each peer. We performed these experiments in two ways. Firstly, by considering the full content of each document. Secondly, by considering only the vocabulary of the document titles. In both cases, but especially when using only titles, some small servers may have fewer than the top n terms requested. Hence, their resource descriptions would be *smaller* than those of larger servers. By default, smaller matching documents get a boost in Lucene, to cope with the fact that longer documents are more likely to match any query[2]. However, since these are large document representations, this is the opposite of what we want. Hence, we turned off this document length normalisation when scoring a query against the resource descriptions. Note that, even with length normalisation turned off, Lucene does use the inverse document frequencies when scoring.

In contrast with the previous sections, we fixed the number of peers to 10 in this setting. This way we can see the influence of changing the number of top terms n. We wanted to find out if a larger or smaller top n matters. Intuitively, a higher n value would give more information for selection. Hence, using a larger n is expected to yield better performance. However, a larger n also has higher costs in terms of bandwidth consumption and storage.

Table 5.10 shows the results for using the top n content and title vocabularies for ClueWeb09-B. The fraction of search results provided by Wikipedia is shown in the last column. Note that Wikipedia was never selected for any of the queries for the runs with a fixed n. For the titles only the count-based results are shown, while for full content the results of all methods are shown. As can be seen, the local.tfidf approach consistently performs worse, and at best equal to using global tf.idf's. It seems to be always best to base the top n term vocabularies on simple counts. For

2 For details see `http://lucene.apache.org/core/old_versioned_docs/versions/3_4_0/api/all/org/apache/lucene/search/Similarity.html` (Retrieved June 25th 2012)

Table 5.10: ClueWeb09-B Top n Vocabulary-based Selection

Method	#Terms	P@5	P@10	P@30	P@100	MAP	WP
C-C	10	0.15	0.13	0.08	0.04	0.03	0%
	100	0.09	0.09	0.04	0.02	0.02	0%
	1000	0.03	0.03	0.01	<0.01	<0.01	0%
	1%	0.17	0.15	0.10	0.05	0.04	10%
C-L	10	0.07	0.04	0.02	<0.01	<0.01	0%
	100	0.04	0.02	0.01	<0.01	<0.01	0%
	1000	0.02	0.01	0.01	<0.01	<0.01	0%
	1%	0.09	0.07	0.03	0.02	0.01	32%
C-G	10	0.10	0.08	0.05	0.02	0.02	0%
	100	0.08	0.06	0.03	0.01	0.01	0%
	1000	0.03	0.02	0.01	0.01	0.01	0%
	1%	0.15	0.12	0.08	0.04	0.04	25%
T-C	10	0.11	0.09	0.06	0.03	0.02	0%
	100	0.12	0.10	0.06	0.03	0.03	0%
	1000	0.09	0.07	0.04	0.02	0.02	0%
	1%	0.16	0.15	0.11	0.08	0.05	46%

Selection is based on (C)ontents using term (C)ounts, (L)ocal tf.idf or (G)lobal tf.idf. For (T)itles only term (C)ounts are shown. At most 10 peers are selected.

titles, not all results are shown here, the differences in ClueWeb09-B are small, in fact using global tf.idf gives about equal performance to using counts when using 100 and 1,000 terms respectively. When using 10 terms the performance is about two thirds of the count-based approach. Briefly: using simple counts is the most effective approach for ClueWeb09-B and also the cheapest. Using more evidence, that is: more terms, does not yield better performance in terms of precision. One could conclude that using only the top 10 terms is optimal within the explored settings for content, and the top 100 terms for titles. Using 1,000 terms for content is in fact worse than using 1,000 title terms. In general: using just titles works well, consistent with the observations of Lu and Callan (2003).

A possible cause of the poor performance when using more terms for content is that small peers become overrepresented and large peers underrepresented. To investigate this issue further we also performed a run that uses 1 percent of all the terms for each peer. The results of this are consistently better than any of the fixed cut-offs for ClueWeb and the reduction in precision is much less than when using a fixed number of terms. This suggests that, indeed, using a fixed number of terms may not be the best approach if collection sizes are heterogeneous, as this may overrepresent the small servers. Having the larger peers represented more strongly helps performance, which pleads for using a size-relative number of terms instead. Although for ClueWeb these results are distorted by Wikipedia, we can see that when using counts only 10 percent of the results are from Wikipedia, while counts give better results than any of the other methods explored.

Table 5.11 shows the results for Gov2 using all methods for content and using counts for the titles. The count-based content selection performs about 0.03 worse for P@5, but that difference becomes smaller for higher precision levels. For title selection the situation is reversed: count-based performs better than the global tf.idf, not shown here, by about the same margin. There are two things different about the results for Gov2. Firstly, using more terms gives better performance, this is true for all variants tested. Secondly, using only titles outperforms content-based summaries by a large margin at higher precision levels. This may be the result of the organisation of the Gov2 source data. The average title length for Gov2 is 16 ± 24 characters, which is in fact much shorter than for ClueWeb09-B that has titles of 42 ± 45 characters. Hence, we have to assume that Gov2's titles are more descriptive in some other, perhaps semantic, way.

Table 5.11: Gov2 Top *n* Vocabulary-based Selection

Method	#Terms	P@5	P@10	P@30	P@100	MAP
C-C	10	0.11	0.10	0.07	0.04	0.01
	100	0.16	0.13	0.09	0.05	0.02
	1000	0.25	0.22	0.15	0.08	0.03
	1%	0.23	0.22	0.20	0.14	0.05
C-L	10	0.10	0.07	0.04	0.02	0.01
	100	0.15	0.12	0.07	0.03	0.01
	1000	0.22	0.18	0.10	0.04	0.02
	1%	0.24	0.23	0.18	0.12	0.04
C-G	10	0.13	0.11	0.08	0.04	0.01
	100	0.19	0.16	0.11	0.06	0.02
	1000	0.27	0.23	0.16	0.07	0.03
	1%	0.27	0.25	0.21	0.14	0.05
T-C	10	0.13	0.11	0.08	0.04	0.01
	100	0.19	0.17	0.12	0.07	0.02
	1000	0.28	0.25	0.20	0.12	0.05
	1%	0.15	0.13	0.10	0.07	0.02

Based on Contents (C) using term counts (C), local tf.idf (L) and global tf.idf (G). For titles (T) only term counts (C) are shown. Selecting at most 10 peers.

We also applied the 1 percent term selection on Gov2, the results here are less conclusive. For titles, using a fixed percentage is consistently worse than using 1,000 terms and mostly worse when using 100 title terms. If we look at the content-based results, we see that for counts using a fixed selection of 1,000 terms gives equal or better performance at low precision levels, only for global and local tf.idf we see improved results. However, we do see, like for ClueWeb, that the performance is more stable at higher precision levels, equalling or outperforming all other fixed term selections from P@10 and higher, and offering better performance in terms of MAP. Hence, this approach may be better when a high number of relevant results is desired.

We have already seen that considering the size of the document collection of each peer can be very effective. Whilst this approach has been explored in previous work when using document samples, it has not been used in the same way for using vocabularies. Hence, we introduce a weighting function:

$$\text{score}_{weighted}\left(s_p, \mathcal{C}_p\right) = s_p \cdot \left(1 + \log\left|\mathcal{C}_p\right|\right), \tag{5.7}$$

where s_p is the unweighted score of the vocabulary document of peer p and $\left|\mathcal{C}_p\right|$ gives the number of documents at peer p. Effectively, we reweight the score of each vocabulary by giving peers with more documents the ability to compensate for the fixed size of the vocabulary document. Reweighting applies only to peers with a non-zero score.

Table 5.12 shows the results for ClueWeb. These can be compared with Table 5.10, as the only difference is the weighting formula introduced. We see that similar patterns still hold as in the original results, the difference being that using 10 terms is now also the best performing approach when using titles and the drop in performance when using more terms for contents is less pronounced. The performance when using only titles is also much closer to using a content vocabulary. All results are consistently better than when not using size weighting. Using global tf.idfs performs the best for low precision levels. However, using just counts seem better for higher precision levels. Interestingly, Wikipedia is hardly ever selected for the fixed-size vocabularies. It does not benefit from the size bias, likely due to it being underrepresented. Contrast this with the 1 percent runs and we see a higher percentage of Wikipedia results compared to the non size-weighted approach.

Table 5.12: ClueWeb09-B Top *n* Size-weighted Vocabulary-based Selection

Method	#Terms	P@5	P@10	P@30	P@100	MAP	WP
C-C	10	0.17	0.15	0.11	0.06	0.04	0%
	100	0.14	0.13	0.10	0.05	0.04	0%
	1000	0.10	0.08	0.04	0.02	0.02	1%
	1%	0.15	0.15	0.12	0.09	0.06	51%
C-L	10	0.10	0.06	0.03	<0.01	<0.01	0%
	100	0.11	0.07	0.03	0.01	0.01	0%
	1000	0.11	0.08	0.04	0.01	0.01	0%
	1%	0.16	0.16	0.13	0.08	0.05	71%
C-G	10	0.17	0.15	0.09	0.04	0.03	0%
	100	0.19	0.17	0.10	0.04	0.04	0%
	1000	0.15	0.11	0.05	0.02	0.02	0%
	1%	0.21	0.19	0.14	0.10	0.07	62%
T-C	10	0.16	0.15	0.11	0.05	0.04	0%
	100	0.16	0.14	0.09	0.05	0.03	0%
	1000	0.11	0.09	0.05	0.03	0.02	2%
	1%	0.17	0.16	0.11	0.08	0.05	49%

Based on (C)ontents or (T)itles using term (C)ounts, (L)ocal tf.idf or (G)lobal tf.idf. Selecting at most 10 peers.

Table 5.13: Gov2 Top *n* Size-weighted Vocabulary-based Selection

Method	#Terms	P@5	P@10	P@30	P@100	MAP
C-C	10	0.16	0.14	0.10	0.06	0.02
	100	0.20	0.18	0.13	0.09	0.03
	1000	0.28	0.25	0.20	0.12	0.05
	1%	0.23	0.23	0.20	0.13	0.04
C-L	10	0.11	0.08	0.05	0.02	<0.01
	100	0.23	0.19	0.12	0.05	0.02
	1000	0.27	0.22	0.16	0.08	0.03
	1%	0.26	0.25	0.20	0.13	0.05
C-G	10	0.16	0.13	0.09	0.05	0.01
	100	0.25	0.20	0.15	0.08	0.03
	1000	0.34	0.29	0.21	0.12	0.05
	1%	0.28	0.27	0.23	0.15	0.06
T-C	10	0.16	0.14	0.10	0.05	0.01
	100	0.21	0.18	0.14	0.09	0.03
	1000	0.30	0.27	0.21	0.14	0.05
	1%	0.15	0.14	0.11	0.07	0.02

Based on (C)ontents or (T)itles using (C)ounts, (L)ocal tf.idf or (G)lobal tf.idf. Selecting at most 10 peers.

Table 5.13 shows the size-weighted results for Gov2, that show a similar pattern of improvement as ClueWeb09-B. However, the differences here are larger and, in terms of early precision, also improve over the baseline centralised search when using global tf.idfs. A difference with the results in Table 5.11 is that the content-based approach really does perform better here by a margin.

Overall using just a title-based vocabulary already offers impressive performance at a low cost, but using a content-based vocabulary offers further performance improvements. Weighting by including the size of the peers offers large performance gains.

In brief: using counts is the best and least expensive approach. When using content-based vocabularies using fewer terms is in fact preferable, while for titles using a bit more seems better. Weighting in the size of the collections is a good idea and offers a competitive boost to performance.

We can conclude in general that for one corpus, Gov2, using more terms helps performance. However, this is not true for ClueWeb, in which using more terms actually degrades performance. Since we see opposite trends, this makes a strong case for the conclusion that the number of terms selected from each peer is highly corpus dependent, and that more is not always better.

5.3.5 *Term-weighted Vocabulary-based*

In this section we apply weighting to the individual terms in the vocabularies. We use the top 1,000 terms based on counts, as in some of the experiments in the previous section, but instead of throwing the frequency information away, we use it for scoring. What we would like to find out is if this makes a difference. The scoring in this experiment was done outside of Lucene with a custom implementation.

Table 5.14 shows the results for ClueWeb09-B. These are better than both the unweighted and size-weighted results of the previous section for 1,000 terms, which shows that including frequency information helps performance. It compensates for the overrepresentation of small servers when using a fixed number of terms. We experimented with two lambda values: 0.8 and 0.1. The higher value gives better results, also in the results for Gov2 in Table 5.15. The performance for content here is nearly identical to that of unweighted vocabularies, but not for titles. However, the general results are not as good as the size-weighted results.

Table 5.14: ClueWeb09-B Top $n = 1000$ Term-weighted Vocabulary-based Selection

Method	P@5	P@10	P@30	P@100	MAP	WP
C ($\lambda = 0.8$)	0.18	0.15	0.10	0.05	0.03	0%
T ($\lambda = 0.8$)	0.16	0.13	0.08	0.04	0.03	0%
C ($\lambda = 0.1$)	0.17	0.15	0.10	0.05	0.03	0%
T ($\lambda = 0.1$)	0.14	0.12	0.07	0.03	0.03	0%

Based on (C)ontent and (T)itles. Selecting at most 10 peers.

Table 5.15: Gov2 Top $n = 1000$ Term-weighted Vocabulary-based Selection

Method	P@5	P@10	P@30	P@100	MAP
C ($\lambda = 0.8$)	0.25	0.22	0.16	0.09	0.03
T ($\lambda = 0.8$)	0.24	0.22	0.17	0.11	0.04
C ($\lambda = 0.1$)	0.19	0.17	0.13	0.07	0.02
T ($\lambda = 0.1$)	0.22	0.19	0.13	0.08	0.03

Based on (C)ontent and (T)itles. Selecting at most 10 peers.

As we have suggested in the previous section, using a fixed number of terms can overrepresent small servers. In this section we have shown that this can be attributed to stripping of frequency information from the representations, at least for ClueWeb09-B. This is consistent with other experiments, like those of Lu and Callan (2003), and shows that language models can be pruned significantly, while still offering competitive performance. However, for Gov2, the results are less conclusive. Including frequency information does not matter much for content, and can even negatively affect performance for titles. Future experiments could look at the effect of selecting different amounts of terms, either fixed or percentage-based.

5.3.6 *Query Log based*

In this section we discuss the results of the experiments that use a query log as representation. Each peer is represented by a history of its queries. Table 5.16 shows the results for ClueWeb09-B and Table 5.17 shows the results for Gov2. We tested these collections both unweighted and size-

Table 5.16: ClueWeb09-B Top n Query Log (U)nweighted and Size-(W)eighted Selection

Method	#Terms	P@5	P@10	P@30	P@100	MAP	WP
U	10	0.16	0.14	0.09	0.04	0.03	0%
	100	0.15	0.12	0.09	0.05	0.03	0%
	1000	0.16	0.13	0.10	0.05	0.04	0%
W	10	0.15	0.13	0.10	0.05	0.04	0%
	100	0.16	0.13	0.10	0.06	0.04	0%
	1000	0.17	0.14	0.10	0.06	0.03	0%

Selecting at most 10 peers.

Table 5.17: Gov2 Top n Query Log (U)nweighted and Size-(W)eighted Selection

Method	#Terms	P@5	P@10	P@30	P@100	MAP
U	10	0.13	0.11	0.08	0.04	0.01
	100	0.22	0.18	0.14	0.07	0.02
	1000	0.26	0.24	0.18	0.11	0.04
W	10	0.17	0.15	0.11	0.06	0.02
	100	0.26	0.21	0.16	0.09	0.03
	1000	0.27	0.24	0.19	0.12	0.04

Selecting at most 10 peers.

weighted, using the algorithm discussed in Section 5.3.4. For ClueWeb09-B the influence of the number of queries used is quite minimal. For lower precision levels using fewer queries seems desirable, whereas higher precision levels can be boosted slightly by using more queries. Size-weighting makes some difference, but the improvement is quite minimal, and there is even some deterioration when using fewer queries. We never consult Wikipedia, showing that it is not required for decent performance. The results for Gov2 show something different though: using more queries leads to larger performance gains, and also doubles the MAP. The unweighted results are somewhat similar to those using peer vocabularies with global tf.idf in Table 5.11. Size-weighting gives a bigger boost for Gov2 compared with ClueWeb09-B. Although, this diminishes as more queries are used.

Using unweighted queries gives performance that is on par with using peer vocabularies, and does not show the diminishing performance for ClueWeb09-B when using just terms. However, the gain from size weighting is less prominent here than for peer vocabularies.

5.3.7 *Document Sample based*

In this section we evaluate the effectiveness of resource selection algorithms that use a central sample index: ReDDE (Si and Callan, 2003a) and CRCS (Shokouhi, 2007). When we create an index like this, we have to choose how many documents we sample from each server. Since the collection size in a peer-to-peer network can be heterogeneous, we use a percentage of the number of documents in the index instead of a fixed number. We aim to represent servers with about 300 documents, similar to what is often randomly sampled by query-based sampling approaches (Callan et al., 1999). We apply the following rule: we use a random selection of 1 percent of the document collection at each peer as sample, with a minimum of 1 document and a maximum of 3,000. This gives us:

$$\text{samplesize}\left(\mathcal{C}_p\right) = \min\left(\max\left(1, \left|\mathcal{C}_p\right| \cdot 0.01\right), 3000\right), \tag{5.8}$$

where $\mathcal{C}_p$ is a peer's document collection. In the formulas in Section 5.1 $\text{samplesize}\left(\mathcal{C}_p\right)$ is referred to as $\left|\mathcal{S}_p\right|$.

Wikipedia in ClueWeb09-B is an unreasonably large outlier: it consists of 6 million documents, whereas the next largest server consists of only about 30,000 or so. Hence, Wikipedia is capped to 3,000 documents. For

Table 5.18: ClueWeb09-B (R)eDDE and (C)RCS Selection

Method	#Peers	P@5	P@10	P@30	P@100	MAP	WP
R-C	1	0.11	0.11	0.07	0.05	0.03	44%
	3	0.16	0.15	0.09	0.05	0.03	25%
	5	0.18	0.16	0.10	0.05	0.03	22%
	10	0.17	0.16	0.09	0.06	0.04	19%
R-T	1	0.08	0.08	0.05	0.02	0.02	39%
	3	0.14	0.12	0.07	0.03	0.02	24%
	5	0.13	0.12	0.07	0.04	0.02	21%
	10	0.13	0.12	0.07	0.04	0.03	18%
C-C	1	0.05	0.04	0.02	0.01	0.01	0%
	3	0.09	0.07	0.05	0.02	0.02	0%
	5	0.11	0.11	0.07	0.03	0.02	0%
	10	0.13	0.13	0.09	0.04	0.03	0%
C-T	1	0.05	0.03	0.02	0.01	0.01	0%
	3	0.09	0.07	0.04	0.02	0.01	0%
	5	0.10	0.07	0.04	0.02	0.01	0%
	10	0.11	0.08	0.05	0.02	0.02	0%

Based on Full (C)ontent or (T)itles.

Gov2 we used the same settings, here the largest peer has about 700,000 documents, and in fact only the top 19 servers have sizes above 300,000. Hence, only these servers are slightly underrepresented.

Table 5.18 shows the results of using ReDDE and CRCS on ClueWeb09-B. We select from 1, 3, 5 and 10 peers: commonly used values. The selection of 10 peers can be compared against previous result tables. We see CRCS consistently performs poorer than ReDDE and, again, just using titles gives reasonable performance. Comparing ReDDE's selection of 10 peers with selecting the 10 largest peers in Section 5.3.3, we see that we might as well statically select the 10 largest peers, as this gives better performance. While using just vocabularies does not work as well as ReDDE, using size-weighted vocabularies gives similar performance at a lower cost. Table 5.19 shows results for Gov2. The situation is reversed: CRCS outperforms ReDDE with few exceptions, results are good compared with the baseline, also in terms of precision at higher levels.

Table 5.19: Gov2 (R)eDDE and (C)RCS Selection

Method	#Peers	P@5	P@10	P@30	P@100	MAP
R-C	1	0.06	0.05	0.04	0.02	0.01
	3	0.13	0.12	0.09	0.05	0.02
	5	0.17	0.16	0.12	0.08	0.02
	10	0.24	0.23	0.18	0.12	0.04
R-T	1	0.06	0.05	0.03	0.01	<0.01
	3	0.12	0.10	0.07	0.04	0.01
	5	0.14	0.12	0.10	0.06	0.02
	10	0.18	0.16	0.13	0.08	0.02
C-C	1	0.20	0.17	0.12	0.07	0.03
	3	0.28	0.24	0.18	0.11	0.04
	5	0.26	0.25	0.20	0.13	0.05
	10	0.27	0.27	0.22	0.15	0.06
C-T	1	0.07	0.07	0.05	0.03	0.01
	3	0.13	0.12	0.09	0.05	0.01
	5	0.17	0.14	0.11	0.06	0.02
	10	0.19	0.16	0.12	0.08	0.02

Based on Full (C)ontent or (T)itles.

Finally, let us look at ReDDE with the reputation adjustment. We use two different sources to estimate a reputation for each peer in the network. Firstly, the average content spam scores of all documents of the peer. Secondly, the average pagerank of all documents of the peer. We scale these to a number between 0.0 and 1.0 and apply the ReDDE formula with the reputation adjustment. Table 5.20 shows the results.

We also experimented with using the median page rank and spam score of each server, as well as the page rank or spam score of the homepage or index page of each peer. Unfortunately, none of these alternatives showed consistently better results compared to just using the averages.

Perhaps we should look at other sources of information for determining the reputation of peers. For example, in a dynamic system this can be based on the behaviour of peers observed over time, see Section 3.4. We experiment with an alternative simple reputation scheme that affects peer selection in Section 6.3.

Table 5.20: ClueWeb ReDDE Reputation Selection using (S)pam Scores or (P)ageranks

Method	#Peers	P@5	P@10	P@30	P@100	MAP	WP
S-C	1	0.11	0.11	0.07	0.05	0.03	44%
	3	0.15	0.14	0.09	0.05	0.03	26%
	5	0.16	0.15	0.08	0.05	0.03	23%
	10	0.16	0.15	0.09	0.06	0.04	20%
S-T	1	0.08	0.08	0.05	0.02	0.02	41%
	3	0.13	0.11	0.06	0.03	0.02	24%
	5	0.15	0.12	0.08	0.04	0.03	21%
	10	0.15	0.12	0.08	0.04	0.03	18%
P-C	1	0.11	0.11	0.07	0.05	0.03	44%
	3	0.16	0.15	0.09	0.05	0.03	26%
	5	0.17	0.15	0.10	0.05	0.03	22%
	10	0.16	0.15	0.09	0.06	0.04	20%
P-T	1	0.08	0.08	0.05	0.02	0.02	37%
	3	0.13	0.12	0.07	0.03	0.02	24%
	5	0.16	0.13	0.07	0.04	0.02	21%
	10	0.14	0.12	0.07	0.04	0.03	18%

Based on Full (C)ontent or (T)itles.

Table 5.21: Approximate Storage Costs of each Approach

	Complete (megabytes)		*Per Peer (bytes)*	
	CW09-B	Gov2	CW09-B	Gov2
Random	0	0	0	0
Size	82	1/2	29	31
Vocabulary (C)				
- 10 terms/peer	701	4	251	251
- 100 terms/peer	4,697	26	1,680	1,593
- 1000 terms/peer	24,576	213	8,789	12,973
- 1%	679	51	243	3,085
Vocabulary (T)				
- 10 terms/peer	520	4	186	212
- 100 terms/peer	942	13	337	772
- 1000 terms/peer	1,292	37	462	2,282
- 1%	15	1	5	54
Term-Weighted Vocab. (C)				
- 1000 terms/peer	35,761	278	12,789	16,973
Term-Weighted Vocab. (T)				
- 1000 terms/peer	1,880	49	672	2,986
Query Log				
- 10/peer	1,153	8	412	481
- 100/peer	3,396	25	1,215	1,504
- 1000/peer	6,503	54	2,326	3,314
Samples (C)*	12,481	1,233	4,464	75,230
Samples (T)*	108	4	39	214

* = counts exclude index overhead.

5.4 DISCUSSION

In Table 5.21 we show an overview of the actual cost of each method in terms of storage space required at the tracker, that can be compared against the estimates in Table 5.1 in Section 5.1.2. This includes the overhead needed for storing the index, the host names of the peers, et cetera. It gives a rough indication of the required storage space and how much bandwidth would be needed to transfer peer representations. However, the bandwidth required is likely lower than the estimates shown here, as this would not include the overhead incurred by the indices. These

indices also require processing power to construct and maintain.

Since ClueWeb09-B has many more peers, the costs are higher than for Gov2. However, the last two columns show the storage cost normalised per peer, and thus provide a basis for comparison. For content vocabularies only the count-based tracker index sizes are shown, since those of local tf.idf and global tf.idf are comparable in size. The storage requirements for samples are based on the random document sample used during the experiments. Although random document samples can be different in size, the numbers reported here still give a rough impression of the required storage space.

The least expensive option is random selection, but it also performs poorly, particularly for ClueWeb. A simple size-based approach is the first option that actually needs some storage space, and also the least amount of all other approaches discussed. It requires about 30 bytes per peer, for storing the peer's hostname and size. The downside is that this approach depends heavily on several large peers, using such a strategy for selection exclusively would not be good for load balancing and would also adversely affect the diversity of search results. While better than random selection, it is not a feasible approach to use exclusively for a large-scale real-world peer-to-peer information retrieval system.

Moving on, we see the vocabulary-based approaches get more expensive as more terms are selected. However, this levels off more quickly for ClueWeb09-B than for Gov2. The reason is that ClueWeb09-B contains many small servers that do not consist of that many terms, a similar argument holds true for the titles, that are on average shorter in Gov2, but with a more uniform length distribution. Combining the vocabulary-based approaches with size-weighting gives good performance and the storage requirements for these are modest compared to storing document samples. Selecting a percentage of the vocabulary is not very expensive, in terms of space, compared with some of the higher fixed amounts.

In terms of required storage, anchors are more expensive to store than individual terms. However, using top $n = 1{,}000$ term-weighted vocabularies is the most expensive option in terms of storage space for ClueWeb09-B, requiring almost 13 kilobytes per peer. For Gov2 the only more expensive option is storing document samples. Using term-weighted vocabularies is a necessity when using a fixed amount of terms with ClueWeb09-B. However, for Gov2 it may be better to rely on plain or size-weighted vocabularies instead.

5.5 CONCLUSION

We have discussed and presented a range of methods for representing peers in a peer-to-peer information retrieval network. With respect to the centralised baselines, the methods particularly negatively affect the precision at higher levels (P@30, P@100). However, this may not be a problem when one is interested only in the top results. The size of peers plays a role in peer selection. Indeed, when no other information is available, using just the size is a legitimate fallback option with a low cost compared to the alternative when there is no information: random selection. Document sampling-based approaches, like ReDDE and CRCS, give good performance. However, they are expensive in terms of transmission and storage costs. Using simple content-based top n term vocabularies, count-based for ClueWeb09-B and based on the global tf.idf score for Gov2, offers fair precision, but not as good as the document sample-based approaches. However, adapting the size-weighting, central to the sampling approaches, offers competitive performance at a lower cost. Reweighting by reputation is not effective with the reputation models we used. Using top n term-weighted vocabularies, that include term frequency information, seems a necessity for ClueWeb09-B, but not for Gov2. Finally, using query logs offers decent performance, but has no advantage over using just vocabularies.

It is difficult to give a definitive recommendation based on the results presented, as we have seen that results also differ depending on the corpus used. However, for a peer-to-peer information retrieval network it seems best to let each peer transmit its size, and a top n vocabulary preferably based on document content, but alternatively based on just the document titles, including frequency information if possible. Extending this conclusion to non-cooperative peer-to-peer environments, this implies that estimating the size of a peer and performing query-based sampling using only document titles, should work well. Although using larger snippets is explored in Section 4.4, using just document titles needs to be further investigated, as does size estimation.

6

CACHING SEARCH RESULTS

'The content people have no clue. The cost of bandwidth is going down to nothing and hard drives are getting so big, and they're so cheap. The content distribution industry is going to evaporate.'

Bram Cohen (in 2005)

Quickly providing search results for queries is an important feature of any large-scale web search engine. In this chapter we investigate distributed demand-driven search result caching as a means for keeping latency low in a large peer-to-peer web search network, particularly for obtaining results for popular queries.

As we have learned, in peer-to-peer information retrieval a network of peers provide a search service collaboratively. The term peer refers to the fact that in a peer-to-peer system all peers are considered equal and can both supply and consume resources. Each additional peer adds extra processing capacity and bandwidth in contrast with typical client/server search systems where each additional client puts extra strain on the server. When a peer-to-peer network has good load balancing properties it can scale up to handle millions of peers simultaneously. However,

This chapter is based on Tigelaar et al. (2011): *Search Result Caching in Peer-to-Peer Information Retrieval Networks*, that appeared in *Multidisciplinary Information Retrieval*, Lecture Notes in Computer Science 6653, ©Springer, 2011.

performance is strongly affected by how well it can deal with the continuous rapid joining and departing of peers called *churn*.

In this chapter we explore search result caching as a load balancing strategy. We assume that for each query there is a peer that can provide a set of original search results. If this query is posed often, that peer would cripple under the demand for providing this set of results over and over again. Hence, we propose that each peer that obtains search results for a particular query *caches* those results. The effect is that search results for popular queries can be obtained from many peers: *high availability*, and the load on the peer that provided the original results is reduced: *load balancing*.

We define the following research questions:

1. What fraction of queries can be potentially answered from caches?
2. How can the cache hit distribution be characterised?
3. What is the distribution of cached result sets given an unbounded cache?
4. What is the effect of bounding the cache: how does the bound and cache policy affect performance?
5. What optimisations can be applied to make caching more effective?
6. How does churn affect caching?
7. How can free-riding peers, that do not cache, be detected and penalised?

Most research in peer-to-peer information retrieval focuses on simulating networks of hundreds (Skobeltsyn and Aberer, 2006) to thousands (Lu and Callan, 2006a) of peers. In contrast, our experiments are of a larger scale: using over half a million peers. To our knowledge, there is no previous scientific work that investigates the properties of networks of this size. Our motivation is that large peer-to-peer information retrieval networks deserve more attention because of their real-world potential (Lu, 2007) and that this size is in the range of operational peer-to-peer networks used for other applications (Stutzbach and Rejaie, 2006).

Markatos (2001) analysed the effectiveness of caching search results for a centralised web search engine, combined with a caching web accelerator. His experiments suggest that one out of three queries submitted

has already been submitted previously. Cache hit ratios between 25 to 75 percent are possible. He showed that even a small bounded cache (100 megabytes) can be effective, but that the hit ratios still increase slowly when the cache size is increased to several gigabytes. The larger the cache, the less difference the policy for replacing items in the cache makes. He recommends taking into account both access frequency and recency.

Baeza-Yates et al. (2007b) analysed the performance of caching for a centralised search engine. They compare both caching search results and the posting lists of terms. They conclude that caching is an effective technique for improving response times, reducing the query processing load and improving bandwidth utilisation. They find that caching posting lists has a much lower miss rate than caching search results. They also introduce their own caching policy for evicting terms from this cache that is more effective than existing cache eviction policies.

Skobeltsyn and Aberer (2006) investigated how search result caching can be used in a peer-to-peer information retrieval network. When a peer issues a query, it first looks in a meta-index, kept in a distributed hash table, to see if there are peers with cached results for this query. If so, the results are obtained from one of those peers, but if no cached results exist, the query is broadcast through the entire network. The costs of this fallback are $\mathcal{O}(n)$ for a network of n peers. In our experiments we do not distribute the meta-index, but focus only on a distributed cache. An additional difference is that they always use query subsumption: obtaining search results for subsets of the terms of the full query. They claim that with subsumption, cache hit rates of 98 percent are possible as opposed to 82 percent without. The authors also utilised bounded caches, but do not show the effect of different limits.

Bhattacharjee et al. (2003) propose using a special data structure combined with a distributed hash table to efficiently locate cached search result sets stored for particular term intersections. This is especially helpful in approaches that store an inverted index with query terms, as it reduces the amount of network traffic necessary for performing list intersections for whole queries. This could be considered to be bottom-up caching: first storing results for individual terms, then for combinations of terms up to the whole query level. Whereas subsumption is top-down caching: first storing results for the whole query, then for combinations of terms and finally for individual terms.

Table 6.1: Query Log Statistics

Users	651,647
Queries (All)	21,082,980
Queries (Unique)	10,092,307

6.1 EXPERIMENT SET-UP

Our experiments give insight into the *maximum benefits* of caching. Each experiment has been repeated at least five times, averages are reported. No differences were observed that exceeded 0.5 percent. We assume there are three types of peers: *suppliers* that have their own locally searchable index, *consumers* that have queries to issue and *mixed peers* that have both. In our experiments, the indices do not actually exist and we assume that, for each query, a fixed set of pre-merged search results is available. We also assume that all peers cooperate in caching search result sets unless otherwise noted.

6.1.1 *Dataset*

To simulate a network of peers posing queries we use a large search engine query log (Pass et al., 2006). This log consists of over twenty million queries of users recorded over a time span of three months. Each unique user in the log is a distinct peer in our experiment for a total of 651,647 peers. We made several adjustments. Firstly, some queries are censored and appear in the log as a single dash, these were removed (Brenes and Gayo-Avello, 2009). Secondly, we removed entries by one user in the log that poses an unusually high number of queries: likely some type of proxy. Furthermore, we assume that a search session lasts at most one hour. If the exact same query was recorded multiple times in this time window, these are assumed to be requests for subsequent search result pages and are used only once in the simulation. Table 6.1 shows statistics regarding the log. We play back the log in chronological order. One day in the log, May 17th 2006, is truncated and does not contain data for the full day. This has consequences for one of our experiments described later. For clarity: we do not use real search results for the queries in the log. In our experiments we assume that specific subsets of peers have search result sets and obtain experimental results by counting hits only.

6.1.2 *Tracker*

For query routing we introduce the *tracker* that keeps track of which peers cache search result sets for each query. This is inspired by BitTorrent (Cohen, 2003). However, in BitTorrent the tracker is used for locating a specific file: *exact search*. A hash sequence based on a file's contents yields a list of all peers that have an exact copy of that particular file. In contrast, we want to obtain a list of peers that have cached search result sets for a specific free-text query: *approximate search*. See Section 3.5 for a more detailed explanation.

The tracker can be implemented in various ways: as a single dedicated machine, as a group of high capacity machines, as a distributed hash table or by fully replicating a global data index over all peers. Let us first explore if a single machine solution is feasible. The tracker needs to store only queries and mappings to all peers in the network. We can make a rudimentary calculation based on our log: storing IPv6 addresses for all the 650,000 peers would take about 10 megabytes. Storing all the queries in the log, assuming an average query length of 15 bytes (Pass et al., 2006; McNamee and Mayfield, 2004), would take about 315 megabytes. Even including the overhead of data structures we could store this within 1 gigabyte. Consider that most desktop machines nowadays have 4 gigabytes of main memory and disk space in the range of terabytes. However, storage space is not the only aspect to consider, bandwidth is equally important. Assume the tracker is connected to a 100 megabit line, that can transfer 12.5 megabytes per second. The tracker receives queries, 15 bytes each, and sends out sets of peer addresses, let us say 10 per query: 160 bytes. This means a single machine can process 81,920 queries per second. This works even if 12 percent of the participating peers query it every second.

In our calculation we have made many idealisations, but it shows that a single machine can support a large peer-to-peer network. Nevertheless, there are three reasons to distribute the tracker. Firstly, a single machine is also a single point of failure: if it becomes unreachable, due to technical malfunction or attacks, the peer-to-peer network is rendered useless. Secondly, a single machine may become a bottleneck even outside its own wrongdoing: for example due to poor bandwidth connections of participating peers. Thirdly, putting all this information in one place opens up possibilities for manipulation. See also Section 3.5.

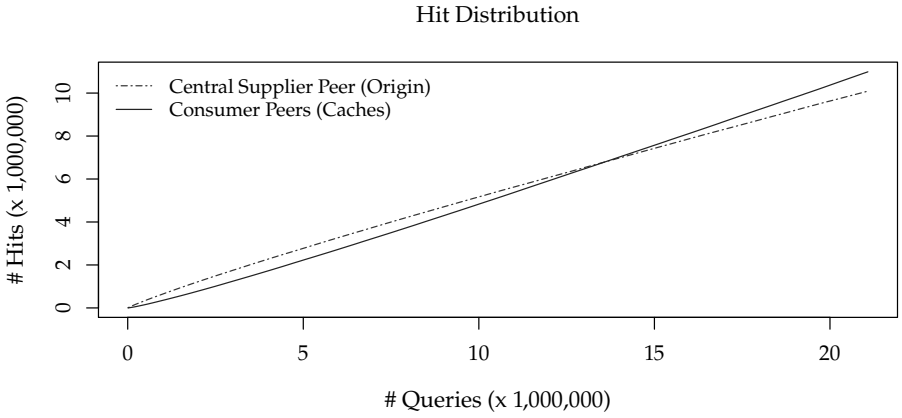


Figure 6.1: Distribution of hits when peers perform result caching.

6.2 CENTRALISED EXPERIMENTS

Let us first consider the case where one supplier peer in the system is the only peer that can provide search results. This peer does not pose queries. This scenario provides a baseline that resembles a centralised search system. Calculating the query load is trivial: all 21 million queries *have to be* answered by this single central supplier peer. However, what if the search results provided by the central supplier peer can be cached by the consuming peers? In this scenario, the tracker makes the assumption that all queries are initially answered by the central peer. When a consuming peer asks the tracker for advice for a particular query, this peer is registered at the tracker as caching search results for that query. Subsequent requests for that same query are offloaded to caching peers. When there are multiple caching peers for a query, one is selected randomly. Furthermore, we assume unbounded caches for now.

Figure 6.1 shows the number of search results provided by the central supplier peer and the summed hits on the caches at the consumers. Results for about half of the queries need to be given by the supplier at least once. The other half can be served from consumer caches. Caching can reduce the load on the central peer by about 50 percent, suggesting about half the queries we see are unique, consistent with Zimmer et al. (2008). Skobeltsyn and Aberer (2006) find only 18 percent of their queries are

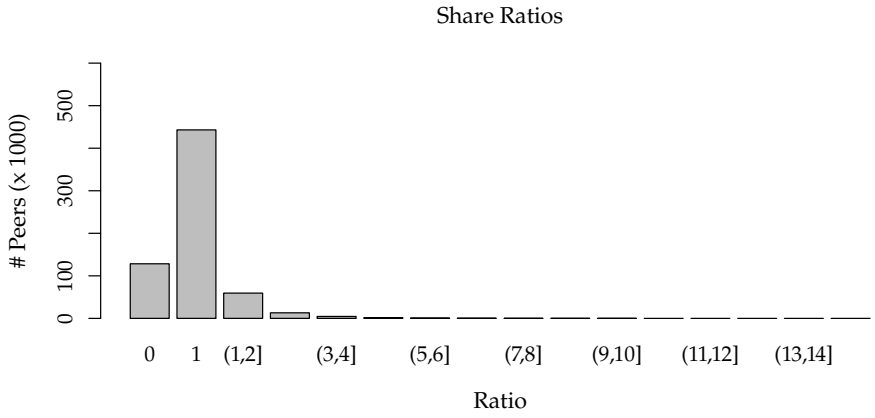


Figure 6.2: Observed share ratios.

unique, perhaps because they use a Wikipedia trace. This is inconsistent with our findings and those of others (Croft et al., 2010, p. 183).

Caching becomes more effective as more queries flow through the system, because there are increasingly more repeated queries and less unique queries. So, you always see slightly fewer new queries. Perhaps there is mild influence of Heap's law at the query level (Croft et al., 2010, p. 83).

How many results can a peer serve from its local cache and for how many does it have to consult caches of other peers? The local cache hit ratio climbs from around 22 percent for several thousand queries to 39 percent for all queries. These local hits are a result of re-search behaviour (Teevan et al., 2007). The majority of hits, 61–78 percent, is external.

Let us take a look at external hits. We define a peer's share ratio as:

$$shareratio = \#cachehits / \#queries, \tag{6.1}$$

where *#cachehits* is the number of external hits on a peer's cache: all hits not caused by its own queries, and *#queries* the number of queries issued by the peer. A *shareratio* of 0 means a peer's cache is never used to answer external queries, 1 that a peer handles as many queries as it poses and above 1 indicates it serves results for more queries than it sends.

Figure 6.2 shows about 20 percent of peers does not share anything. The majority, 68 percent, at least serve results for some queries, whereas 12 percent (80,000 peers) serve results for more queries than they issue.

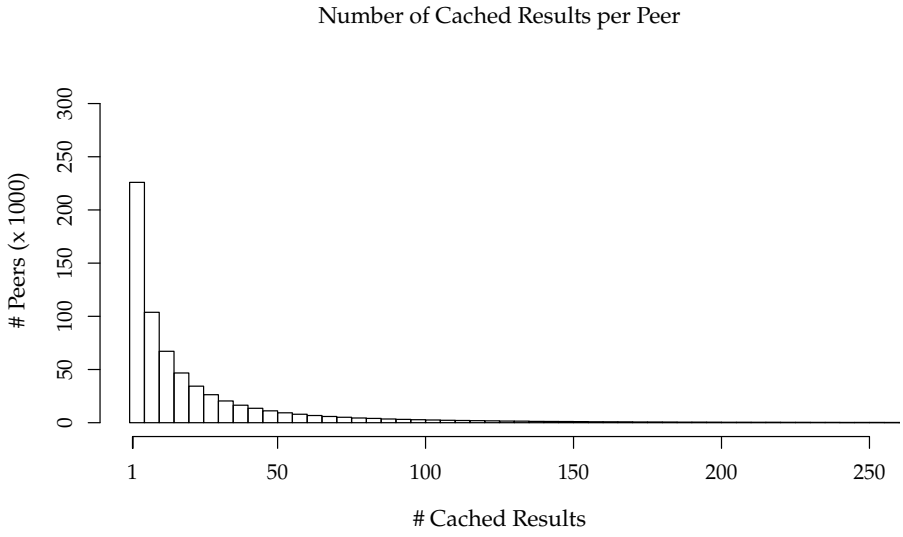


Figure 6.3: Observed cache sizes. Each bar represents 5 search result sets. The horizontal axis extends to 7,500. The visible part of the graph covers 99.2 percent of all peers, each peer caches at least one result set.

6.2.1 *Required Cache Sizes*

So far we have assumed caches of unbounded size. This is not very realistic since machines in a peer-to-peer network have limited resources. Let us try to find out how big a cache we really need. Figure 6.3 shows the distribution of the number of cached items per peer for the previous experiment. We see that the vast majority of peers, about 225 thousand, cache 1–5 result sets. The graph is cut-off after 250 results, but extends to the highest number of cached items seen at a single peer: about 7,500.

How much space does it take to store a set of search results? Assume that each set of results consists of 10 items and that each item consists of a URI, a summary and some additional metadata, taking up 1 kilobyte of space: 10 kilobytes per set. Even for the peer with the largest number of cached results this takes only 73 megabytes. However, a cache of 5 items, 50 kilobytes, is much more typical. Table 6.2 gives an overview of storage requirements for various search result set sizes. Most modern personal computers can keep the entire cache in main memory, even with a supporting data structure like a hash table.

Table 6.2: Cache Storage Requirements in Megabytes (MB)

Result Set Size	Low (5)	Medium (100)	High (7,500)
10	0.05	1	73
100	0.5	10	730
1,000	5	98	7,300

Assumes each search result takes up 1KB: 5 results for low, 100 for medium and 7,500 for high.

At the start of this section we discussed the share ratios: the relationship between the number of queries posed by a peer and the number of times that same peer's cache was used by an other peer. A relationship that is related is the number of queries posed versus the number of search result sets cached. If a peer poses a query it automatically also caches results for that query so this is always at least 1, but may be more when it consults its own cache frequently. The average is 1.19 standard deviation ± 0.63 and the median is 1.06. Thus: an interesting property of the set-up we have discussed is that machines that pose a lot of queries also need to have more capacity in order to cache results: he who consumes the most needs to provide the most.

6.2.2 *Bounded Caches*

As suggested in the previous subsection: it is possible to use unbounded caches for at least some time. However, it is not very desirable to do so for three reasons. Firstly, if systems run for an extended period of time, the cache has to be bounded somehow, since they will run out of space eventually. Secondly, there is no point in keeping around result sets that are not requested any more. Thirdly, in a real system, search results expire due to index updates at the suppliers.

We want to limit the size of the cache at some maximum number of search result sets to keep. To this end we investigate three different cache policies, with different limits on the cache size. When a new result set has to be inserted in the cache and the cache limit is reached, the cache policy comes into play.

The most basic policy when the cache limit is reached is to throw out a random result set, this is called *Random Replacement* (RR) (Podlipnig and Böszörmenyi, 2003). The advantage of this method is that it requires no additional administration at all. The downside is that we may be

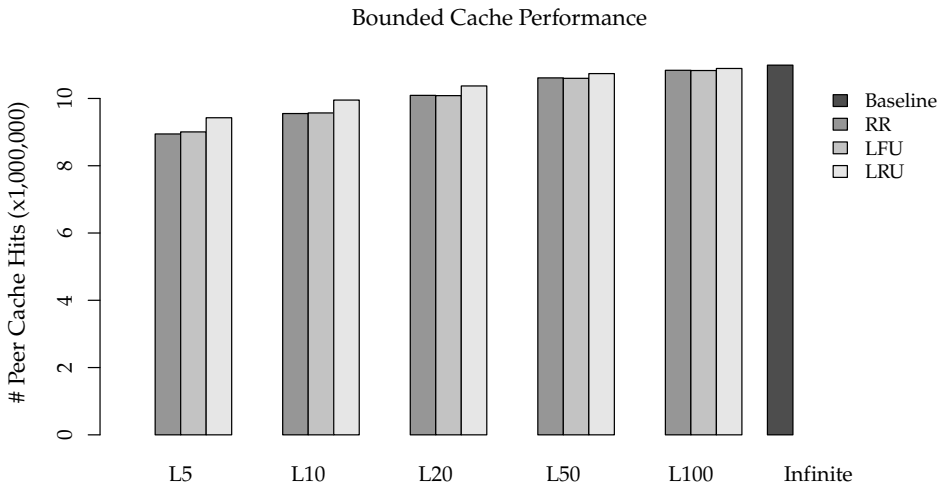


Figure 6.4: Bounded cache performance. The total number of queries is 21,082,980. The bars show the amount serviceable from peer caches for various per-peer cache size limits (5, 10, 20, 50 and 100) and strategies (RR, LFU and LRU). The rightmost bar shows the performance with unbounded caches.

throwing away valuable sets from the cache. What is valuable is conventionally expressed using either frequency or recency which provides the motivation for the two other policies tested (Markatos, 2001). In the *Least Frequently Used* (LFU) policy, the search result set that was consulted the least amount of times, meaning: that has the least hits, is removed. In *Least Recently Used* (LRU) the search result set that was least recently consulted is removed. In the case of LFU there can be multiple 'least' sets that have the same lowest hit count. If this occurs, a random choice is made among those sets.

Figure 6.4 shows the hit distribution for the baseline unbounded cache and the RR, LFU and LRU caching strategies with various cache limits after running through the entire log. Experiments were conducted with per-peer cache limits of 5, 10, 20, 50 and 100 result sets. We can see that a higher cache limit brings the results closer to the unbounded baseline, which is what we would expect. The most basic policy, Random Replacement, performs worst, particularly when the cache size is small (L5, L10).

However, it performs almost the same as the LFU algorithm for large caches (L50, L100). In fact LFU performs quite poorly across the board, inconsistent with Zimmer et al. (2008). We believe this is caused by the fact that there can be many sets with the same hit count in a cache, which degrades LFU to RR. For all cases the LRU policy is clearly superior. Although, the higher the limit, the less it matters what policy is used, also found by Markatos (2001). L100/LRU with 10 results per set takes only 1 megabyte of space and achieves 99.1 percent of the performance of using unbounded caches.

6.2.3 *Optimisations*

In this subsection we use unbounded caches and investigate the impact of several optimisations: *stopword removal*, *reordering*, *stemming* and *query subsumption*. These techniques map multiple queries, that were previously considered distinct, to one common query representation. Since the number of representations is lower than the original number of queries, the strain of serving original search results on the central supplier peer is also lower. This capitalises on the fact that there are cached copies of search result sets available for similar queries.

For stopword removal we remove words from queries that match those in a stopword list used by Apache Lucene consisting of 33 English terms. For reordering, the words in the query are alphabetically sorted, for example: from 'with MacCutcheon you live happily ever after' to 'after ever happily live MacCutcheon with you'. The last common technique is stemming, for example from 'airplane' and 'airplanes' to 'airplan'. This example also shows the well known drawback of stemming: that of reducing distinct meanings to an unrelated form. We used the Porter2 English stemming algorithm (Porter, 2001).

We ran experiments with the three described techniques individually and combined. The first five rows of Table 6.3 show the results. Without any optimisations, the central peer has to serve 47.9 percent of all queries. Applying stopping or re-ordering only marginally improves this by about a half percent. Stemming offers the most improvement: over 1.6 percent. Combining techniques is effective and yields a 3.1 percent improvement, exceeding the sum of the individual techniques.

One final technique, that is less commonly used, is query subsumption (Skobeltsyn and Aberer, 2006). When a full query yields no search results,

Table 6.3: Cache Hits for Various Optimisations (×1,000)

	Central		Internal		External	
Baseline	10,092	47.9%	4,237	20.1%	6,754	32.0%
Sto(P)	9,993	47.4%	4,265	20.2%	6,824	32.4%
(R)eorder	9,992	47.4%	4,274	20.3%	6,816	32.3%
(S)tem	9,768	46.3%	4,359	20.7%	6,955	33.0%
P+R+T	9,449	44.8%	4,462	21.2%	7,172	34.0%
S(U)bsumption	6,352	26.5%	7,239	30.2%	10,365	43.3%
P+R+T+U	5,773	22.3%	8,335	32.1%	11,834	45.6%

Shows what party answers what query as an absolute number and percentage. The first five rows have a total query count of 21 million, the sixth 24 and the seventh 26 million.

subsumption breaks the query into multiple subqueries. This process iterates with increasingly smaller subqueries until at least one of these queries yields search results. The subqueries generated are combinations, with no repetition, of the terms in the full query $\mathcal{Q}$. The length goes down each iteration, starting from $\text{len}(\mathcal{Q}) - 1$ terms to a minimum of 1 term. For example, given a resultless query $\mathcal{Q}$ of length three: '*a b c*', we next try the three combinations of length two: '*a b*', '*a c*' and '*b c*'. If that yields no results we try all combinations of length one, which are the individual terms '*a*', '*b*' and '*c*'. The rationale for iterating top-down, from the whole query to the individual terms, is that longer queries are more specific and are thus expected to yield more specific, higher-quality, results. Long queries can generate an unwieldy number of possible subqueries. Therefore, we restrict the maximum number of generated combinations at any level to 1000, to keep it manageable.

In our experiment we evaluate at each iteration whether there is a query that yields at least one search result set. If so, all queries at that same iteration level, for which there are cached result sets, generate cache hits. Hence, for the example above, if for the full query '*a b c*' search results are not available, but there is at least one result at the level of individual terms: '*a*', '*b*' and '*c*'. The full query can generate 1–3 cache hits: one for each individual term for which a result set is available. This thus causes the total amount of cache hits to increase beyond the number of original queries and simulates the effect of increased query load for merging result sets from multiple peers.

Table 6.3 shows the results in the bottom two rows. As mentioned the total amount of cache hits is different: 24 million for subsumption alone, a 13.6 percent increase. Nevertheless, performance improves with 21.4 percent less strain on the central peer. Combining subsumption with the three other techniques further increases the query total to nearly 26 million, but also further decreases the central peer load by 4.2 percent. The trade-off with subsumption is a higher total query load, but a lower load on the central peer. It reduces query-level caching to term-level caching that is known to have higher hit rates (Croft et al., 2010, p. 183)

All the discussed optimisations decrease precision in favour of higher recall. Hence, the quantity of search results for a particular query goes up, but the quality is likely to go down. Whether such a trade-off is justified depends on how sparse the query space is to begin with. However, for a general search engine, it certainly makes sense to apply some, if not all, of these techniques.

6.3 DECENTRALISED EXPERIMENTS

Now that we have shown the effectiveness of caching for offloading one central peer, we make the scenario more realistic. Instead of a central peer we introduce n peers that are *both* supplier and consumer. These mixed peers are chosen at random. They serve search results, pose queries and participate in caching. The remaining peers are merely consumers that can only cache results.

The central hits in the previous sections become hits per supplier in this scenario. We assume unbounded caches and no optimisations to focus on the differences between the centralised and decentralised case. How does the distribution of search results affect the external cache hit ratios of the supplier peers? We examine two distribution cases:

SINGLE SUPPLIER For each query there is always only exactly one supplier with unique relevant search results.

MULTIPLE SUPPLIERS The number of supplier peers that have relevant results for a query depends on the popularity. There is always at least one supplier, but the more popular a query the more suppliers there are: up to all n suppliers for very popular queries.

For simplicity we assume in both cases that there is only one set of search results per query. In the first case this set is present at exactly one supplier

Table 6.4: Original Search Results and Cache Hits (×1,000)

	Single	Multiple
Suppliers (origin)	11,599	12,111
Consumers internal (caches)	3,683	3,930
Consumers external (caches)	5,801	5,042

10,000 peers are suppliers operating in mixed mode.

peer. However, the second case is more complicated: among the mixed peers we distribute the search results by considering each peer as a bin covering a range in the query frequency histogram. We assume that for each query there is at least one peer with relevant results. However, if a query is more frequent it can be answered by more peers. The most frequent queries can be served by *all* n suppliers. The distribution of search results is, like the queries themselves, *Zipf* over the suppliers. We believe that this is realistic, since popular queries on the Internet tend to have many search results as well. In this case the random choice is between a variable number m of n peers that supply search results for a given query. Thus, when the tracker receives a query for which there are multiple possible peers with results, it chooses one randomly.

We performed two experiments to examine the influence on query load. The first is based on the single supplier case. The second is based on the multiple suppliers case. For multiple suppliers we first used the query log to determine the popularity of queries and then used this to generate the initial distribution of search results over the suppliers. This distribution is performed by randomly assigning the search results to a fraction of the suppliers depending on the query popularity. Since normally the query popularity can only be approximated, the results represent an ideal outcome. We used $n = 10{,}000$ supplier peers in a network of 651,647 peers in total (about 1.53 percent). This mimics the World Wide Web with a small number of websites and a very large number of surfing clients.

Figure 6.5 and Table 6.4 show the results. The number of original search results provided by the suppliers is about five percent higher than in the central peer scenario. This is the combined effect of no explicit offloading of the supplier peers by the tracker, and participation of the suppliers in caching for other queries. In the second case there is slightly more load on the supplier peers than in the first case: 57 percent versus 55

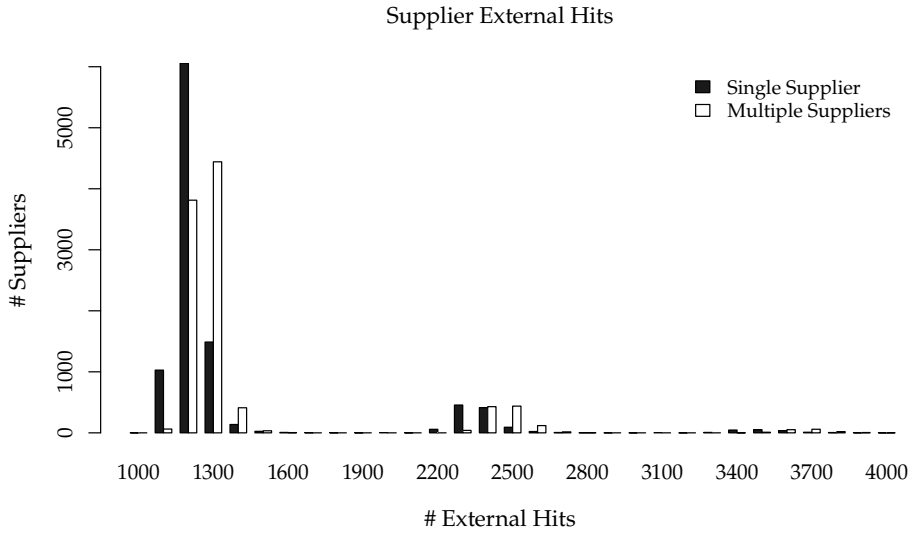


Figure 6.5: Supplier external hit distributions ($n = 10{,}000$ suppliers).

percent. The hit distribution in Figure 6.5 is similar even though the underlying assumptions are different. About 87 percent of peers answer between 1,000 and 1,500 queries. A very small number of peers answers up to about five times that many queries. Differences are found near the low end, that seems somewhat more spread in the single supplier than in the multiple suppliers case. Nevertheless, all these differences are relatively small. The distribution follows a wave-like pattern with increasingly smaller peaks: near 1300, 2500, 3700 and 4900 (not shown). The cause of this is unknown.

6.3.1 *Churn*

The experiments thus far have shown the maximum improvements that are attainable with caching. In this subsection we add one more level of realism: we no longer assume that peers are on-line infinitely. We base this experiment on the single supplier case where the search results are uniformly distributed over the suppliers. The query log contains timestamps and we assume if a specific peer has not issued a query for some period of time, its session has ended and its cache is temporarily no longer available. If the same peer issues a query later, it comes back on-line and its

cache becomes available again. This simulates churn in a peer-to-peer network where peers join and depart from the network. We assume the presence of persistent peer identifiers, also used in real-world peer-to-peer systems (Pouwelse et al., 2008a). All peers, including supplier peers, are subject to churn. For bootstrapping: if there are no suppliers on-line, an off-line one is randomly chosen to provide results.

Assuming that all peers are on-line for a fixed amount of time is unrealistic. Stutzbach and Rejaie (2006) show that download session lengths, post-download lingering time and the total up-time of peers in peer-to-peer file sharing networks are best modelled by using *Weibull distributions*. However, our scenario differs from file sharing. An information retrieval session does not end when a search result has been obtained, rather it spans multiple queries over some length of time. Even when a search session ends, the machine itself is usually not immediately turned off or disconnected from the Internet. This leads us to two important factors for estimating how long peers remain joined to the network. Firstly, there should be some reasonable minimum that covers at least a browsing session. Secondly, up-time should be used rather than 'download' session length. As soon as a peer issues its first query we calculate the remaining up-time of that peer in seconds as follows :

$$remaininguptime = 900 + (3600 \cdot 8) \cdot w, \tag{6.2}$$

where w is a random number drawn from a Weibull distribution with $\lambda = 2$ and $k = 1$. The w parameter is usually near 0 and very rarely near 10. The up-time thus spans from at least 15 minutes to at most about 80 hours. About 20 percent of the peers is on-line for longer than one day. This mimics the distribution of up-times as reported in Stutzbach and Rejaie (2006), making the assumption that the uptime of peers in file sharing systems resembles that of information retrieval systems.

Figure 6.6 shows the results: the number of origin search results served by suppliers as well as the number of internal and external hits on the caches of consumer peers. We see the number of supplier hits increases to over 12.75 million: over 1.16 million more compared to the situation with no churn. The majority of this increase can be attributed to a decrease in the number of external cache hits. The dotted cloud shows the size of the peer-to-peer network on the right axis: this is the number of peers that is on-line simultaneously. We can see that this varies somewhere between

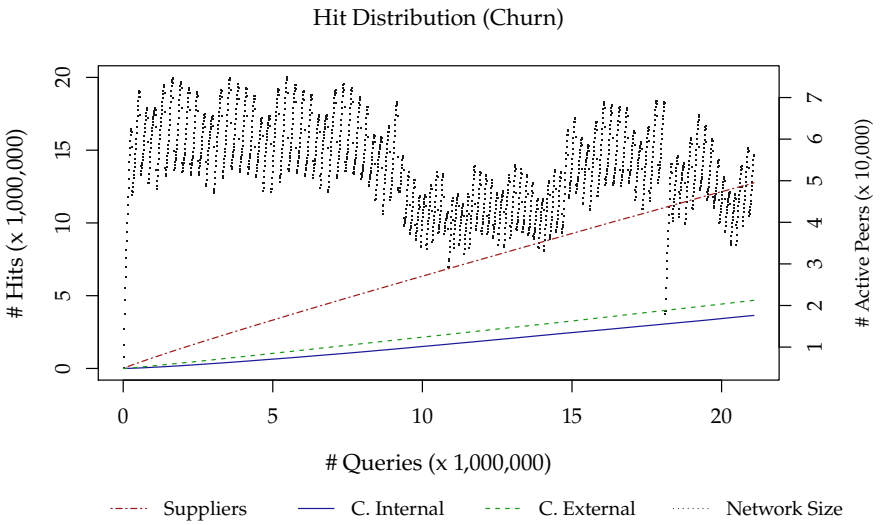


Figure 6.6: Distribution of hits under churn conditions ($n = 651{,}647$ peers).

about 30,000 and 80,000 peers. There is a sudden dip in the graph caused by the earlier described log truncation on one specific date. A rhythmic pattern can also be seen, which may be circadian in nature.

We also ran this distributed experiment with churn with a L100 LRU cache and all optimisations enabled: stopping, re-ordering, stemming and query subsumption. This yields a cache hit ratio of 69 percent (for 25.45 million queries) for this most realistic scenario.

6.3.2 *Reputations*

As a final experiment we drop the cooperativeness assumption partially and assume that there are some peers in the network that do not want to cache. From a peer's perspective a cache takes up space and cache hits consume bandwidth. Hence, if a peer can get away with not caching any search results while it can still use the caches of other peers, that would give it the largest utility at the lowest cost. This would be somewhat similar to a situation that can occur in BitTorrent: a peer trying to download from other peers without uploading back to them. Whilst this behaviour is optimal for the peer, it is not optimal for the network as a whole. To prevent a 'tragedy of the commons' BitTorrent uses a tit-for-tat mecha-

nism: if I can download from you, I will give you some upload in return. Unfortunately, this is difficult to apply directly to peer-to-peer information retrieval, since the interests of peers are usually not symmetric at the exact same time (see also Section 3.5.2). Hence, we introduce a simple binary reputation system to handle this. A peer has a positive reputation by default, but its reputation can become negative based on its behaviour.

In this experiment we divide the consumer peers into two groups: those that cache and those that do *not* cache. There is a 50 percent probability of ending up in either group. The supplier peers always cache, we assume unbounded caches and all peers start out with a positive reputation. When a peer poses a query, receives advice from the tracker, and, based on that, contacts a peer that caches, everything proceeds as normal. However, if the peer contacted does not cache it is 'caught'. This is reported to the tracker that sets the reputation of the offending peer to negative. Thereafter, the querying peer will try to obtain cached results from other peers. In worst case it has to resort to a supplier. Before a caching peer provides search results it checks the reputation of the requesting peer at the tracker. When the requesting peer's reputation is negative, caching peers will not interact with it anymore, a grim trigger, and thus the peer has to resort to always contacting the suppliers. These are assumed to be slower due to their higher query processing load.

We perform the experiment using the single supplier case with $n = 10{,}000$ suppliers in mixed mode, consuming and supplying, and no optimisations, like stemming or subsumption, enabled. Over the experiment runs there were an average 321,108 peers out of the 651,647 that engaged in non-caching behaviour, a little less than half of the total number of peers, as the suppliers always cache. The average number of peers getting caught not caching was 272,691: nearly 85 percent. Hence, the detection mechanism is fairly effective in identifying free-riding peers.

Table 6.5 shows the total number of hits for each type of peer and the amount of those hits that were on the suppliers. We define that having less supplier hits is better, as it means the other hits are being served from peer caches. We can see that the caching consumer peers, which includes the suppliers themselves operating in mixed mode, are able to utilise the caches of other peers most effectively: only about 54 percent of the hits is on suppliers. The second group are the peers that are not-caching and that are also not caught doing so, they are already less effective, requiring the suppliers for over 61 percent of their queries. The penalty

Table 6.5: Hits for the Reputation Experiment (×1,000)

	Total Hits	Supplier Hits	%
Caching Consumers	11,181	6,045	54.1%
Non-Caching Consumers (Not Caught)	1,569	964	61.4%
Non-Caching Consumers (Caught)	8,332	6,811	81.7%

that they are already paying by not caching is that not caching also has the consequence that there can be no internal cache hits. For repeated queries such peers thus have to utilise the network again: shooting themselves in the foot. Finally, there is the group of non-caching consumers that are caught at some point, they have to contact the suppliers for nearly 82 percent of all queries. Hence, they can still get away with using some of the caches, but they are heavily penalised in terms of performance when they are caught.

6.4 CONCLUSION

We conducted several experiments that simulate a large-scale peer-to-peer information retrieval network. Our research questions can be answered as follows:

1. At least 50 percent of the queries can be answered from search result caches in a centralised scenario. For the decentralised case cache hits up to 45 percent are possible.

2. Share ratios, the rate between cache hits and issued queries, are skewed which suggests that additional mechanisms are needed for cache load balancing.

3. The typical cache is small, with outliers for eager consuming peers. Peers that issue many queries also provide lots of cached results.

4. Small bounded caches approach the performance of unbounded caching. The Least Recently Used (LRU) cache replacement policy consistently outperforms the other policies. However, the larger the cache, the less the policy matters. If each peer were to keep just 100 cached search result sets, the performance is 99.1 percent of unbounded caches.

5. We have shown that stopword removal, stemming and alphabetical term re-ordering can be combined to boost the amount of cache hits by about 3.1 percent. Query subsumption can increase cache hits by 21.4 percent, to nearly 80 percent, but also imposes a higher total query load. All of the optimisation techniques trade search result quality for quantity. However, they all improve the effective usage of caches.

6. Introducing churn reduces the maximum attainable cache hits to 33 percent (-12 percent) without optimisations and 69 percent (-11 percent) with optimisations.

7. A simple reputation scheme is effective for detecting 85 percent of free-riding peers, that can be effectively penalised by redirecting them to slower peers for obtaining search results.

We have shown the potential of caching under increasingly realistic conditions using a large query log. Caching search results significantly offloads the origin suppliers that provide search results under all considered scenarios using this log. Our set-up has nice scaling properties, where the supply matches with the demand for search results. These experiments could be extended by adding extra layers of realism. For example, individual search results could be used instead of fixed result sets, allowing merging and construction of new search result sets.

We have explored several fundamental caching policies showing that Least Recently Used (LRU) is the best approach for our scenario. However, more advanced policies could be explored that include frequency as a component such as 2Q, LRFU or ARC (Markatos, 2001; Megiddo and Modha, 2003; Podlipnig and Böszörmenyi, 2003). These techniques combine advantages of LRU and LFU. Nevertheless, in reality there may be more than just LRU/LFU to take into account. For example queries pertaining to current events for which the relevant search results frequently change. The result sets for such queries should have a short time to live, whereas there are queries for which the search results change very rarely, these could be cached much longer. Making informed decisions about invalidation requires knowledge about the rate of change for particular queries (Blanco et al., 2010). Perhaps this information, or an estimate thereof, could be made an integral part of the search results, similar to the way in which Domain Name System (DNS) records work. Further-

more, we assumed the capacity of the tracker is unbounded. However, a policy similar to what is used to maintain peer caches could be applied there too. This does have the consequence that the tracker loses track of search results that are available, but for which the mappings have been thrown away. The fact that caching requires keeping track of which peers issued what query raises privacy concerns that we have not addressed here. This basic problem is shared with BitTorrent and other file sharing networks and can be alleviated to some extent using onion routing at the cost of additional latency, although this does not solve the problem completely (Goldschlag et al., 1999; Le Blond et al., 2010). Finally, we have not investigated peer selection and result merging in this chapter, both of which are relevant for real-world systems (Lu, 2007). An investigation of peer selection is contained in Chapter 5.

7 CONCLUSION

'Some things need to be
believed to be seen...'

Guy Kawasaki

Many important concepts and distinctions have been introduced and the research topics from the introduction chapter have been explored. What conclusions can we draw from our investigation and experiments?

This thesis began with the aim of fusing peer-to-peer computing with information retrieval, as to provide the groundwork for developing a distributed alternative to the contemporary centralised web search engines. We started with a set of research topics in the introductory chapter, each of which we will address in the following paragraphs:

> *RT 1. What is peer-to-peer information retrieval, how does it differ from related research fields and what are its unique challenges?* (Chapter 2)

Peer-to-peer information retrieval uses the bandwidth, storage and processing capacity of machines at the edge of the network to provide a free-text search service capable of returning relevant search results for queries with minimal delay. It shares the three main tasks central to all peer-to-peer networks: searching, locating and transferring. However, in peer-to-peer information retrieval, the emphasis is on distributing the first task: searching. This differs from peer-to-peer file sharing where the emphasis is on the transfer task. When searching, the goal is to maximise the relevance of the results while minimising the latency to obtain those results.

In contrast, when transferring, the goal is maximising the transfer speed for obtaining an exact copy of a resource selected in the initial search step. Furthermore, searching in information retrieval networks focuses on free text search within the contents of documents, whilst searching in file sharing networks conventionally considers only the names of documents, not their actual content.

Peer-to-peer information retrieval overlaps with federated information retrieval, and shares its central challenges: resource description, collection selection and result merging. However, the key difference is that in federated information retrieval all queries and search results are channelled through a centralised mediator party. This enforces a strict division between the consumers and providers of information. In peer-to-peer information retrieval the predominant mode of interaction is with other peers, not with a centralised mediator.

We identified a number of key challenges for peer-to-peer networks with respect to usage guarantees, behaviour of peers and evaluation. Challenges unique and important to peer-to-peer information retrieval are minimising latency, maintaining index freshness and developing test collections (Tigelaar et al., 2012). We also presented an economic perspective on peer-to-peer search.

> *RT 2. What general peer-to-peer architectures, information retrieval systems and optimisation techniques already exist?* (Chapter 2 and Chapter 3)

We organised the main architectures around index placement, and identified four types of indices that are used: the centralised global index, the distributed global index, aggregated local indices and strict local indices. Each has different consequences for query routing and processing. Most existing research and systems today use a combination of these, with the distributed global index and aggregated local indices being the most popular. We have also shown the difference between a one-step index, where keys map directly to documents, and a two-step index, where the keys map to peers and the peers contain document mappings.

We showed various optimisations that can be applied in peer-to-peer information retrieval networks to reduce bandwidth usage and latency and to improve the quality and quantity of the search results returned. Besides this we gave an overview and classification of existing systems, like Minerva, ALVIS and PHIRST (Bender et al., 2005b; Luu et al., 2006;

Rosenfeld et al., 2009). We also looked at applied optimisations, like: search result caching, topical clustering and global index replication. Finally, we discussed the future of peer-to-peer information retrieval, indicating specific challenges and key areas to focus on, the most important of which are: emphasising precision over recall, focusing on search results instead of documents, combining the strengths of local and global indices, applying clustering, using search result caching, performing load balancing and using relevance feedback (Tigelaar et al., 2012).

We gave an impression of how economics can be applied to peer-to-peer systems and presented a peer-to-peer information retrieval architecture that uses BitTorrent as inspiration (Cohen, 2003). Whilst BitTorrent is a file sharing peer-to-peer network, that focuses on quickly transferring files, its basic incentive mechanism can be adapted for minimising latency and maximising relevance when conducting searches. Central to BitTorrent is the presence of a tracker that maintains global knowledge. While we often think of this tracker as a centralised party, its functionality can be distributed over the peers that make up the network. In fact, this natural evolution has already taken place for BitTorrent.

> *RT 3. How can content at uncooperative peers, and existing external search systems, be made searchable?* (Chapter 4)

For modelling the contents of uncooperative peers we investigated a technique that uses probe queries and downloads the documents in the returned search results: query-based sampling (Callan, 2000). The sample documents obtained this way are used as resource description of the peer. New queries can be scored against this to estimate the relevance of a peer for a particular query. Conventionally, random queries are used for probing. We experimented with different ways of query selection. However, only using the least-frequent terms as queries showed a very marginal improvement in terms of similarity to using the full language model of the peers. Furthermore, we presented a method where the documents themselves are no longer downloaded. Instead, it uses the document summaries that are part of each search result. This gives better modelling performance per unit of bandwidth consumed. Whilst this approach takes longer to converge to a representative resource description: more query-result retrieval iterations are needed, its strength is that it does not require any additional downloading beyond the search results that are already returned anyway (Tigelaar and Hiemstra, 2009, 2010b).

RT 4. How could we best represent the content of cooperative peers to maximise the relevance of search results? (Chapter 5)

It is difficult to give a general recommendation for peer representation, as this depends on the characteristics of the collections of each peer. Although using peer size is a poor selection criterion by itself, it performs better than random and is effective when combined with other methods. It seems that representing peers by simple term vocabularies combined with size weighting is an effective method that gives results competitive with the more conventional document sampling approach. For certain collections term-weighted vocabularies can offer even better performance.

RT 5. How can we distribute the query load and minimise the latency for retrieving search results? (Chapter 6)

We have shown that by having peers maintain fairly small search result caches, holding onto copies of search result sets for 100 previously issued queries, is effective for distributing the query load. With optimisations nearly 70 percent of queries can be answered from peer caches despite peers turning on and off. However, the share ratios of peers, the number of cached result sets a peer holds versus the number of queries a peer poses, is skewed. This suggests that additional mechanism are needed to enforce fair load balancing (Tigelaar and Hiemstra, 2011; Tigelaar et al., 2011). We showed that a simple reputation system can discourage peers from free-riding in a caching peer-to-peer information retrieval network.

We have seen the strengths of peer-to-peer networks: each peer is equal and can both supply and consume resources. Important advantages of these networks are that they have no central point of failure, are self-organising and self-scaling. However, we have also learned about the challenges such an environment poses. A peer-to-peer network typically consists of thousands of low-cost machines all with different processing and storage capacities, as well as different link speeds. This makes it more difficult to provide a consistent quality of service in terms of retrieval latency and data availability. Not all peers are on-line all the time, and as a result a peer-to-peer network is subject to churn: the joining and departing of peers. For information retrieval, one of the central problems in this environment is routing queries to available peers with relevant results. At least some of these challenges can be solved by using incentives for participation.

Peer-to-peer technology has the potential to offer a robust solution to the ethical and technical problems that plague centralised solutions and deserves more attention. In this thesis we have given a clear definition of peer-to-peer information retrieval and what distinguishes it from related and overlapping fields, like file sharing and federated information retrieval. Furthermore, we provided an overview of what has been done so far and what choices can be made when implementing a system in practice. Both of these aspects are important for future research and development of real-world systems. We have also investigated several specific problems experimentally, particularly focusing on peer representations and load balancing mechanisms. These concrete contributions may aid in the design and construction of a large-scale peer-to-peer web search engine, which is the primary goal of this thesis.

BIBLIOGRAPHY

Aberer, K. and Hauswirth, M. 2002. An Overview on Peer-to-Peer Information Systems. In *Proceedings of the Workshop on Distributed Data & Structures (WDAS'02)*, Paris, France. Proceedings in Informatics 14, Carleton Scientific 2002, 171–188. (pages 16, 19, and 22).

Adamic, L. A., Lukose, R. M., Puniyani, A. R., and Huberman, B. A. 2001. Search in Power-law Networks. *Physical Review E 64*, 4, 046135:1–046135:8. (pages 19 and 51).

Akavipat, R., Wu, L.-S., Menczer, F., and Maguitman, A. G. 2006. Emerging Semantic Communities in Peer Web Search. In *Proceedings of the Workshop on Information Retrieval in Peer-to-Peer Networks (P2PIR'06)*, Arlington, VA, US. ACM Press, New York, NY, US, 1–8. (pages 4, 49, 50, 51, 52, and 54).

Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. 2004a. Discouraging Free Riding in a Peer-to-Peer CPU-Sharing Grid. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, Honolulu, HI, US. IEEE Computing Society, Los Alamitos, CA, US, 129–137. (pages 69 and 73).

Andrade, N., Mowbray, M., Cirne, W., and Brasileiro, F. 2004b. When can an Autonomous Reputation Scheme Discourage Free-riding in a Peer-to-Peer system? In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04)*, Chicago, IL, US. IEEE Computer Society, Los Alamitos, CA, US, 440–448. (pages 69 and 73).

Arguello, J., Callan, J., and Díaz, F. 2009a. Classification-based Resource Selection. In *Proceeding of the ACM Conference on Information and Knowledge Management (CIKM'09)*, Hong Kong, CN. ACM, New York, NY, US, 1277–1286. (page 128).

Azzopardi, L., de Rijke, M., and Balog, K. 2007. Building Simulated Queries for Known-item Topics: An Analysis using Six European Languages. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'07)*, Amsterdam, NL. ACM Press, New York, NY, US, 455–462. (page 108).

Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., and Silvestri, F. 2007b. The Impact of Caching on Search Engines. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'07)*, Amsterdam, NL. ACM Press, New York, NY, US, 183–190. (pages 49 and 159).

Bailey, P., Craswell, N., and Hawking, D. 2003. Engineering a Multi-purpose Test Collection for Web Retrieval Experiments. *Information Processing & Management 39*, 6, 853–871. (pages 39, 87, and 122).

Baillie, M., Azzopardi, L., and Crestani, F. 2006a. Adaptive Query-based Sampling of Distributed Collections. In *Proceedings of the Conference on String Processing and Information Retrieval (SPIRE'06)*, Glasgow, UK. Springer-Verlag, Heidelberg, DE, 316–328. (pages 89, 91, 92, 107, and 108).

Baillie, M., Carman, M. J., and Crestani, F. 2009. A Topic-based Measure of Resource Description Quality for Distributed Information Retrieval. In *Proceedings of the European Conference on Information Retrieval Research (ECIR'09)*, Toulouse, FR. Springer-Verlag, Heidelberg, DE, 485–497. (page 119).

Balke, W.-T., Nejdl, W., Siberski, W., and Thaden, U. 2005. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In *Proceedings of the International Conference on Data Engineering (ICDE'05)*, Tokyo, JP. IEEE Computer Society, Los Alamitos, CA, US, 174–185. (pages 46 and 47).

Bar-Yossef, Z. and Gurevich, M. 2006. Random Sampling from a Search Engine's Index. In *Proceedings of the Conference on the World Wide Web (WWW'06)*, Edinburgh, UK. ACM Press, New York, NY, US, 367–376. (pages 92 and 98).

Bar-Yossef, Z. and Gurevich, M. 2008b. Random Sampling from a Search Engine's Index. *Journal of the ACM 55*, 5, 1–74. (pages 98 and 108).

Bawa, M., Manku, G. S., and Raghavan, P. 2003. SETS: Search Enhanced by Topic Segmentation. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'03)*, Toronto, ON, CA. ACM Press, New York, NY, US, 306–313. (pages 10, 42, 49, and 52).

Bender, M., Michel, S., Triantafillou, P., and Weikum, G. 2007. Design Alternatives for Large-scale Web Search: Alexander was Great, Aeneas a Pioneer, and Anakin has the Force. In *Proceedings of the Workshop on Large-Scale and Distributed Systems for Information Retrieval (LSDS-IR'07)*, Amsterdam, NL. 16–22. (page 52).

Bender, M., Michel, S., Triantafillou, P., Weikum, G., and Zimmer, C. 2005a. Improving Collection Selection with Overlap Awareness in P2P Search Engines. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'05)*, Salvador, BR. ACM Press, New York, NY, US, 67–74. (page 56).

Bender, M., Michel, S., Triantafillou, P., Weikum, G., and Zimmer, C. 2005b. MINERVA: Collaborative P2P Search. In *Proceedings of the Conference on Very Large Databases (VLDB'05)*, Trondheim, NO. 1263–1266. (pages 4, 32, 52, 55, and 180).

Bender, M., Michel, S., Triantafillou, P., Weikum, G., and Zimmer, C. 2006. P2P Content Search: Give the Web Back to the People. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'06)*, Santa Barbara, CA, US. (pages 52 and 55).

Bendersky, M., Croft, W. B., and Diao, Y. 2011. Quality-biased Ranking of Web Documents. In *Proceedings of the ACM Conference on Web Search and Data Mining (WSDM'11)*, Hong Kong, CN. ACM Press, New York, NY, USA, 95–104. (page 137).

Bergholz, A. and Chidlovskii, B. 2003. Using Query Probing to Identify Query Language Features on the Web. In *Proceedings of the SIGIR Workshop on Distributed Information Retrieval*, Toronto, ON, CA. 21–30. (page 93).

BERGMAN, M. K. 2001. The Deep Web: Surfacing Hidden Value. *Journal of Electronic Publishing 7*, 1. (pages 3 and 90).

BHARAMBE, A. R., HERLEY, C., AND PADMANABHAN, V. N. 2005. Analyzing and Improving BitTorrent Performance. Tech. Rep. MSR-TR-2005-03, Microsoft Research. Feb. (page 80).

BHATTACHARJEE, B., CHAWATHE, S., GOPALAKRISHNAN, V., KELEHER, P., AND SILAGHI, B. 2003. Efficient Peer-to-Peer Searches Using Result-Caching. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, US. 225–236. (page 159).

BLANCO, R., BORTNIKOV, E., JUNQUEIRA, F., LEMPEL, R., TELLOLI, L., AND ZARAGOZA, H. 2010. Caching Search Engine Results over Incremental Indices. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'10)*, Geneva, CH. ACM Press, New York, NY, US, 82–89. (page 176).

BLOOM, B. H. 1970. Space/time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM 13*, 7, 422–426. (page 44).

BRENES, D. J. AND GAYO-AVELLO, D. 2009. Stratified Analysis of AOL Query Log. *Information Sciences 179*, 12, 1844–1858. (page 160).

BUNTINE, W., ABERER, K., PODNAR, I., AND RAJMAN, M. 2005. Opportunities from Open Source Search. In *Proceedings of the Conference on Web Intelligence (WI'05)*, Compiègne, FR. 2–8. (pages 52 and 56).

BURAGOHAIN, C., AGRAWAL, D., AND SURI, S. 2003. A Game Theoretic Framework for Incentives in P2P Systems. In *Proceedings of the IEEE Conference on P2P Computing (P2P'03)*, Linköping, SE. IEEE Computer Society, Los Alamitos, CA, US, 48–56. (pages 26, 29, and 73).

BUSTAMANTE, F. AND QIAO, Y. 2004. Friendships that Last: Peer Lifespan and its Role in P2P Protocols. In *Proceedings of the International Workshop on Web Content Caching and Distribution (IWCW'03)*, Hawthorne, NY, US. Kluwer Academic Publishers, Norwell, MA, US, 233–246. (page 18).

Buyya, R., Stockinger, H., Giddy, J., and Abramson, D. 2001. Economic Models for Management of Resources in Peer-to-Peer and Grid Computing. In *Proceedings of the Symposium on The Convergence of Information Technologies and Communications (ITCom'01)*, Denver, CO, US. SPIE, Bellingham, WA, US, 13–25. (pages 26, 27, and 28).

Callan, J. 2000. Distributed Information Retrieval. In *Advances in Information Retrieval*. Kluwer Academic Publishers, 127–150. (pages 24, 34, 123, and 181).

Callan, J. and Connell, M. 2001. Query-based Sampling of Text Databases. *ACM Transactions on Information Systems 19*, 2, 97–130. (pages 86, 87, 91, 93, 94, 98, 107, and 112).

Callan, J., Connell, M., and Du, A. 1999. Automatic Discovery of Language Models for Text Databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD'99)*, Philadelphia, PA, US. ACM Press, New York, NY, US, 479–490. (pages 89, 91, 94, 98, 107, 124, and 150).

Callan, J., Lu, Z., and Croft, W. B. 1995. Searching Distributed Collections with Inference Networks. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'95)*, Seattle, WA, US. ACM Press, New York, NY, US, 21–28. (pages 35 and 124).

Chang, K. C.-C., He, B., Li, C., Patel, M., and Zhang, Z. 2004. Structured Databases on the Web: Observations and Implications. *ACM SIGMOD Record 33*, 3, 61–70. (page 91).

Chang, K. C.-C., He, B., and Zhang, Z. 2005. Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR'05)*, Asilomar, CA, US. 44–55. (page 93).

Chernov, S., Serdyukov, P., Bender, M., Michel, S., Weikum, G., and Zimmer, C. 2005. Database Selection and Result Merging in P2P Web Search. In *Proceedings of the Conference on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P'05)*, Trondheim, NO. Springer-Verlag, Heidelberg, DE, 26–37. (page 52).

Chidlovskii, B., Glance, N. S., and Grasso, M. A. 2000. Collaborative Re-Ranking of Search Results. In *Proceedings of AAAI-2000 Workshop on AI for Web Search*, Austin, TX, US. 18–23. (page 4).

Clarke, C., Craswell, N., and Soboroff, I. 2004. Overview of the TREC 2004 Terabyte Track. In *Proceedings of the Text REtrieval Conference (TREC'04)*. National Institute of Standards and Technology, Gaithersburg, MD, US. (pages 39 and 131).

Clarke, C., Craswell, N., and Soboroff, I. 2009. Overview of the TREC 2009 Web Track. In *Proceedings of the Text REtrieval Conference (TREC'09)*. National Institute of Standards and Technology, Gaithersburg, MD, US. (page 131).

Clarke, I., Sandberg, O., Wiley, B., and Hong, T. 2001. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proceedings of the Workshop on Designing Privacy Enhancing Technologies (PET'00)*, Berkeley, CA, US. Springer-Verlag, New York, NY, US, 46–66. (page 17).

Cohen, B. 2003. Incentives Build Robustness in BitTorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems (P2PEcon'03)*, Berkeley, CA, US. (pages 15, 70, 74, 79, 161, and 181).

Craswell, N. and Hawking, D. 2009. Web Information Retrieval. In *Information Retrieval: Searching in the 21st Century*, A. Göker and J. Davies, Eds. John Wiley & Sons, Ltd., Chichester, UK, 85–101. (page 62).

Crespo, A. and Garcia-Molina, H. 2004. Semantic Overlay Networks for P2P Systems. In *Proceedings of the International Workshop on Agents and Peer-to-Peer Computing (AP2PC'04)*, New York, NY, US. Springer-Verlag, Heidelberg, DE, 1–13. (pages 19 and 49).

Croft, W. B., Metzler, D., and Strohman, T. 2010. *Search Engines: Information Retrieval in Practice* 1st Ed. Pearson Education. ISBN 9780131364899. (pages 163 and 169).

Crosby, S. A. and Wallach, D. S. 2007. An Analysis of BitTorrent's Two Kademlia-based DHTs. Tech. Rep. TR-07-04, Department of Computer Science, Rice University, Houston, TX, US. June. (page 81).

Cuenca-Acuna, F. M., Martin, R. P., and Nguyen, T. D. 2003. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Proceedings of the IEEE International Symposium on High-Performance Distributed Computing (HPDC'03)*, Seattle, WA, US. IEEE Computer Society, Los Alamitos, CA, US, 236–249. (pages 18, 44, 46, 48, and 52).

Dagan, I., Lee, L., and Pereira, F. 1997. Similarity-based Methods for Word Sense Disambiguation. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL'97)*, Madrid, ES. Association for Computational Linguistics, Stroudsburg, PA, US, 56–63. (page 90).

Dang, V. and Croft, W. B. 2010. Query Reformulation using Anchor Text. In *Proceedings of the ACM Conference on Web Search and Data Mining (WSDM'10)*, New York, NY, USA. ACM Press, New York, NY, USA, 41–50. (page 127).

Daswani, N., Garcia-Molina, H., and Yang, B. 2003. Open Problems in Data-Sharing Peer-to-Peer Systems. In *Proceedings of the International Conference on Database Theory (ICDT'03)*, Siena, IT. Springer-Verlag, London, UK, 1–15. (pages 12, 15, and 24).

Datta, A., Hauswirth, M., John, R., Schmidt, R., and Aberer, K. 2005. Range Queries in Trie-structured Overlays. In *Proceedings of the IEEE Conference on P2P Computing (P2P'05)*, Konstanz, DE. IEEE Computer Society, Los Alamitos, CA, US, 57–66. (page 49).

Davis, M. D. 1983. *Game Theory: A Nontechnical Introduction* 2nd Ed. Dover Publications. ISBN 9780486296722. (pages 29 and 30).

Di Buccio, E., Masiero, I., and Melucci, M. 2009. Improving Information Retrieval Effectiveness in Peer-to-Peer Networks through Query Piggybacking. In *Proceedings of the European Conference on Digital Libraries (ECDL'09)*, Corfu, GR. Springer-Verlag, Heidelberg, DE, 420–424. (pages 32, 52, and 58).

Du, A. and Callan, J. 1998. Probing a Collection to Discover its Language Model. Tech. Rep. UM-CS-1998-029, University of Massachusetts, Amherst, MA, US. July. (pages 35 and 91).

EDELMAN, B. 2010. Hard-coding Bias in Google 'Algorithmic' Search Results. http://www.benedelman.org/hardcoding/ (Retrieved June 24th 2012). (page 3).

EIRON, N. AND MCCURLEY, K. S. 2003. Analysis of Anchor Text for Web Search. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'03)*, Toronto, ON, CA. ACM Press, New York, NY, US, 459–460. (page 116).

FAGIN, R., LOTEM, A., AND NAOR, M. 2001. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Symposium on Principles of Database Systems (PODS'01)*, Santa Barbara, CA, US. ACM Press, New York, NY, US, 102–113. (page 47).

FAGNI, T., PEREGO, R., SILVESTRI, F., AND ORLANDO, S. 2006. Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data. *ACM Transactions on Information Systems 24*, 1, 51–78. (pages 49 and 83).

FALKNER, J., PIATEK, M., JOHN, J. P., KRISHNAMURTHY, A., AND ANDERSON, T. 2007. Profiling a Million User DHT. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'07)*, San Diego, CA, US. ACM Press, New York, NY, US, 129–134. (page 81).

FALLEN, C. T. AND NEWBY, G. B. 2006. Partitioning the Gov2 Corpus by Internet Domain Name: A Result-set Merging Experiment. In *Proceedings of the Text REtrieval Conference (TREC'06)*. National Institute of Standards and Technology. (pages 39 and 137).

FELDMAN, M., LAI, K., STOICA, I., AND CHUANG, J. 2004. Robust Incentive Techniques for Peer-to-Peer Networks. In *Proceedings of the ACM Conference on Electronic Commerce (EC'04)*, New York, NY, US. ACM Press, New York, NY, US, 102–111. (pages 70 and 73).

FERGUSON, D. F., NIKOLAOU, C., SAIRAMESH, J., AND YEMINI, Y. 1996. Economic Models for Allocating Resources in Computer Systems. In *Market-based Control: a Paradigm for Distributed Resource Allocation*. World Scientific Publishing Co., Inc., River Edge, NJ, US, 156–183. (pages 26, 27, and 28).

FLYNN, S. M. 2005. *Economics for Dummies* 1st Ed. Pearson Education. ISBN 9789043011358 (Dutch Translation). (page 27).

FREYNE, J., SMYTH, B., COYLE, M., BALFE, E., AND BRIGGS, P. 2004. Further Experiments on Collaborative Ranking in Community-based Web Search. *Artificial Intelligence Review 21*, 3, 229–252. (page 4).

FRIEDMAN, E. J. AND RESNICK, P. 2001. The Social Cost of Cheap Pseudonyms. *Journal of Economics & Management Strategy 10*, 2, 173–199. (pages 70 and 73).

FUHR, N. 1999. A Decision-theoretic Approach to Database Selection in Networked IR. *ACM Transactions on Information Systems 17*, 3, 229–249. (page 52).

GALANIS, L., WANG, Y., JEFFERY, S., AND DEWITT, D. 2003. Processing Queries in a Large Peer-to-Peer System. In *Proceedings of the Conference on Advanced information Systems Engineering (CAiSE'03)*, Klagenfurt, AT. Springer-Verlag, Heidelberg, DE, 273–288. (pages 3, 48, and 58).

GIRDZIJAUSKAS, S., GALUBA, W., DARLAGIANNIS, V., DATTA, A., AND ABERER, K. 2011. Fuzzynet: Ringless Routing in a Ring-like Structured Overlay. *Peer-to-Peer Networking and Applications 4*, 3, 259–273. (page 22).

GOLDSCHLAG, D., REED, M., AND SYVERSON, P. 1999. Onion Routing. *Communications of the ACM 42*, 2, 39–41. (page 177).

GOLLE, P., LEYTON-BROWN, K., MIRONOV, I., AND LILLIBRIDGE, M. 2001. Incentives for Sharing in Peer-to-Peer Networks. In *Proceedings of the ACM Conference on Electronic Commerce (EC'01)*, Tampa, FL, US. ACM Press, New York, NY, US, 264–267. (page 29).

GRAVANO, L., CHANG, K. C.-C., GARCIA-MOLINA, H., LAGOZE, C., AND PAEPCKE, A. 1997. Stanford Protocol Proposal for Internet Retrieval and Search. Tech. rep., Stanford University. Jan. (pages 35 and 125).

GRAVANO, L., GARCIA-MOLINA, H., AND TOMASIC, A. 1999. GlOSS: Text-source Discovery over the Internet. *ACM Transactions on Database Systems 24*, 2, 229–264. (pages 35, 124, and 125).

GROTHOFF, C. 2003. An Excess-based Economic Model for Resource Allocation in Peer-to-Peer Networks. *Wirtschaftsinformatik 3*. (pages 68 and 73).

HARMAN, D. K. 1993. Overview of the First TREC Conference. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'93)*, Pittsburgh, PA, US. ACM Press, New York, NY, US, 36–47. (page 112).

HARMAN, D. K. 1994. Overview of the Third Text REtrieval Conference (TREC-3). In *Proceedings of the Third Text REtrieval Conference (TREC-3)*. Gaithersburg, MD, US, National Institute of Standards and Technology. (page 87).

HAWKING, D. AND THOMAS, P. 2005. Server Selection Methods in Hybrid Portal Search. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'05)*, Salvador, BR. ACM Press, New York, NY, US, 75–82. (page 131).

HAWKING, D., VOORHEES, E., CRASWELL, N., AND BAILEY, P. 2000. Overview of the TREC-8 Web Track. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, US. Nov. (page 87).

HE, B., PATEL, M., ZHANG, Z., AND CHANG, K. C.-C. 2007. Accessing the deep web. *Communications of the ACM 50*, 5, 94–101. (page 91).

HE, J., BALOG, K., HOFMANN, K., MEIJ, E., DE RIJKE, M., TSAGKIAS, M., AND WEERKAMP, W. 2009. Heuristic Ranking and Diversification of Web Documents. In *Proceedings of the Text REtrieval Conference (TREC'09)*, Gaithersburg, MD, US. National Institute of Standards and Technology. (page 137).

HERBERT, F. 1965. *Dune*. Chilton Books. Based on Dune World and The Prophet of Dune, ISBN 9780801950773.

HUEBSCH, R., CHUN, B., HELLERSTEIN, J. M., LOO, B. T., MANIATIS, P., ROSCOE, T., SHENKER, S., STOICA, I., AND YUMEREFENDI, A. R. 2005. The Architecture of PIER: an Internet-scale Query Processor. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR'05)*, Asilomar, CA, US. 28–43. (page 57).

IDE, N. AND SUDERMAN, K. 2010. The Open American National Corpus. `http://www.americannationalcorpus.org/` (Retrieved June 24th 2012). (page 87).

IPEIROTIS, P. G. AND GRAVANO, L. 2002. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. In *Proceedings of the Conference on Very Large Databases (VLDB'02)*, Hong Kong, CN. 394–405. (pages 92 and 107).

IPEIROTIS, P. G. AND GRAVANO, L. 2008. Classification-aware Hidden-web Text Database Selection. *ACM Transactions on Information Systems 26*, 2, 1–66. (pages 92 and 107).

JIAN, L. AND MACKIE-MASON, J. K. 2008. Why Share in Peer-to-Peer Networks? In *Proceedings of the International Conference on Electronic Commerce (ICEC'08)*, Innsbruck, AT. ACM Press, New York, NY, US, 4:1–4:8. (pages 72 and 73).

JORDAN, C., WATTERS, C., AND GAO, Q. 2006. Using Controlled Query Generation to Evaluate Blind Relevance Feedback Algorithms. In *Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'06)*, Chapel Hill, NC, US. ACM Press, New York, NY, US, 286–295. (pages 97 and 107).

JOSEPH, S. 2002. NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks. In *Proceedings of the Networking 2002 Workshops on Web Engineering and Peer-to-Peer Computing*, Pisa, IT. Springer-Verlag, London, UK, 202–214. (pages 14, 51, 52, and 58).

KALOGERAKI, V., GUNOPULOS, D., AND ZEINALIPOUR-YAZTI, D. 2002. A Local Search Mechanism for Peer-to-Peer Networks. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'02)*, McLean, VA, US. ACM Press, New York, NY, US, 300–307. (pages 51 and 52).

KAMVAR, S. D., SCHLOSSER, M. T., AND GARCIA-MOLINA, H. 2003. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the Conference on the World Wide Web (WWW'03)*, Budapest, HU. ACM Press, New York, NY, US, 640–651. (pages 13, 67, 68, and 73).

KARAKAYA, M., KÖRPEOĞLU, I., AND ULUSOY, Ö. 2007. Counteracting Free Riding in Peer-to-Peer Networks. *Computer Networks 52*, 3, 675–694. (pages 65, 72, and 73).

KEYANI, P., LARSON, B., AND SENTHIL, M. 2002. Peer Pressure: Distributed Recovery from Attacks in Peer-to-Peer Systems. In *Proceedings of the Networking 2002 Workshops on Web Engineering and Peer-to-Peer Computing*, Pisa, IT. Springer-Verlag, London, UK, 306–320. (page 13).

KILGARRIFF, A. AND ROSE, T. 1998. Measures for Corpus Similarity and Homogeneity. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'98)*, Granada, ES. Association for Computational Linguistics, Stroudsburg, PA, US, 46–52. (page 115).

KIRSCH, S. T. 1997. Document Retrieval over Networks wherein Ranking and Relevance Scores are Computed at the Client for Multiple Database Documents. United States Patent 5,659,732. (pages 36 and 136).

KLAMPANOS, I. A. AND JOSE, J. M. 2004. An Architecture for Information Retrieval over Semi-collaborating Peer-to-Peer Networks. In *Proceedings of the ACM Symposium on Applied Computing (SAC'04)*, Nicosia, CY. ACM Press, New York, NY, US, 1078–1083. (pages 25, 42, 49, 54, 57, and 123).

KLAMPANOS, I. A. AND JOSE, J. M. 2007. An Evaluation of a Cluster-based Architecture for Peer-to-Peer Information Retrieval. In *Proceedings of the International Conference on Database and Expert Systems (DEXA'07)*, Regensburg, DE. 380–391. (pages 49, 50, and 62).

KLAMPANOS, I. A., POZNAŃSKI, V., JOSE, J. M., AND DICKMAN, P. 2005. A suite of testbeds for the realistic evaluation of peer-to-peer information retrieval systems. In *Proceedings of the European Conference on Information Retrieval Research (ECIR'05)*, Santiago de Compostela, ES. 38–51. (pages 13, 38, 39, 62, and 131).

KLEINBERG, J. M. 2006. Complex Networks and Decentralized Search Algorithms. In *Proceedings of the International Congress of Mathematicians (ICM'06)*, Madrid, ES. 1019–1044. (page 22).

KRISHNAN, R., SMITH, M. D., TANG, Z., AND TELANG, R. 2002. The Virtual Commons: Why Free-riding can be Tolerated in File Sharing Networks. In *Proceedings of the International Conference on Information Systems (ICIS'02)*, Barcelona, ES. (pages 13, 71, and 73).

Krishnan, R., Smith, M. D., Tang, Z., and Telang, R. 2007. Digital Business Models for Peer-to-Peer Networks: Analysis and Economic Issues. *Review of Network Economics 6*, 2, 194–213. (pages 4 and 28).

Kulathuramaiyer, N. and Balke, W.-T. 2006. Restricting the View and Connecting the Dots - Dangers of a Web Search Engine Monopoly. *Journal of Universal Computer Science 12*, 12, 1731–1740. (page 3).

Kurose, J. F. and Ross, K. W. 2003. *Computer Networking: A Top-Down Approach Featuring the Internet* 2nd Ed. Addison-Wesley. ISBN 0201976994. (pages 9, 10, and 19).

Le Blond, S., Legout, A., Lefessant, F., Dabbous, W., and Kaafar, M. A. 2010. Spying the World from your Laptop: Identifying and Profiling Content Providers and Big Downloaders in BitTorrent. In *Proceedings of the USENIX Conference Large-scale Exploits and Emergent Threats (LEET'10)*, San Jose, CA, US. USENIX Association, Berkeley, CA, US. (page 177).

Lele, N., Wu, L.-S., Akavipat, R., and Menczer, F. 2009. Sixearch.org 2.0 Peer Application for Collaborative Web Search. In *Proceedings of the ACM Conference on Hypertext and Hypermedia (HT'09)*, Torino, IT. ACM Press, New York, NY, US, 333–334. (pages 49, 52, and 54).

Lempel, R. and Moran, S. 2003. Predictive Caching and Prefetching of Query Results in Search Engines. In *Proceedings of the Conference on the World Wide Web (WWW'03)*, Budapest, HU. ACM Press, New York, NY, US, 19–28. (pages 49 and 83).

Lesniewski-Laas, C. and Kaashoek, M. F. 2010. Whanau: A Sybil-proof Distributed Hash Table. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'10)*, San Jose, CA, US. USENIX Association, Berkeley, CA, US, 8–24. (page 18).

Lewandowski, D., Wahlig, H., and Meyer-Bautor, G. 2006. The Freshness of Web Search Engine Databases. *Journal of Information Science 32*, 2, 131–148. (page 3).

Leyton-Brown, K. and Shoham, Y. 2008. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction* eBook Ed. Morgan & Claypool. ISBN 978159829548. (pages 26 and 29).

Li, C., Yu, B., and Sycara, K. 2009. An Incentive Mechanism for Message Relaying in Unstructured Peer-to-Peer Systems. *Electronic Commerce Research and Applications 8*, 6, 315–326. (pages 51 and 63).

Li, J., Loo, B. T., Joseph, L., Hellerstein, J. M., Karger, D. R., Morris, R., and Kaashoek, M. F. 2003. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, US. 207–215. (pages 41, 42, and 43).

Liang, J., Kumar, R., and Ross, K. W. 2006. The FastTrack Overlay: A Measurement Study. *Computer Networks 50*, 6, 842–858. (page 20).

Loo, B. T., Huebsch, R., Stoica, I., and Hellerstein, J. M. 2004. The Case for a Hybrid P2P Search Infrastructure. In *Proceedings of Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'04)*, San Diego, CA, US. 141–150. (page 57).

Lu, J. 2007. Full-text Federated Search in Peer-to-Peer Networks. Ph.D. thesis, Carnegie Mellon University. CMU-LTI-07-003. (pages 19, 22, 32, 35, 36, 37, 39, 42, 131, 136, 158, and 177).

Lu, J. and Callan, J. 2003. Content-based Retrieval in Hybrid Peer-to-Peer Networks. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'02)*, New Orleans, LA, US. ACM Press, New York, NY, US, 199–206. (pages 39, 122, 125, 127, 131, 142, and 148).

Lu, J. and Callan, J. 2006a. Full-text Federated Search of Text-based Digital Libraries in Peer-to-Peer Networks. *Information Retrieval 9*, 4, 477–498. (pages 20, 32, and 158).

Lu, J. and Callan, J. 2007. Content-based Peer-to-Peer Network Overlay for Full-text Federated Search. In *Proceedings of Recherche d'Information Assistée par Ordinateur (RIAO'07)*, Pittsburgh, PA, US. 490–509. (page 49).

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. 2005. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *Communications Surveys & Tutorials 7*, 2, 72–93. (pages 17, 20, and 24).

Luu, T., Klemm, F., Podnar, I., Rajman, M., and Aberer, K. 2006. ALVIS Peers: A Scalable Full-text Peer-to-Peer Retrieval Engine. In *Proceedings of the Workshop on Information Retrieval in Peer-to-Peer Networks (P2PIR'06)*, Arlington, VA, US. ACM Press, New York, NY, US, 41–48. (pages 4, 46, 52, 56, and 180).

Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. 2002. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the International Conference on Supercomputing (ICS'02)*, New York, NY, US. ACM Press, New York, NY, US, 84–95. (pages 19, 51, and 125).

Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., and Halevy, A. Y. 2008. Google's Deep Web Crawl. *Proceedings of the VLDB Endowment 1*, 2, 1241–1252. (page 92).

Maier, G., Feldmann, A., Paxson, V., and Allman, M. 2009. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'09)*, Chicago, IL, US. ACM Press, New York, NY, US, 90–102. (page 74).

Manning, C. D., Raghavan, P., and Schütze, H. 2008. *Introduction to Information Retrieval* 1st Ed. Cambridge University Press, New York, NY, US. ISBN 9780521865715. (pages 89, 95, 108, and 110).

Manning, C. D. and Schütze, H. 1999. *Foundations of Statistical Natural Language Processing* 1st Ed. MIT Press, Cambridge, MA, US. ISBN 9780262133609. (page 89).

Markatos, E. P. 2001. On Caching Search Engine Query Results. *Computer Communications 24*, 2, 137–143. (pages 158, 166, 167, and 176).

Maymounkov, P. and Mazières, D. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA, US. 53–65. (pages 17 and 76).

McGraw-Hill. 2002. *McGraw-Hill Dictionary of Scientific and Technical Terms* 6th Ed. McGraw-Hill Professional. ISBN 9780070423138. (page 1).

McNamee, P. and Mayfield, J. 2004. Character N-Gram Tokenization for European Language Text Retrieval. *Information Retrieval 7*, 1, 73–97. (page 161).

Megiddo, N. and Modha, D. S. 2003. ARC: A Self-tuning, Low Overhead Replacement Cache. In *Proceedings of the USENIX Conference on File And Storage Technologies (FAST'03)*, San Francisco, CA, US. USENIX Association, Berkeley, CA, US, 115–130. (page 176).

Menczer, F., Wu, L.-S., and Akavipat, R. 2008. Intelligent Peer Networks for Collaborative Web Search. *Artificial Intelligence Magazine 29*, 3, 35–46. (pages 52 and 54).

Michel, S., Bender, M., Triantafillou, P., and Weikum, G. 2006. IQN Routing: Integrating Quality and Novelty in P2P Querying and Ranking. In *Proceedings of the International Conference on Extending Database Technology (EDBT'06)*, Munich, DE. 149–166. (pages 44, 45, 52, and 56).

Michel, S., Triantafillou, P., and Weikum, G. 2005a. KLEE: A Framework for Distributed Top-k Query Algorithms. In *Proceedings of the International Conference on Very Large Databases (VLDB'05)*, Trondheim, NO. 637–648. (pages 44, 46, and 47).

Michel, S., Triantafillou, P., and Weikum, G. 2005b. MINERVA infinity: A Scalable Efficient Peer-to-Peer Search Engine. In *Proceedings of the International Conference on Middleware'05*. Springer-Verlag, New York, NY, US, 60–81. (page 55).

Milojicic, D. S., Kalogeraki, V., Lukose, R. M., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. 2003. Peer-to-Peer Computing. Tech. rep., Hewlett-Packard Company. July. (page 12).

Monnerat, L. and Amorim, C. 2009. Peer-to-Peer Single Hop Distributed Hash Tables. In *Proceedings of the IEEE Conference on Global Telecommunications (GLOBECOM'09)*, Honolulu, HI, US. IEEE Press, Piscataway, NJ, US. (pages 17 and 22).

Monroe, G., French, J. C., and Powell, A. L. 2002. Obtaining Language Models of Web Collections using Query-based Sampling Techniques. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS'02)*, Big Island, HI, US. Vol. 3. IEEE Computer Society, Washington, DC, US, 1241–1247. (page 91).

Mowshowitz, A. and Kawaguchi, A. 2002. Assessing Bias in Search Engines. *Information Processing & Management 38*, 1, 141–156. (page 3).

Myerson, R. B. 1997. *Game Theory: Analysis of Conflict* 1st Ed. Harvard University Press. ISBN 9780674341166. (page 30).

Nah, F. F.-H. 2004. A Study on Tolerable Waiting Time: How Long are Web Users Willing to Wait? *Behaviour & Information Technology 23*, 3, 153–163. (page 37).

Naicken, S., Basu, A., Livingston, B., and Rodhetbhai, S. 2006. A Survey of Peer-to-Peer Network Simulators. In *Proceedings of the Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet'06)*, Liverpool, UK. (page 13).

Naicken, S., Livingston, B., Basu, A., Rodhetbhai, S., Wakeman, I., and Chalmers, D. 2007. The State of Peer-to-Peer Simulators and Simulations. *ACM SIGCOMM Computer Communication Review 37*, 2, 95–98. (page 13).

Neumann, T., Bender, M., Michel, S., Weikum, G., Bonnet, P., and Manolescu, I. 2006. A Reproducible Benchmark for P2P Retrieval. In *Proceedings of the Workshop on Performance and Evaluation of Data Management Systems (EXPDB'06)*, Chicago, IL, US. ACM Press, New York, NY, US. (page 38).

Ngan, T.-W. J., Wallach, D. S., and Druschel, P. 2003. Enforcing Fair Sharing of Peer-to-Peer Resources. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, US. 149–159. (pages 67 and 73).

Nguyen, L. T., Yee, W. G., and Frieder, O. 2008. Adaptive Distributed Indexing for Structured Peer-to-Peer Networks. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'08)*, Napa Valley, CA, US. ACM Press, New York, NY, US, 1241–1250. (page 49).

Nottelmann, H. and Fuhr, N. 2007. A Decision-theoretic Model for Decentralised Query Routing in Hierarchical Peer-to-Peer Networks. In *Proceedings of the European Conference on Information Retrieval Research (ECIR'07)*, Rome, IT. 148–159. (pages 35, 52, and 124).

NTOULAS, A., ZERFOS, P., AND CHO, J. 2005. Downloading Textual Hidden Web Content through Keyword Queries. In *Proceedings of the ACM/IEEE-CS Joint Conference on Digital libraries (JCDL'05)*, Denver, CO, US. ACM Press, New York, NY, US, 100–109. (page 92).

ORAM, A. 2001. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies* 1st Ed. O'Reilly Media. ISBN 9780596001100. (page 4).

OULASVIRTA, A., HUKKINEN, J. P., AND SCHWARTZ, B. 2009. When More is Less: The Paradox of Choice in Search Engine Use. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'09)*, Boston, MA, US. ACM Press, New York, NY, US, 516–523. (pages 27 and 46).

PALTOGLOU, G., SALAMPASIS, M., AND SATRATZEMI, M. 2007. Hybrid Results Merging. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'07)*, Lisbon, PT. ACM Press, New York, NY, US, 321–330. (page 109).

PAPAPETROU, O., SIBERSKI, W., AND FUHR, N. 2010. Text Clustering for Peer-to-Peer Networks with Probabilistic Guarantees. In *Proceedings of the European Conference on Information Retrieval Research (ECIR'10)*, Milton Keynes, UK. Springer-Verlag, Heidelberg, DE, 293–305. (page 49).

PASS, G., CHOWDHURY, A., AND TORGESON, C. 2006. A Picture of Search. In *Proceedings of the Conference on Scalable Information Systems (InfoScale'06)*, Hong Kong, CN. ACM Press, New York, NY, US, 1–7. (pages 160 and 161).

PIATEK, M., ISDAL, T., ANDERSON, T., KRISHNAMURTHY, A., AND VENKATARAMANI, A. 2007. Do Incentives Build Robustness in BitTorrent? In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'07)*, Cambridge, MA, US. USENIX Association, Berkeley, CA, US. (page 80).

PODLIPNIG, S. AND BÖSZÖRMENYI, L. 2003. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys 35*, 4, 374–398. (pages 165 and 176).

PORTER, M. F. 2001. The English (Porter2) Stemming Algorithm. `http://snowball.tartarus.org/algorithms/english/stemmer.html` (Retrieved January 13th 2011). (page 167).

Pouwelse, J. A., Garbacki, P., Epema, D. H. J., and Sips, H. J. 2005. The BitTorrent P2P File-Sharing System: Measurements and Analysis. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'05)*, Ithaca, NY, US. 205–216. (pages 74 and 79).

Pouwelse, J. A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D. H. J., Reinders, M., van Steen, M. R., and Sips, H. J. 2008a. Tribler: A Social-based Peer-to-Peer System. *Concurrency and Computation: Practice and Experience 20*, 2, 127–138. (pages 82 and 172).

Pouwelse, J. A., Yang, J., Meulpolder, M., Epema, D. H. J., and Sips, H. J. 2008b. BuddyCast: An Operational Peer-to-Peer Epidemic Protocol Stack. Tech. Rep. PDS-2008-005, Delft University of Technology. (page 82).

Qiu, D. and Srikant, R. 2004. Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks. In *Proceedings of the Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM'04)*, Portland, OR, US. ACM Press, New York, NY, US, 367–378. (page 80).

Ranganathan, K., Ripeanu, M., Sarin, A., and Foster, I. 2003. To Share or not to Share: An Analysis of Incentives to Contribute in Collaborative File Sharing Environments. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems (P2PEcon'03)*, Berkeley, CA, US. (pages 66 and 73).

Reynolds, P. and Vahdat, A. 2003. Efficient Peer-to-Peer Keyword Searching. In *Proceedings of the International Conference on Middleware'03*, Rio de Janeiro, BR. Springer-Verlag, New York, NY, US, 21–40. (pages 32, 33, 44, 45, and 49).

Risson, J. and Moors, T. 2006. Survey of Research Towards Robust Peer-to-Peer Networks: Search Methods. *Computer Networks 50*, 17, 3485–3521. (pages 10, 16, 19, 22, and 25).

Risvik, K. M. and Michelsen, R. 2002. Search Engines and Web Dynamics. *Computer Networks 39*, 3, 289–302. (page 38).

Rosenfeld, A., Goldman, C. V., Kaminka, G. A., and Kraus, S. 2009. PHIRST: A Distributed Architecture for P2P Information Retrieval. *Information Systems 34*, 2, 290–303. (pages 25, 43, 52, 57, and 181).

Senellart, P., Mittal, A., Muschick, D., Gilleron, R., and Tommasi, M. 2008. Automatic Wrapper Induction from Hidden-web Sources with Domain Knowledge. In *Proceedings of the Workshop on Web Information and Data Management (WIDM'08)*, Napa Valley, CA, US. ACM Press, New York, NY, US, 9–16. (page 107).

Shneidman, J. and Parkes, D. 2003. Rationality and Self-Interest in Peer to Peer Networks. In *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, US. 139–148. (pages 29, 30, and 31).

Shokouhi, M. 2007. Central-Rank-based Collection Selection in Uncooperative Distributed Information Retrieval. In *Proceedings of the European Conference on Information Retrieval Research (ECIR'07)*, Rome, IT. 160–172. (pages 35, 124, 128, 129, and 150).

Shokouhi, M. and Si, L. 2011. Federated Search. *Foundations and Trends in Information Retrieval 5*, 1, 1–102. (pages 128 and 129).

Shokouhi, M. and Zobel, J. 2007. Federated Text Retrieval from Uncooperative Overlapped Collections. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'07)*, Amsterdam, NL. ACM Press, New York, NY, US, 495–502. (page 35).

Shokouhi, M., Zobel, J., Scholer, F., and Tahaghoghi, S. M. M. 2006b. Capturing Collection Size for Distributed Non-cooperative Retrieval. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'06)*, Seattle, WA, US. ACM Press, New York, NY, US, 316–323. (pages 124 and 125).

Shokouhi, M., Zobel, J., Tahaghoghi, S. M. M., and Scholer, F. 2007b. Using Query Logs to Establish Vocabularies in Distributed Information Retrieval. *Information Processing & Management 43*, 1, 169–180. (page 56).

Si, L. and Callan, J. 2003a. Relevant Document Distribution Estimation Method for Resource Selection. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'03)*, Toronto, ON, CA. ACM Press, New York, NY, US, 298–305. (pages 35, 124, 128, 139, and 150).

Si, L. and Callan, J. 2003b. A Semisupervised Learning Method to Merge Search Engine Results. *ACM Transactions on Information Systems 21*, 4, 457–491. (page 36).

Skobeltsyn, G. and Aberer, K. 2006. Distributed Cache Table: Efficient Query-driven Processing of Multi-term Queries in P2P Networks. In *Proceedings of the Workshop on Information Retrieval in Peer-to-Peer Networks (P2PIR'06)*, Arlington, VA, US. ACM Press, New York, NY, US, 33–40. (pages 46, 48, 49, 52, 158, 159, 162, and 167).

Skobeltsyn, G., Luu, T., Aberer, K., Rajman, M., and Podnar Žarko, I. 2007a. Query-driven Indexing for Peer-to-Peer Text Retrieval. In *Proceedings of the Conference on the World Wide Web (WWW'07)*, Banff, AB, CA. ACM Press, New York, NY, US, 1185–1186. (pages 46, 49, 52, and 56).

Skobeltsyn, G., Luu, T., Podnar Žarko, I., Rajman, M., and Aberer, K. 2007b. Web Text Retrieval with a P2P Query-driven Index. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'07)*, Amsterdam, NL. ACM Press, New York, NY, US, 679–686. (page 49).

Skobeltsyn, G., Luu, T., Podnar Žarko, I., Rajman, M., and Aberer, K. 2009. Query-driven Indexing for Scalable Peer-to-Peer Text Retrieval. *Future Generation Computer Systems 25*, 1, 89–99. (pages 46, 49, 52, and 56).

Song, W., Zeng, X., Hu, W., Chen, Y., Wang, C., and Cheng, F. 2010. Resource Search in Peer-to-Peer Network Based on Power Law Distribution. In *Proceedings of the Conference on Networks Security, Wireless Communications and Trusted Computing (NSWCTC'10)*. IEEE Computer Society, Washington, DC, US, 53–56. (page 51).

Sripanidkulchai, K., Maggs, B. M., and Zhang, H. 2003. Efficient Content Location using Interest-based Locality in Peer-to-Peer Systems. In *Proceedings of the IEEE International Conference on Computer Communications (InfoCom'03)*, San Francisco, CA, US. 2166–2176. (pages 19, 49, and 51).

Steiner, M., En-Najjary, T., and Biersack, E. W. 2007. Exploiting KAD: Possible Uses and Misuses. *ACM SIGCOMM Computer Communication Review 37*, 5, 65–70. (pages 18 and 42).

Stoica, I., Morris, R., Karger, D. R., Kaashoek, M. F., and Balakrishnan, H. 2001. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *ACM SIGCOMM Computer Communication Review 31*, 4, 149–160. (pages 17 and 20).

Stokes, M. 2002. Gnutella2 Specifications: Part I. `http://g2.trillinux.org/index.php?title=G2_specs_part1` (Retrieved June 18th 2012). (pages 15 and 20).

Stutzbach, D. and Rejaie, R. 2006. Understanding Churn in Peer-to-Peer Networks. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'06)*, Rio de Janeiro, BR. ACM Press, New York, NY, US, 189–202. (pages 13, 158, and 172).

Suel, T., Mathur, C., Wu, J.-w., Zhang, J., Delis, A., Kharrazi, M., Long, X., and Shanmugasundaram, K. 2003. ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval. In *Proceedings of the International Workshop on the Web and Databases (WebDB'03)*, San Diego, CA, US. 67–72. (pages 4, 25, 32, 42, 44, 46, 47, 49, 52, and 54).

Sun, Q. and Garcia-Molina, H. 2004. SLIC: A Selfish Link-based Incentive Mechanism for Unstructured Peer-to-Peer Networks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, Hachioji, Tokyo, JP. IEEE Computer Society, Los Alamitos, CA, US, 506–515. (pages 64 and 72).

Suryanarayana, G. and Taylor, R. N. 2004. A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications. Tech. Rep. UCI-ISR-04-6, Institute for Software Research. July. (page 73).

Tang, C. and Dwarkadas, S. 2004. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'04)*, San Francisco, CA, US. USENIX Association, Berkeley, CA, US. (pages 46, 48, and 52).

TANG, C., XU, Z., AND MAHALINGAM, M. 2002. pSearch: Information Retrieval in Structured Overlays. In *Proceedings of the Workshop on Hot Topics in Networks (HotNets-I'02)*, Princeton, NY, US. (pages 46 and 52).

TEEVAN, J., ADAR, E., JONES, R., AND POTTS, M. A. S. 2007. Information Re-retrieval: Repeat Queries in Yahoo's Logs. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'07)*, Amsterdam, NL. ACM Press, New York, NY, US, 151–158. (page 163).

TENE, O. 2008. What Google Knows: Privacy and Internet Search Engines. *Utah Law Review 2008*, 4, 1434–1490. (page 3).

THOMAS, P. AND HAWKING, D. 2007. Evaluating Sampling Methods for Uncooperative Collections. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'07)*, Amsterdam, NL. ACM Press, New York, NY, US, 503–510. (pages 39, 108, and 137).

TIGELAAR, A. S. AND HIEMSTRA, D. 2009. Query-based Sampling using Only Snippets. Tech. Rep. TR-CTIT-09-42, Centre for Telematics and Information Technology, University of Twente, Enschede, NL. Nov. (pages 85 and 181).

TIGELAAR, A. S. AND HIEMSTRA, D. 2010a. Query-based Sampling: Can we do Better than Random? Tech. Rep. TR-CTIT-10-04, Centre for Telematics and Information Technology, University of Twente, Enschede, NL. Feb. (page 85).

TIGELAAR, A. S. AND HIEMSTRA, D. 2010b. Query-based Sampling using Snippets. In *Proceedings of the Workshop on Large-Scale and Distributed Systems for Information Retrieval (LSDS-IR'10)*, Geneva, CH. CEUR-WS, Aachen, DE, 9–14. (pages 35, 85, and 181).

TIGELAAR, A. S. AND HIEMSTRA, D. 2011. Query Load Balancing by Caching Search Results in Peer-to-Peer Information Retrieval Networks. In *Proceedings of the Dutch-Belgian Information Retrieval Workshop (DIR'11)*, Amsterdam, NL. 28–31. (pages 49 and 182).

TIGELAAR, A. S., HIEMSTRA, D., AND TRIESCHNIGG, D. 2011. Search Result Caching in Peer-to-Peer Information Retrieval Networks. In *Proceedings of the Information Retrieval Facility Conference (IRFC'11)*, Vienna, AT. Springer-Verlag, Heidelberg, DE, 134–138. (pages 49, 157, and 182).

TIGELAAR, A. S., HIEMSTRA, D., AND TRIESCHNIGG, D. 2012. Peer-to-Peer Information Retrieval: An Overview. *ACM Transactions on Information Systems 30*, 2, 9:1–9:34. (pages 9, 41, 180, and 181).

TIRADO, J. M., HIGUERO, D., ISAILA, F., CARRETERO, J., AND IAMNITCHI, A. 2010. Affinity P2P: A Self-organizing Content-based Locality-aware Collaborative Peer-to-Peer Network. *Computer Networks 54*, 12, 2056–2070. (page 49).

TRIANTAFILLOU, P., XIRUHAKI, C., KOUBARAKIS, M., AND NTARMOS, N. 2003. Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR'03)*, Asilomar, CA, US. 120–132. (pages 12, 42, 59, and 61).

TSOUMAKOS, D. AND ROUSSOPOULOS, N. 2003. Adaptive Probabilistic Search for Peer-to-Peer Networks. In *Proceedings of the IEEE Conference on P2P Computing (P2P'03)*, Linköping, SE. IEEE Computer Society, Washington, DC, US, 102–110. (page 51).

VAN HEERDE, H. J. W. 2010. Privacy-aware Data Management by means of Data Degradation - Making Private Data Less Sensitive over Time. Ph.D. thesis, Centre for Telematics and Information Technology, University of Twente, Enschede, NL. ISBN 9789036530026. (page 44).

VENABLES, W. N. AND SMITH, D. M. 2012. *An Introduction to R*. `http://cran.r-project.org/doc/manuals/R-intro.html` (Retrieved June 25th 2012). (page 88).

VISHNUMURTHY, V., CHANDRAKUMAR, S., AND SIRER, E. G. 2003. KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems (P2PEcon'03)*, Berkeley, CA, US. Department of Heuristics And Research on Material Applications. (pages 66 and 73).

WANG, W. AND LI, B. 2003. To Play or to Control: A Game-based Control-theoretic Approach to Peer-to-Peer Incentive Engineering. In *Proceedings of the International Workshop on Quality of service (IWQoS'03)*, Monterey, CA, US. Springer-Verlag, Heidelberg, DE, 174–192. (page 73).

WATERHOUSE, S., DOOLIN, D. M., KAN, G., AND FAYBISHENKO, Y. 2002. Distributed Search in P2P Networks. *IEEE Internet Computing 6*, 1, 68–72. (page 54).

WHITE, A. 2009. Search Engines: Left Side Quality versus Right Side Profits. Working paper, Toulouse School of Economics. Dec. `http://dx.doi.org/10.2139/ssrn.1694869` (Retrieved June 25th 2012). (page 3).

WITSCHEL, H. F. 2008. Ranking Information Resources in Peer-to-Peer Text Retrieval: An Experimental Study. In *Proceedings of the Workshop on Large-Scale and Distributed Systems for Information Retrieval (LSDS-IR'08)*, Napa Valley, CA, US. ACM Press, New York, NY, US, 75–82. (pages 122 and 138).

WITTEN, I. H., MOFFAT, A., AND BELL, T. C. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images* 2nd Ed. Morgan Kaufmann. ISBN 9781558605701. (page 57).

XU, J. AND CROFT, W. B. 1999. Cluster-based Language Models for Distributed Retrieval. In *Proceedings of the ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'99)*, Berkeley, CA, US. ACM Press, New York, NY, US, 254–261. (pages 35 and 124).

YANG, B., CONDIE, T., KAMVAR, S. D., AND GARCIA-MOLINA, H. 2005. Non-Cooperation in Competitive P2P Networks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, Columbus, OH, US. IEEE Computer Society, Washington, DC, US, 91–100. (pages 64 and 72).

YANG, B. AND GARCIA-MOLINA, H. 2002. Improving Search in Peer-to-Peer Networks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, AT. IEEE Computer Society, Washington, DC, US, 5–14. (pages 51 and 52).

YANG, Y., DUNLAP, R., REXROAD, M., AND COOPER, B. F. 2006. Performance of Full Text Search in Structured and Unstructured Peer-to-Peer Systems. In *Proceedings of the IEEE International Conference on Computer Communications (InfoCom'06)*, Barcelona, ES. 2658–2669. (pages 20, 22, 43, and 51).

YUWONO, B. AND LEE, D. L. 1997. Server Ranking for Distributed Text Retrieval Systems on the Internet. In *Proceedings of the Conference on Database Systems For Advanced Applications (DASFAA'97)*, Melbourne, AU. World Scientific Press, 41–50. (pages 35 and 124).

ZEINALIPOUR-YAZTI, D., KALOGERAKI, V., AND GUNOPULOS, D. 2004. Information Retrieval Techniques for Peer-to-Peer Networks. *Computing in Science & Engineering 6*, 4, 20–26. (pages 2, 19, 24, 31, 38, 41, 44, 48, 51, 52, and 61).

ZHANG, J. AND SUEL, T. 2005. Efficient Query Evaluation on Large Textual Collections in a Peer-to-Peer Environment. In *Proceedings of the IEEE Conference on P2P Computing (P2P'05)*, Konstanz, DE. IEEE Computer Society, Los Alamitos, CA, US, 225–233. (pages 44, 46, 47, and 52).

ZHONG, S., CHEN, J., AND YANG, Y. R. 2003. Sprite: A Simple, Cheat-proof, Credit-based System for Mobile Ad-hoc Networks. In *Proceedings of the IEEE International Conference on Computer Communications (InfoCom'03)*, San Francisco, CA, US. 1987–1997. (pages 51, 63, 72, and 73).

ZIMMER, C., BEDATHUR, S., AND WEIKUM, G. 2008. Flood Little, Cache More: Effective Result-reuse in P2P IR Systems. In *Proceedings of the Conference on Database Systems For Advanced Applications (DASFAA'08)*, New Delhi, IN. Springer-Verlag, Berlin, DE, 235–250. (pages 49, 52, 56, 162, and 167).

LIST OF PUBLICATIONS

Refereed

Tigelaar, A. S. and Hiemstra, D. 2010. Query-based Sampling using Snippets. In *Proceedings of the Workshop on Large-Scale and Distributed Systems for Information Retrieval (LSDS-IR'10)*, Geneva, CH. CEUR-WS, Aachen, DE, 9–14.

Tigelaar, A. S. and Hiemstra, D. 2011. Query Load Balancing by Caching Search Results in Peer-to-Peer Information Retrieval Networks. In *Proceedings of the Dutch-Belgian Information Retrieval Workshop (DIR'11)*, Amsterdam, NL. 28–31.

Tigelaar, A. S., Hiemstra, D., and Trieschnigg, D. 2011. Search Result Caching in Peer-to-Peer Information Retrieval Networks. In *Proceedings of the Information Retrieval Facility Conference (IRFC'11)*, Vienna, AT. Springer-Verlag, Heidelberg, DE, 134–138.

Tigelaar, A. S., Hiemstra, D., and Trieschnigg, D. 2012. Peer-to-Peer Information Retrieval: An Overview. *ACM Transactions on Information Systems 30*, 2, 9:1–9:34.

Non-Refereed

Tigelaar, A. S. and Hiemstra, D. 2009. Query-based Sampling using Only Snippets. Tech. Rep. TR-CTIT-09-42, Centre for Telematics and Information Technology, University of Twente, Enschede, NL.

Tigelaar, A. S. and Hiemstra, D. 2010. Query-based Sampling: Can we do Better than Random? Tech. Rep. TR-CTIT-10-04, Centre for Telematics and Information Technology, University of Twente, Enschede, NL.

SIKS DISSERTATIONS

Since 1998, all dissertations written by PhD students who have conducted their research under the auspices of a senior research fellow of the SIKS research school are published in the SIKS Dissertation Series.

2012-28 Nancy Pascall (UvT) *Engendering Technology Empowering Women*

2012-27 Hayrettin Gürkök (UT) *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games*

2012-26 Emile de Maat (UvA) *Making Sense of Legal Texts*

2012-24 Laurens van der Werff (UT) *Evaluation of Noisy Transcripts for Spoken Document Retrieval*

2012-23 Christian Muehl (UT) *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction*

2012-22 Thijs Vis (UvT) *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?*

2012-21 Roberto Cornacchia (TUD) *Querying Sparse Matrices for Information Retrieval*

2012-20 Ali Bahramisharif (RUN) *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing*

2012-19 Helen Schonenberg (TUE) *What's Next? Operational Support for Business Process Execution*

2012-18 Eltjo Poort (VU) *Improving Solution Architecting Practices*

2012-17 Amal Elgammal (UvT) *Towards a Comprehensive Framework for Business Process Compliance*

2012-16 Fiemke Both (VU) *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment*

2012-15 Natalie van der Wal (VU) *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes*

2012-14 Evgeny Knutov (TUE) *Generic Adaptation Framework for Unifying Adaptive Web-based Systems*

2012-13 Suleman Shahid (UvT) *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions*

2012-12 Kees van der Sluijs (TUE) *Model Driven Design and Data Integration in Semantic Web Information Systems*

2012-11 J.C.B. Rantham Prabhakara (TUE) *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*

2012-10 David Smits (TUE) *Towards a Generic Distributed Adaptive Hypermedia Environment*

2012-09 Ricardo Neisse (UT) *Trust and Privacy Management Support for Context-Aware Service Platforms*

2012-08 Gerben de Vries (UVA) *Kernel Methods for Vessel Trajectories*

2012-07 Rianne van Lambalgen (VU) *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions*

2012-06 Wolfgang Reinhardt (OU) *Awareness Support for Knowledge Workers in Research Networks*

2012-04 Jurriaan Souer (UU) *Development of Content Management System-based Web Applications 2012-05 Marijn Plomp (UU) Maturing Interorganisational Information Systems*

2012-03 Adam Vanya (VU) *Supporting Architecture Evolution by Mining Software Repositories*

2012-02 Muhammad Umair (VU) *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models*

2012-01 Terry Kakeeto (UvT) *Relationship Marketing for SMEs in Uganda*

2011-49 Andreea Niculescu (UT) *Conversational Interfaces for Task-oriented Spoken Dialogues: design aspects influencing interaction quality*

2011-48 Mark Ter Maat (UT) *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent*

2011-47 Azizi Bin Ab Aziz (VU) *Exploring Computational Models for Intelligent Support of Persons with Depression*

2011-46 Beibei Hu (TUD) *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work*

2011-45 Herman Stehouwer (UvT) *Statistical Language Models for Alternative Sequence Selection*

2011-44 Boris Reuderink (UT) *Robust Brain-Computer Interfaces*

2011-43 Henk van der Schuur (UU) *Process Improvement through Software Operation Knowledge*

2011-42 Michal Sindlar (UU) *Explaining Behavior through Mental State Attribution*

2011-41 Luan Ibraimi (UT) *Cryptographically Enforced Distributed Data Access Control*

2011-40 Viktor Clerc (VU) *Architectural Knowledge Management in Global Software Development*

2011-39 Joost Westra (UU) *Organizing Adaptation using Agents in Serious Games*

2011-38 Nyree Lemmens (UM) *Bee-inspired Distributed Optimization*

2011-37 Adriana Burlutiu (RUN) *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference*

2011-36 Erik van der Spek (UU) *Experiments in serious game design: a cognitive approach*

2011-35 Maaike Harbers (UU) *Explaining Agent Behavior in Virtual Training*

2011-34 Paolo Turrini (UU) *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations*

2011-33 Tom van der Weide (UU) *Arguing to Motivate Decisions*

2011-32 Nees-Jan van Eck (EUR) *Methodological Advances in Bibliometric Mapping of Science*

2011-31 Ludo Waltman (EUR) *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*

2011-30 Egon van den Broek (UT) *Affective Signal Processing (ASP): Unraveling the mystery of emotions*

2011-29 Faisal Kamiran (TUE) *Discrimination-aware Classification*

2011-28 Rianne Kaptein (UVA) *Effective Focused Retrieval by Exploiting Query Context and Document Structure*

2011-27 Aniel Bhulai (VU) *Dynamic website optimization through autonomous management of design patterns*

2011-26 Matthijs Aart Pontier (VU) *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots*

2011-25 Syed Waqar ul Qounain Jaffry (VU) *Analysis and Validation of Models for Trust Dynamics*

2011-24 Herwin van Welbergen (UT) *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior*

2011-23 Wouter Weerkamp (UVA) *Finding People and their Utterances in Social Media*

2011-22 Junte Zhang (UVA) *System Evaluation of Archival Description and Access*

2011-21 Linda Terlouw (TUD) *Modularization and Specification of Service-Oriented Systems*

2011-20 Qing Gu (VU) *Guiding service-oriented software engineering - A view-based approach*

2011-19 Ellen Rusman (OU) *The Mind's Eye on Personal Profiles*

2011-18 Mark Ponsen (UM) *Strategic Decision-Making in Complex Games*

2011-17 Jiyin He (UVA) *Exploring Topic Structure: Coherence, Diversity and Relatedness*

2011-16 Maarten Schadd (UM) *Selective Search in Games of Different Complexity*

2011-15 Marijn Koolen (UvA) *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*

2011-14 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*

2011-13 Xiaoyu Mao (UvT) *Airport under Control. Multiagent Scheduling for Airport Ground Handling*

2011-12 Carmen Bratosin (TUE) *Grid Architecture for Distributed Process Mining*

2011-11 Dhaval Vyas (UT) *Designing for Awareness: An Experience-focused HCI Perspective*

2011-10 Bart Bogaert (UvT) *Cloud Content Contention*

2011-09 Tim de Jong (OU) *Contextualised Mobile Media for Learning*

2011-08 Nieske Vergunst (UU) *BDI-based Generation of Robust Task-Oriented Dialogues*

2011-07 Yujia Cao (UT) *Multimodal Information Presentation for High Load Human Computer Interaction*

2011-06 Yiwen Wang (TUE) *Semantically-Enhanced Recommendations in Cultural Heritage*

2011-05 Base van der Raadt (VU) *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline*

2011-04 Hado van Hasselt (UU) *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms*

2011-03 Jan Martijn van der Werf (TUE) *Compositional Design and Verification of Component-Based Information Systems*

2011-02 Nick Tinnemeier (UU) *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language*

2011-01 Botond Cseke (RUN) *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*

2010-54 Riham Abdel Kader (UT) *ROX: Runtime Optimization of XQueries*

2010-53 Edgar Meij (UVA) *Combining Concepts and Language Models for Information Access*

2010-52 Peter-Paul van Maanen (VU) *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention*

2010-51 Alia Khairia Amin (CWI) *Understanding and Supporting Information Seeking Tasks in Multiple Sources*

2010-50 Bouke Huurnink (UVA) *Search in Audiovisual Broadcast Archives*

2010-49 Jahn-Takeshi Saito (UM) *Solving Difficult Game Positions*

1962-01 Philip K. Dick (UCB) *The Man in the High Castle*

2010-47 Chen Li (UT) *Mining Process Model Variants: Challenges, Techniques, Examples*

2010-46 Vincent Pijpers (VU) *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*

2010-45 Vasilios Andrikopoulos (UvT) *A Theory and Model for the Evolution of Software Services*

2010-44 Pieter Bellekens (TUE) *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*

2010-43 Peter van Kranenburg (UU) *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*

2010-42 Sybren de Kinderen (VU) *Needs-driven Service Bundling in a Multi-supplier Setting - the computational e3-service approach*

2010-41 Guillaume Chaslot (UM) *Monte-Carlo Tree Search*

2010-40 Mark van Assem (VU) *Converting and Integrating Vocabularies for the Semantic Web*

2010-39 Ghazanfar Farooq Siddiqui (VU) *Integrative Modeling of Emotions in Virtual Agents*

2010-38 Dirk Fahland (TUE) *From Scenarios to Components*

2010-37 Niels Lohmann (TUE) *Correctness of Services and their Composition*

2010-36 Jose Janssen (OU) *Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification*

2010-35 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*

2010-34 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*

2010-33 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*

2010-32 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*

2010-31 Victor de Boer (UVA) *Ontology Enrichment from Heterogeneous Sources on the Web*

2010-30 Marieke van Erp (UvT) *Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval*

2010-29 Stratos Idreos (CWI) *Database Cracking: Towards Auto-tuning Database Kernels*

2010-28 Arne Koopman (UU) *Characteristic Relational Patterns*

2010-27 Marten Voulon (UL) *Automatisch Contracteren*

2010-26 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*

2010-25 Zulfiqar Ali Memon (VU) *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*

2010-24 Dmytro Tykhonov (TUD) *Designing Generic and Efficient Negotiation Strategies*

2010-23 Bas Steunebrink (UU) *The Logical Structure of Emotions*

2010-22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*

2010-21 Harold van Heerde (UT) *Privacy-aware Data Management by means of Data Degradation*

2010-20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*

2010-19 Henriette Cramer (UvA) *People's Responses to Autonomous and Adaptive Systems*

2010-18 Charlotte Gerritsen (VU) *Caught in the Act: Investigating Crime by Agent-Based Simulation*
2010-17 Spyros Kotoulas (VU) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*
2010-16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*
2010-15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*
2010-14 Sander van Splunter (VU) *Automated Web Service Reconfiguration*
2010-13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*
2010-12 Susan van den Braak (UU) *Sensemaking Software for Crime Analysis*
2010-11 Adriaan ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*
2010-10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*
2010-09 Hugo Kielman (UL) *A Politiële gegevensverwerking en Privacy, Naar een effectieve waarborging*
2010-08 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*
2010-07 Wim Fikkert (UT) *Gesture interaction at a Distance*
2010-06 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*
2010-05 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*
2010-04 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*
2010-03 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*
2010-02 Ingo Wassink (UT) *Work flows in Life Science*
2010-01 Matthijs van Leeuwen (UU) *Patterns that Matter*
2009-46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*
2009-45 Jilles Vreeken (UU) *Making Pattern Mining Useful*
2009-44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*
2009-43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*
2009-42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*
2009-41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*
2009-40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*
2009-39 Christian Stahl (TUE, Humboldt-Universität zu Berlin) *Service Substitution – A Behavioral Approach Based on Petri Nets*
2009-38 Riina Vuorikari (OU) *Tags and Self-organisation: a metadata ecology for learning resources in a multilingual context*
2009-37 Hendrik Drachsler (OUN) *Navigation Support for Learners in Informal Learning Networks*
2009-36 Marco Kalz (OUN) *Placement Support for Learners in Learning Networks*
2009-35 Wouter Koelewijn (UL) *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling*
2009-34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*
2009-33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU) *Architectural Knowledge Management: Supporting Architects and Auditors*
2009-31 Sofiya Katrenko (UVA) *A Closer Look at Learning Relations from Text*
2009-30 Marcin Zukowski (CWI) *Balancing Vectorized Query Execution with Bandwidth-optimized Storage*
2009-29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*
2009-28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*
2009-27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*
2009-26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*
2009-25 Alex van Ballegooij (CWI) *RAM: Array Database Management through Relational Mapping*
2009-24 Annerieke Heuvelink (VUA) *Cognitive Models for Training Simulations*
2009-23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*
2009-22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*
2009-21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*
2009-20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*

2009-19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*

2009-18 Fabian Groffen (CWI) *Armada, An Evolving Database System*

2009-17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*

2009-16 Fritz Reul (UvT) *New Architectures in Computer Chess*

2009-15 Rinke Hoekstra (UVA) *Ontology Representation - Design Patterns and Ontologies that Make Sense*

2009-14 Maksym Korotkiy (VU) *From Ontology-enabled Services to Service-enabled Ontologies (making ontologies work in e-science with ONTO-SOA)*

2009-13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*

2009-12 Peter Massuthe (TUE, Humboldt-Universität zu Berlin) *Operating Guidelines for Services*

2009-11 Alexander Boer (UVA) *Legal Theory, Sources of Law & the Semantic Web*

2009-10 Jan Wielemaker (UVA) *Logic Programming for Knowledge-intensive Interactive Applications*

2009-09 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*

2009-08 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*

2009-07 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*

2009-06 Muhammad Subianto (UU) *Understanding Classification*

2009-05 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality*

2009-04 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*

2009-03 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*

2009-02 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*

2009-01 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*

2008-35 Ben Torben Nielsen (UvT) *Dendritic Morphologies: function shapes structure*

2008-34 Jeroen de Knijf (UU) *Studies in Frequent Tree Mining*

2008-33 Frank Terpstra (UVA) *Scientific Workflow Design; theoretical and practical issues*

2008-32 Trung H. Bui (UT) *Toward Affective Dialogue Management using Partially Observable Markov Decision Processes*

2008-31 Loes Braun (UM) *Pro-Active Medical Information Retrieval*

2008-30 Wouter van Atteveldt (VU) *Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content*

2008-29 Dennis Reidsma (UT) *Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans*

2008-28 Ildiko Flesch (RUN) *On the Use of Independence Relations in Bayesian Networks*

2008-27 Hubert Vogten (OU) *Design and Implementation Strategies for IMS Learning Design*

2008-26 Marijn Huijbregts (UT) *Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled*

2008-25 Geert Jonker (UU) *Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency*

2008-24 Zharko Aleksovski (VU) *Using Background Knowledge in Ontology Matching*

2008-23 Stefan Visscher (UU) *Bayesian network models for the management of ventilator-associated pneumonia*

2008-22 Henk Koning (UU) *Communication of IT-Architecture*

2008-21 Krisztian Balog (UVA) *People Search in the Enterprise*

2008-20 Rex Arendsen (UVA) *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.*

SUMMARY

The Internet has become an integral part of our daily lives. However, the essential task of finding information is dominated by a handful of large centralised search engines. In this thesis we study an alternative to this approach. Instead of using large data centres, we propose using the machines that we all use every day: our desktop, laptop and tablet computers, to build a peer-to-peer web search engine. We provide a definition of the associated research field: peer-to-peer information retrieval. We examine what separates it from related fields, give an overview of the work done so far and provide an economic perspective on peer-to-peer search. Furthermore, we introduce our own architecture for peer-to-peer search systems, inspired by BitTorrent.

Distributing the task of providing search results for queries introduces the problem of query routing: a query needs to be sent to a peer that can provide relevant search results. We investigate how the content of peers can be represented so that queries can be directed to the best ones in terms of relevance. While cooperative peers can provide their own representation, the content of uncooperative peers can be accessed only through a search interface and thus they can not actively provide a description of themselves. We look into representing these uncooperative peers by probing their search interface to construct a representation. Finally, the capacity of the machines in peer-to-peer networks differs considerably, making it challenging to provide search results quickly. To address this, we present an approach where copies of search results for previous queries are retained at peers and used to serve future requests and show participation can be incentivised using reputations.

There are still problems to be solved before a real-world peer-to-peer web search engine can be built. This thesis provides a starting point for this ambitious goal and also provides a solid basis for reasoning about peer-to-peer information retrieval systems in general.

SAMENVATTING

Het Internet is wezenlijk onderdeel geworden van ons dagelijks leven. Enkele grote gecentralizeerde zoekmachines domineren de essentiële taak van het zoeken naar informatie op het web. In dit proefschrift kijken we naar een alternatieve aanpak waarbij, in plaats van grote data centra, de computers gebruikt worden die we dagelijks gebruiken: onze desktops, laptops en tablets, om een peer-to-peer zoekmachine te bouwen. We geven een definitie van het gerelateerde onderzoeksveld: peer-to-peer information retrieval, en beschrijven hoe dit verschilt van andere onderzoeksvelden. Daarnaast geven we een overzicht van wat er al gedaan is, alsmede een economische invalshoek op het zoeken in peer-to-peer netwerken. We presenteren ook een eigen architectuur voor peer-to-peer zoeksystemen, geïnspireerd door BitTorrent.

Het distribueren van het leveren van zoekresultaten levert een probleem op: we moeten weten welke zoekvraag het beste door welke computer kan worden afgehandeld. We onderzoeken hoe we de documenten op computers zodanig kunnen representeren dat we elke zoekvraag naar de computer kunnen sturen met de meeste relevante documenten. Niet alle computers kunnen hun eigen representatie leveren. We kijken hoe we een representatie voor deze computers kunnen afleiden via hun zoekfunctie. De capaciteit van de computers in het netwerk loopt ver uiteen, wat het snel leveren van zoekresultaten bemoeilijkt. Als oplossing onderzoeken we een aanpak waarbij kopieën van zoekresultaten voor eerder gestelde zoekvragen worden vastgehouden en gebruikt om toekomstige, gelijkende, zoekvragen af te handelen. We laten zien dat deelname kan worden gestimuleerd met een reputatiesysteem.

Er zijn nog talrijke problemen die moeten worden opgelost voor een peer-to-peer zoekmachine voor het web gerealizeerd kan worden. Dit proefschrift biedt een startpunt voor dit ambitieuze doel en vormt een basis voor het redeneren over peer-to-peer information retrieval systemen.