# The use of rare key indexing for distributed web search

*MSc thesis by Koen Tinselboer*

# Abstract

In the last few years we have seen a rise in the of peer-to-peer applications in areas like file sharing [1][2][3]. However distributed information retrieval applications have not taken off yet. In such an application every peer (website) helps to maintain a global index of all information in the global document collection. When a website is updated the P2P search engine index can also be directly updated by the peer. Each peer only needs to contribute a limited amount of disk space and network bandwidth. Groups of websites can even form their own search engine which specializes in a their area of expertise. In this way the search process can be driven more by the Internet community.

In the first part of this thesis the previous work done in the field of distributed information retrieval is discussed. Most of the recently developed systems (like ALVIS [4], Minerva [5] and pSearch [6]) use a conceptually global but physically distributed index. This index is distributed using a distributed hash table (DHT [7]) based approach. The scalability of such systems is very good and the reported retrieval performance also approaches that of a centralized information retrieval system. However each project uses a different collection and a different set of queries to test their application. Therefore we cannot compare them directly with each other.

In the second part I discuss the implementation and evaluation of a distributed information retrieval system based on rare key indexing. Such an index stores sets of terms that appear near each other in a limited number of documents. This approach was first presented as part of the ALVIS project. In this thesis we tested the suitability of the approach for indexing and searching a realistic collection of websites using a subset of the WT10g collection [8]. To measure performance we look at the top-10 overlap between a single term index and a multi term index. In the best case the average overlap ratio was found to be only 7.5%. In the ALVIS project the average overlap ratio was between 83% and 97% [9]. I outline several causes that attribute to this huge difference. Based on the outcome of the experiments I have to conclude that the rare key indexing method scales well. However its retrieval performance on a realistic collection of websites is very poor. Therefore the rare key indexing method cannot be considered a good choice for a distributed web search application.

**Title:**
The use of rare key indexing for distributed web search

**Keywords:**
distributed information retrieval, web search, P2P, highly discriminative keys

**Supervisors:**
Djoerd Hiemstra, 1st chair
Rongmei Li, 2nd chair
Pavel Serdyukov, 3rd chair

# Preface

Before you lies my master thesis, which is the result of my graduation project of the master program Computer Science (Information Systems Engineering track) at the University of Twente. It is the result of more than half a year of research on distributed information retrieval.

P2P networks are widely used for several applications like file sharing, distributed computing, news groups (usenet), voice-over-ip (voip) and streaming video. From the succes of these applications and from the inherently distributed nature of the internet it follows that distributed web search may be an interesting possibility. How feasibile such an approach could be in reality is the topic of this thesis.

The main problem of distributed information retrieval is the issue of scalability. Peers need to exchange knowledge about the information they offer, but peers cannot use too much bandwidth so they need to choose which information to send. Therefore distributed information retrieval systems need to find a fine balance between scalability and retrieval performance. As a part of this graduation project a proof-of-concept application was developed which researched the concept of using highly discriminative term sets, instead of a single-term index.

With this thesis, I conclude the master Computer Science and thus my studies at this university come to an end. Therefore I would like to thank my girlfriend Femke for her continual moral support during my studies. Furthermore my thanks go out to my parents for their support during my time here. And last but not least, I would like to thank the members of the graduation committee for their help and feedback during the project.

Koen Johan Tinselboer
Wierden, the Netherlands
September 2007

*Luctor et emergo*

# Table of Contents

# List of Figures

# List of Tables

# 1 Introduction

Since the late nineties the internet has grown to hundreds of billions of webpages. Large scale search engines like Google and Yahoo only index a fraction of the internet. Google for example has dozens of datacenters worldwide providing a home to more than 450.000 servers [10]. So it is has become almost impossible for a new web search engine company to compete with these giants.

There is however an interesting alternative to centralized web search, namely distributed web search. In the most extreme case every (sub)domain could provide it's own little search engine. A meta search engine could then be used to query a select few of these little search engines to retrieve the best results. If this could be done efficiently, then perhaps search results could be more relevant or more up-to-date?

## 1.1 Problem statement

A little over ten years ago, in January of 1996, two PhD. students named Larry Page and Sergey Brin started a little research project in an attempt to improve web search results. By analyzing the links between websites they were able to improve the ranking of their results. A simple and clean interface as well as text advertisements instead of graphical advertisements caused their product (Google) to quickly become the de facto standard for web search.

The traditional centralized web search engines like Google, Yahoo and MSN have come to depend on an ever increasing number of server farms. The market for web search is dominated by a handful of multi-billion dollar companies and startups are having trouble to establish a foothold. One possibility, which is researched in this paper, would be to look at the other side of the spectrum.

A complete decentralization of web search could have several benefits, for example:

- There would be no need for huge server farms.
- Decentralized web search could be more tolerant to accidental failures or deliberate attacks.
- A push scenario instead of a pull scenario could be used for the updates to the index. Google has already realized that it needs to work together with web masters to index new or changed web pages more quickly. Webmasters can create so called Sitemap files [11], for example in the form of a RSS feed [12], which helps Google discover new pages. Users however cannot directly control if and when Google's search bot checks their Sitemap file. Therefore the use of Sitemap files cannot be considered a real push scenario; they are just used to help Google index the web more quickly by summarizing websites.
- Webmasters would became less dependent on the ranking Google assigns to their pages. A lot of websites depend on Google for most of their traffic. There are now companies that specialize in Search Engine Optimization (SEO), so webmasters can pay to achieve higher rankings.

- The decentralized network would not belong to any company or individual in particular. The Internet itself is inherently independent and distributed, so it would make sense to be able to search the web using a distributed, independent web search network.

The feasibility of a P2P system that operates over the Internet mainly depends on its scalability. Communication and storage costs need to stay reasonable even if the number of peers increases to a very large number. If a network doesn't scale well it will eventually fail because of bottlenecks in the network.

On the other hand P2P systems need to be able to achieve a retrieval performance similar to centralized search engines. This balance is what makes research into this area interesting. If you communicate too little, you cannot find what you're looking for. On the other hand if you communicate too much the network is not scalable and thus not very feasibile in reality.

## 1.2 Research questions

During this project the main focus is on researching the feasibility of extremely distributed web search. This research will consist of two parts, first researching existing distributed information retrieval systems and their approaches. The feasibilty of a distribute information retrieval depends on its scalability and its retrieval performance. The scalability of a system depends on how the data (index, routing tables, etc) is stored and what data needs to be stored. Further on in this thesis distributed hash tables (DHTs) are introduced, which are an excellent way to store data. What is stored, for example the kind of index, is a topic on which there is more debate in the community.

During the first part the following questions will be answered with respect to existing distributed information retrieval systems:

- Which systems for distributed information retrieval already exist?
- What are the differences among them?
- Distributed information retrieval system in order to be scalable need to find a balance between total knowledge of the global collection (= excellent retrieval performance) and only local knowledge (= excellent scalability in terms of the index). What is their approach to find the balance between retrieval performance and scalability?
- What are the advantages or disadvantages of the approach they use?

The second part of this thesis describes the a proof-of-concept application. This application will demonstrate one approach to indexing that tries to achieve the right balance between scalability and retrieval performance. An index contains a term (or a set of terms) and a list of references to documents/peers where those term(s) can be found in. The list of references to documents (or peers) is also known as a postings list.

The proof-of-concept will be based on the Highly Discriminative Keys (HDK) approach to indexing which was recently developed as part of the ALVIS project [4]. This indexing method was chosen because it offers a novel and scalable solution that also promises excellent retrieval performance. Since the approach is very new, more research is needed to confirm its validity. Both the ALVIS project and the HDK approach will be discussed in depth later on in this paper. Several experiments will be conducted using the WT10g test collection to research how feasible

the approach really is. During this second part the following research questions will be answered by those experiments:

- How does the average HDK vocabulary per peer scale?
- How does the average posting list size scale?
- How does the average number of postings per peer (index size) scale?
- What is the retrieval quality of the system compared to a centralized system, when using top-k retrieval as a measurement?

## 1.3 Thesis outline

This thesis basically consists of two parts, a theoretical part and a more practical part. In chapter two the basics of P2P applications will be discussed, followed by chapter three in which we have a look at some of the major P2P information retrieval systems that exist today.

In the second part the design of a proof-of-concept application will be described. This application will demonstrate a novel and scalable approach to indexing in a P2P distributed information retrieval system. The design of this proof-of-concept will be discussed in chapter four, followed by results from several experiments in chapter five. An overall discussion and suggestions for future work can be found in chapter six. And finally in the seventh and final chapter we present our conclusions.

# 2 Theoretical background

When one wants to look to the future, one first has to look at the present and the past. In this chapter the theoretical background of P2P networks will be discussed. The theoretical background information presented in this chapter will serve as a foundation for the rest of the paper.

## 2.1 Fundamental hardware constraints

In a decentralized web search scenario there are limitations to certain costs. The most obvious are storage and communication constraints [13].

**Disk usage**

There is of course a limit to the amount of disk space a peer can dedicate to the network. How much a peer can use is of course very dependent on the server(s) that the website is running on. Therefore it is important to limit the disk usage to an amount that is acceptable to all participating webservers. This value depends on a number of design choices and network properties, including but not limited to the following:

- Type of index: each peer can store just a local index, a part of a distributed global index or the entire global index.
- Type of mapping: for example single-term-to-document or term-set-to-peer.
- Length of the posting lists: are just the top-k results stored or are all possible matches stored?
- Exclusivity of the terms or term sets. Do we store all terms or term sets or do we, for example, store only the ones that do not occur often.
- The number of documents or peers in the network.
- Compression techniques that are used.

**Cost of communication**

The communication costs of the P2P network should also be limited. Most of the bandwidth should be used by the webserver and not the P2P search network. To keep communication costs down the P2P network should be able to handle queries very efficiently. The more infrequent indexing process can be somewhat less efficient.

## 2.2 Name-based retrieval versus content-based retrieval

Most people will relate the term P2P to popular file sharing applications. Using such an application an user can for example search for all MP3-files that contain the string “Madonna” in the filename. The user can then select and subsequently download a file from the list of results. This kind of information retrieval is called name-based retrieval [14]. The system searches for matches between query terms and document names or other document identifiers. In such a scenario the user assumes that the MP3-file with the string “Madonna” in the filename really is what it claims to be, namely an audio file that contains a song by Madonna. The user performs a so called “known item” search, but he has no guarantee that the result is what he expects it to be.

Distributed information retrieval applications however cannot rely on for example just the title or the url of a web page. The content of the web page needs to be examined so that a list of keywords or perhaps a summary can be produced. This kind of information retrieval is known as content-based retrieval [14]. Unfortunately content-based retrieval is inherently more complex than name-based retrieval, especially in a distributed setting. As explained in the previous section the peers are bound by several constraints like communication costs and storage costs. So distributed information systems need to find a way to represent the contents of a document using as little storage space and network bandwidth as possible, while the user can still find what he is looking for. An optimum balance will result in good scalability as well as good retrieval results.

## 2.3 Architecture of a P2P system

P2P information retrieval systems need to accomplish a number of tasks like routing messages, updating indexes and ranking results. To cleanly separate these concepts a layered architecture is recommended. It is hard to make a general assumption about what is the best architecture for an information systems so here we assume a very basic separation on the basis of the tasks that such a system should perform. A typical separation into four layers can be seen in Figure 2.1 below.

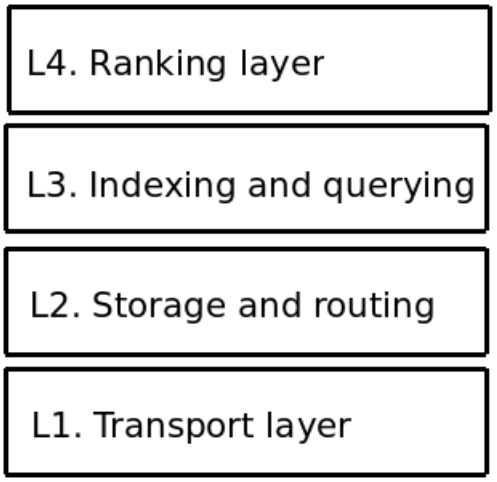


Figure 2.1.: The typical four layers of a P2P System.

The model is made up of four layers, from the lowest to the highest layer they are:

1. **Transport layer.** This layer deals with the transport of data between peers over a network like the Internet using TCP/IP.
2. **Routing and storage layer.** The implementation of this layer depends heavily on the type of network that is used. Most modern systems use a distributed hash table (DHT) as the basis of their network. DHTs will be discussed in one of the following sections.
3. **Index and query layer.** This layer maintains an index structure of a document collection. The document collection can be either local or global, depending on the design of the network. Indexes are often conceptually global, but physically distributed on the routing level. Queries are also handled by this layer.

4. **Ranking layer.** This layer implements the distributed document ranking that is needed when query results are combined.

## 2.4 Transport layer

The Transport layer is the lowest layer in a P2P system. On this layer data is physically routed through cables, routers, firewalls et cetera until it reaches its destination. Most P2P systems are built to operate over the Internet so the time it takes for a message to be handled by the Transport layer is of some importance. Usually the routing and storage layer will use data from the Transport layer like the round trip delay time or the number of hops between to peers into account. Peers that can be reached quickly may not just be preferred, in some cases they are assigned more important tasks than other peers. Such a hybrid approach, in which some peers have more responsibilities than others, will be discussed in the next layer.

## 2.5 Routing and storage layer

This layer deals with routing messages and storing data. In the first section the different network topologies will be discussed and compared. The second section will discuss hash tables which are the most commonly used data storage structure in P2P networks.

### 2.5.1 P2P network topologies

Different P2P systems use different network topologies. Which type of network topology is most suited for a specific system depends heavily on the type of application. There are basically four types of network topologies [15]:

1. Unstructured pure P2P
2. Structured pure P2P
3. Hybrid P2P with centralized indexing
4. Hybrid P2P with distributed indexing

Some papers will combine the hybrid network topologies into one type, however here they are treated separately to illustrate the differences between them.

**Unstructured pure P2P**

The first two types are called *pure* P2P network topologies because in this type of network peers are equal in function. Therefore there is also no kind of centralized control. The difference between the two types is their structure. In an unstructured pure P2P network the peers are connected to each other without any specific kind of structure.

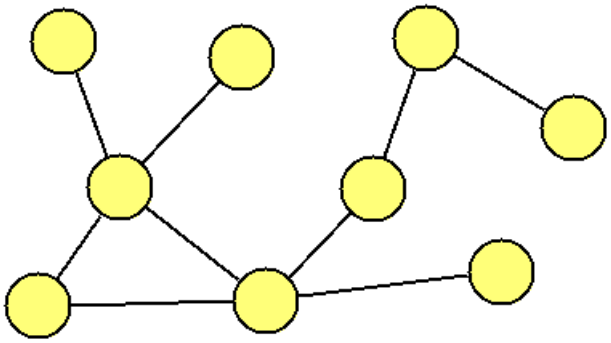

Figure 2.2.: An unstructured P2P network.

**Structured pure P2P**
Peers in a structured network however are always a part of specific structure, for example a ring or a fully connected network.

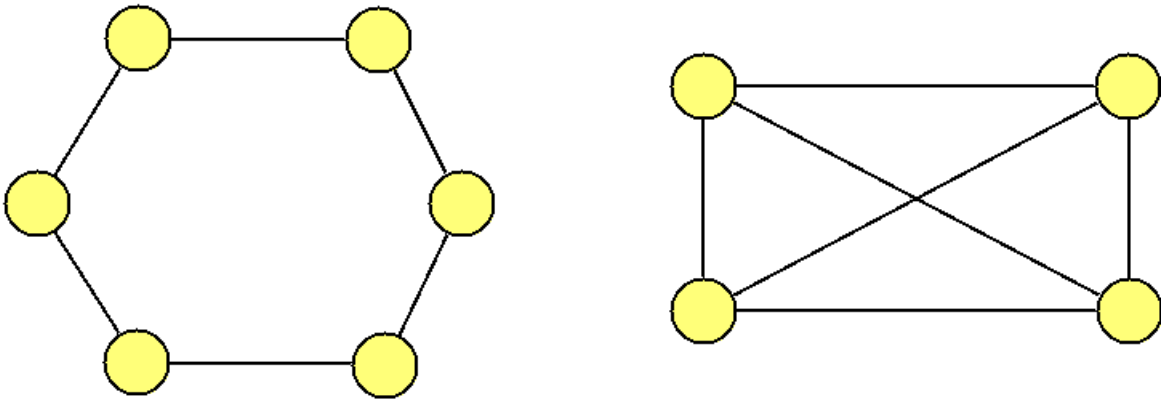

Figure 2.3.: Structured P2P networks. On the left a ring network, on the right a fully connected network.

**Hybrid P2P with centralized indexing**
Hybrid P2P networks are networks which are not pure; that is to say the peers are not equal in functionality. Some peers have more functionality as they provide additional indexing services. In a hybrid P2P network with centralized indexing there is just one server that maintains an index of all the information that is shared by the connected peers. In practice the index is usually provided by a set of servers to handle the load. However to the peers 'outside' there appears to be just one server. Often peers can however connect to multiple centralized indexes, which each offers its own collection of information.

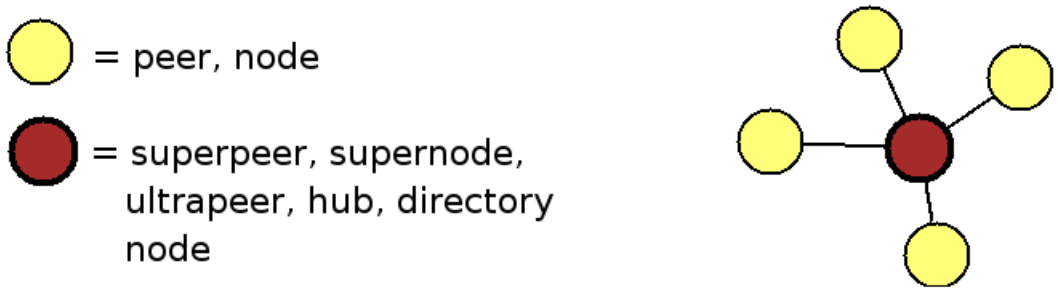


Figure 2.4.: A hybrid P2P network with a centralized index.

**Hybrid P2P with distributed indexing**

A hybrid P2P network can also use a distributed indexing approach in which the index is distributed among a number of so called supernodes. These supernodes often not only maintain the central index but also handle and route search requests from other peers. One could think of these supernodes as high-speed motorways for indexing and search purposes. The actual exchange of the information is usually handled directly between the peers, instead of via supernodes. Supernodes are often dynamically assigned on demand, when a suitable candidate peer is available. Usually the index is stored using a distributed hash table (DHT). The use of a distributed index not only provides better scalability, it can also improve fault-tolerance by duplicating parts of the index across several peers.

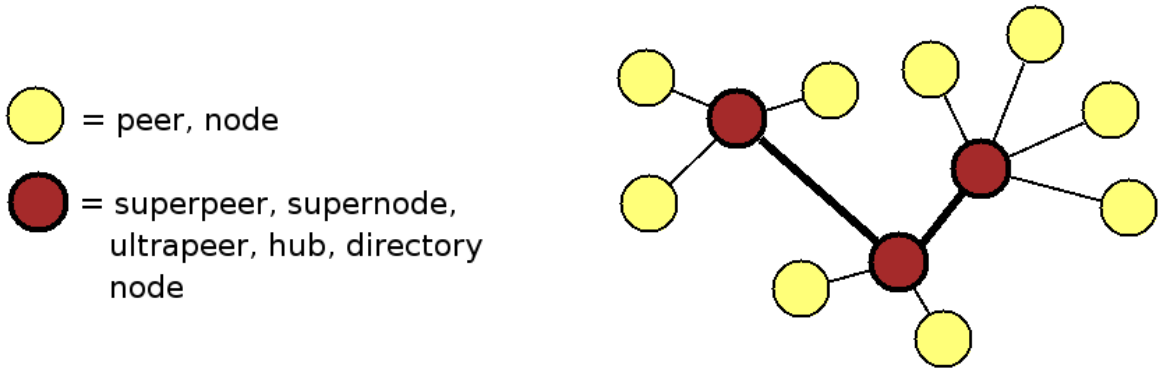


Figure 2.5.: A hybrid P2P network with a distributed index.

**Comparison of the four P2P network topologies**

Each of the four P2P network topologies discussed above has its own set of properties. To compare them here we look at four criteria:

- **Robustness**; does the network still function if certain peers go down?
- **Scalable**; does the network scale well to thousand or even millions of peers?
- **Flexible**; is the assignment of peers flexible?
- **Manageable**; can you control the network by controlling part of it?

In the following table we see that unstructured pure P2P networks are very robust and flexible, but they do not scale well and they cannot be managed. Structured P2P networks improve on this implementation by making the network scalable. However the most popular type of P2P network by far are the hybrid networks, in which so called supernodes (or directory nodes) are used to steer the network. Hybrid P2P networks with a central index lack robustness, scalability and flexibility because they depend on a central indexing server. This may however not be much of a problem if there are a lot of indexing servers to choose from. Hybrid P2P networks with a distributed index solve these issues by distributing the load and the responsibility for the index over a dynamic number of supernodes. therefore this kind of network is not only robust and flexible, but it also still manageable and very scalable.

| | **Unstructured pure p2p** | **Structured pure p2p** | **Hybrid p2p with centralized indexing** | **Hybrid p2p with distributed indexing** |
|---|---|---|---|---|
| **Robustness** | Yes | Yes | No | Yes |
| **Scalable** | No | Yes | No | Yes |
| **Flexible** | Yes | No | No | Yes |
| **Manageable** | No | Yes | Yes | Yes |

Table 2.1: Comparison of P2P network topologies [15].

### 2.5.2 Distributed Hash Tables (DHTs)

The most frequently used approach to storing an index is a hash table. A hash function [16] is a reproducible method which maps some amount of data onto a relatively small number. It creates a digital fingerprint by substituting and transposing the data which results in a hash value. For an example of a typical hash function see Figure 2.6. The example also illustrates an characteristic property of a hash function; a small change in the input dramatically changes the output. Hash functions are also widely used in the field of cryptography, for example to encode passwords. Well known and widely used hash functions are SHA-1 [17] and the somewhat older MD-5 [18].

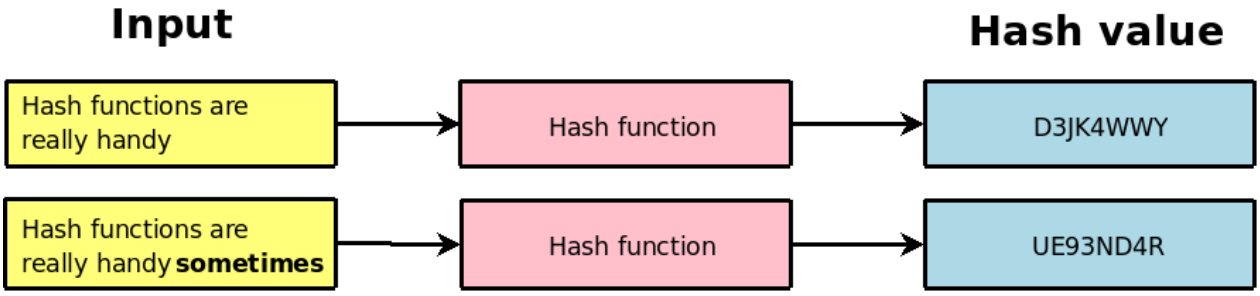


Figure 2.6.: A typical hash function at work.

To guarantee that a hash value can be used as a fingerprint the hash function needs to ensure that there are very few hash collisions. A hash collision is the event that two different inputs, produce the same output. So if the hash value would be a person's fingerprint, this would be the event that two different people have the exact same fingerprint. That would however be very unlikely as not even identical twins have exactly the same fingerprints. In the field of computer science a small chance that two different inputs map to the same output may however sometimes be acceptable if it substantially increases the rate of compression. This will be discussed further along in this section when Bloom filters are explained.

**Distributed Hash Tables (DHTs)**
A hash table is basically a long list of fingerprints, which enables a fast lookup of a data record. In a P2P network the responsibility for the parts of the hash table is often divided among a number of participating peers. Such a table is known as a distributed hash table (DHT) [7].The underlying network topology is designed to efficiently route messages to the owner of any given key. P2P network topologies are further discussed in the next section.

DHTs scale very well to large numbers of peers and they can handle the arrival and failure of peers. A lot of P2P file sharing applications use distributed hash tables, but there also used in other systems like cooperative web caching (coral), domain name services (DNS) and instant messengers. However file sharing applications like Napster [19], Gnutella [20] and Freenet [21] were among the first to use them to efficiently share information (files) over the internet.

DHTs have the following properties:
- Decentralized, each peers is responsible for a part of the total table.
- Scalable, the system can scale easily as the load is divided among the peers.
- Fault tolerant, the continuously joining, leaving and failing of nodes should not have much impact on the system.

A distributed hash table consists of an abstract key space, for example the set of 160-bit strings. Peers are responsible for part of this key space, according to a certain key space partitioning scheme. An overlay network connects the peers so they can find the peer corresponding to any given key in the key space.

Each peer is a part of the overlay network and as such it maintains a set of links to other peers. The actual implementation differs but each distributed hash table topology implements a variant of the following concept. If a peer does not own key $k$ then the node either knows which peer does or it knows which peer is closer to $k$ than itself. Using a greedy routing algorithm it is then easy to get a message across from one peer to any other in the network. To limit the number of hops a trade off has to be made with respect to the number of neighbors a peer can have. Most implementations choose this number to be $O(logN)$ which incurs a route length of $O(logN)$.

**Bloom filters**
Some research papers [13] have suggested the use of Bloom filters to strongly compress the index. A Bloom filter [22] is a very space-efficient probabilistic data structure in which false positives (but not false negatives) are possible. It can be used to test if a certain element is member of a set. However the members of the set themselves cannot be retrieved. An empty Bloom filter is a bit-array of $m$ bits, all set to 0. One also needs $k$ hash functions to map a key

value to one of the array positions. To add an element the *k* hash functions are used to calculate the array positions that have to be set to 1. If one needs to test if the element is part of the set then you can just check if the corresponding array positions are all set to 1. If so, the element may be part of the set. The likelihood depends on the amount of false positives you allow. Because of the nature of a Bloom filter removing an element is not possible, since multiple elements may be mapped to the same array positions.

## 2.6 Indexing and query layer

In a distributed information retrieval system it is important to have an index with which a peer can find the information it needs quickly and reliably. Due to the P2P nature of the system we are however far more limited with respect to storage space and network bandwidth usage. This conflict of interest is what makes the research into this area so interesting, since the results are mostly a trade-off between high costs or good results. To make extremely distributed web search a reality you need to get the balance just right.
There are a number of basic indexing strategies for information retrieval:

1. **Centralized global index;** no P2P network, but a centralized global index instead. The index is often duplicated and/or distributed over a vast number of servers, which are located in a relatively small number of data centers. This is the strategy behind major searchengines like Google [23] and Yahoo [24].
2. **Global single-term-to-document P2P index;** the index consists of a list of single terms which map directly to documents stored on the peers.
3. **Global key-to-document P2P index;** basically the same as the previous indexing strategy, except a set of terms (a key) is used instead of a single-term. This relates better to a realistic user query, but is also requires mapping the query to key(s). The ALVIS project [4] uses this approach for its search engine.
4. **Global key-to-peer P2P index with federated local indices;** basically the same as the previous indexing strategy, except now the peers need to be contacted to perform a local search. The index may be smaller since a peer only appears once in a key's posting list, but on the other hand the local searches add significantly to the network traffic.
5. **Global single-term-to-peer P2P index with federated local indices;** basically the same as the previous indexing strategy, except now the index again consists of single terms instead of term sets. This usually results in a larger index as term sets are more discriminative, while it may not improve search results enough to be worth it.
6. **Federated local P2P indices;** each peer has its own collection and doesn't share any info about it with other peers beforehand. The search queries are flooded to all other peers since noone knows who has the information. A good example of this type of network is Gnutella version 0.4 [25].

The global indexes in indexing strategies 2, 3, 4 and 5 can themselves be distributed over the peers, but this partitioning is not directly related to the documents in the local collection. Often it is however a good idea to distribute the load and responsibility over a select number of the peers, as discussed previously in section 2.5.

**Comparison**

To compare the scalability of the indexing strategies presented above we analyze the traffic load in the peer network. To simplify matters we do not consider traffic inside the network. Only the

load of the number of messages that are going into and are subsequently coming out of the network is calculated.

To perform the calculations certain assumptions need to be made and variables need to be defined. All of the variables that are used in the calculations are defined in table 2.2.

| Variable | Definition |
|---|---|
| $N$ | number of peers |
| $r$ | the fraction $r$ of $N$ peers that produce a query at any given moment |
| $D$ | document collection |
| $d_{max}$ | the maximum number of documents a peer contributes to $D$, so $\|D\| = d_{max}N$ |
| $e$ | uniform term size |
| $q_{max}$ | maximum number of terms in a query |
| $f$ | uniform posting size |
| $S$ | size of the index |
| $V$ | vocabulary |
| $u$ | size of a term's single posting (either a document or a peer reference) |
| $p_{max}$ | a limited number of peers |
| $n(q)$ | the number of term sets (keys) associated with a query of size $q$ |
| $DF_{max}$ | a threshold based on the global document frequency which divides a set of keys into two disjoint classes, a set of rare and a set of non-rare keys |

Table 2.2: Variable definitions for the comparative scalability analysis

The indexing strategies will be analyzed seperately, followed by a discussion in which they will be compared to each other.

1. **Centralized global index;** since there is no P2P network over which the messages need to travel this type of indexing strategy will not be analyzed. We only compare the network load of the applications that use a P2P network here.
2. **Global single-term-to-document P2P index;** on average there are $rN$ query messages, each of size $e\ q_{max}$. There will also be $q_{max}$ answer messages, because it is a single-term index. These messages will be the size of the average posting list size $S/|V|$ multiplied by the size of a term's single posting $u$. The total traffic thus amounts to $(e\ q_{max} + u\ q_{max}\ S/|V|)\ rN$.

   The growth rate of the total amount of traffic is determined by analyzing the growth rate of parts of the equation:
   - $e\ q_{max}$ **grows with** $O(1)$**.** Because $e$ (the uniform term size) and $q_{max}$ (the maximum number of terms in a query are independent from $N$, this part of the equation will grow as $O(1)$.
   - $u\ q_{max}$ **grows with** $O(log(N))$**.** The size of a term's single posting ($u$) is limited because each peer only brings a bounded amount of documents in the system, namely $d_{max}\ N$. The growth rate for $d_{max}\ N$ is $O(N)$, so a lower bound for $u$ is $O(logN)$.

In other words, it takes a minimum of *logN* bits to store an unique id for each peer in a collection of *N* peers. Each new peer only contributes a fixed number of documents to the collection ($d_{max}$). therefore the asymptotic growth rate of the size of a document id is equal to the growth rate of the size of a peer id, which is *O(logN)*.

- ***S/\|V\|* grows with $O(\sqrt{N})$.** Since the global index size *S* is $O(|D|) = O(d_{max} N) = O(N)$ and the vocabulary grows as $O(\sqrt{N})$ because of Heaps law [26], therefore the average posting list size *S/|V|* will grow as $O(\sqrt{N})$.
- ***rN* will grow with *O(N).***

The total amount of traffic will thus grow with $O(N\sqrt{N}\log(N))$.

3. **Global key-to-document P2P index;** instead of using a single-term index, this index uses termsets also known as keys. The other main difference between this type of index and a single-term-to-document index is that here the average posting list size is limited by $DF_{max}$. therefore *S/|V|* grows with *O(1)* instead of $O(\sqrt{N})$. Basically the size of the average posting list is limited, while at the same time we accept an increase in the vocabulary size. However the growth rate of the vocabulary is bounded by $O(|D|) = O(d_{max} N) = O(N)$.
   Using keys instead of single-terms means that query expansion often needs to be used to map query terms to keys. The number of keys associated with a query of size *q* is defined as *n(q)*. And thus the total amount of traffic amounts to $(e\ n(q) + u\ n(q)\ S/|V|)rN$. The growth rate is similar to that of a global single-term index, except that here *S/|V|* grows with *O(1)*. therefore the total amount of traffic grows with *O(N log(N))*.
4. **Global key-to-peer P2P index with federated local indices;** in this two-step approach first a list of peers that can possibly answer the query is retrieved from the P2P overlay. However since here posting elements are peer references instead of document references the second step involves sending queries to $p_{max}rN$ peers, which return at most $d_{max}$ documents. While the second step is thus bounded by *O(N)*, the first is still bounded by $O(N\sqrt{N}\log(N))$ as in the case of a global single-term-to-document index.
   The change in granularity means that the average posting list size will be shorter since it only stores references to a number of peers, instead of to the documents on those peers. However a second step in which these peers are queried is necessary to obtain the list of resulting documents. So the size of the index will be smaller because of the smaller posting lists. However the search of the peers may return more results if some documents do not appear in the key-to-document index, while they are present in the peers local document collection. The asymptotic behaviour of the function that calculates the amount of network traffic is however the same.
5. **Global single-term-to-peer P2P index with federated local indices;** the reasoning here is the same as in the previous case. While the second step is thus bounded by *O(N)*, the first is still bounded by $O(N\sqrt{N}\log(N))$ as in the case of a global single-term-to-document index.
6. **Federated local P2P indices;** each peer that sends a query, needs to send it to *N-1* other peers. therefore the number of messages is *rN(N-1)*. The size of a query message is limited to $e\ q_{max}$ and the size of the answer message to $f\ d_{max}$. The total amount of traffic is thus bounded by $(e\ q_{max} + f\ d_{max})rN(N-1)$, which grows with $O(N^2)$ and is thus not scalable.

Federated local P2P indices offer the worst scalability, since traffic grows with $O(N^2)$, of the five P2P indexing strategies. The use of keys instead of single terms improves scalability because this

method limits the average posting list size by $DF_{max}$. This use of a threshold variable in a search scenario is quite realistic, since users are often just interested in the top-k results.

A key (term set) must be discriminative with respect to the document or peer it is associated with to be of value. Users often pose multi-term queries so keys may relate better to a query than a combination of single terms. The problem of mapping a query term set to one or more keys is however not always easy to solve.

The results of the scalability analysis show that the global key-to-document P2P index approach and the global key-to-peer P2P index offer the best scalability. Results of the analysis are also summarized in table .

| P2P indexing strategy | Rate of growth |
|---|---|
| Global single-term-to-document P2P index | $O(N \sqrt{N} log(N))$ |
| Global key-to-document P2P index | $O(N log(N))$ |
| Global key-to-peer P2P index with federated local indices | $O(N log(N))$ |
| Global single-term-to-peer P2P index with federated local indices | $O(N \sqrt{N} log(N))$ |
| Federated local P2P indices | $O(N^2)$ |

Table 2.3: Results of the scalability analysis for various P2P indexing strategies

## 2.7 Ranking layer

The ranking layer is the highest layer in a P2P architecture. Due to the distributed nature of a P2P system it is not so trivial to rank a list of results. In a centralized setting an information retrieval system will have all the global document collection statistics that it needs. However in the case of a P2P system this information needs to be communicated either before or during the ranking process. So there are basically two strategies here:

1. **Using predetermined weights.** The index can be used to store the quality of a posting list so they can be retrieved along with the posting list. Usually such a weight can however only be computed locally per peer. Another option would be to get a global ranking for each item in the index once in a while with the help of the other peers. Such a value could considered a 'cached' version of a ranking for a term or a set of terms in the index. However the results of a query are often a combination of the results for each query term, which makes the use of individual weights a problem
2. **Ranking on demand.** In this approach the peers that share the documents are contacted to obtain more information so a adequate ranking of the results can be performed. A very basic approach could be to obtain the text of the resulting documents so the collection can be ranked locally on the peer that issued the query. The number of results for an index term would have to be limited to minimize bandwidth consumption. One big advantage of this approach is that the text can be reused when presenting the results to the user.

Both strategies have their pros and cons, however ranking on demand gives the most accurate results in general.

## 2.8 P2P file sharing applications

One of the most widespread and (in)famous uses of P2P networks are file sharing systems. If we look at the evolution of file sharing applications we can distinguish several generations. Researchers do not always agree on how many generations there are and what their characteristics are. In this paper the most common division into four generations of P2P file sharing applications [27] will be discussed. To illustrate the different generations, several well known P2P file sharing applications will be discussed.

### 2.8.1 First generation: server-client

The first generation of P2P file sharing applications used a centralized file list. These applications consists of a hybrid P2P network which makes use of a centralized index, as discussed in section 2.5.3. The peers register with the server and the files it hosts are added to the index. Another peer can then perform a search query by asking the central server if there are any files that match the query. Files are transferred directly between the peers.

The main disadvantages of this first generation are the bad scalability and the threat of legal prosecution. Both these problems are the result of using a central indexing server. First generation file sharing networks do not scale well because the index server quickly becomes the bottleneck. A company would need to keep increasing their server farm to provide indexing services fast and reliably. Furthermore by using a central server the company behind the file sharing network could be held liable for any copyright infringements which it basically facilitates by indexing copyrighted files.

**Hybrid P2P network with a centralized index: Napster**

The earliest well known file sharing application was Napster [19]. The original Napster was released in June of 1999 by a student who wanted an easy way to share and find music in the form of MP3 files. A structured P2P network was used in which a centralized index server provided the search results. The actual file sharing between peers was however done directly. Usage of Napster peaked in February 2001 at 26.4 million users worldwide. However the use of a centralized index server left the company vulnerable to legal prosecution and the network was taken down later that same year.

### 2.8.2 Second generation: decentralization

Napster made clear that P2P file sharing networks were here to stay. Unfortunately most of the files that are shared on these networks can not be freely distributed. This holds for most movies, music (MP3) and applications for example. Although there are exceptions like open-source software, freeware, etc. The second generation of P2P file sharing networks however (tries) to completely eliminate the need for a centralized index server. The network topologies used to achieve this goal however differ per application, so they will be discussed by example.

**Unstructured pure P2P network: Gnutella 0.4**

Because of the legal problems that Napster faced the Gnutella network at first used a completely unstructured P2P network in which all peers were equal, as discussed in section 2.5.1. The success of the original Gnutella was however also the cause of its downfall. Flooding search requests over a unstructured P2P network like Gnutella 0.4 [25] caused bottlenecks in the network.

**Hybrid P2P networks with distributed indexing: Gnutella 0.6 and FastTrack**
The Gnutella developers quickly realized the problem and newer versions of Gnutella [20] used a hybrid P2P network with distributed indexing. Such a network is made up of a mix of regular peers and superpeers. The index is distributed over the superpeers, so no single peer needs to bear the load alone. The first widely used implementation of a hybrid P2P network with distributed indexing was the FastTrack network. The Gnutella developers quickly adopted the same approach.

The most famous FastTrack client is known as Kazaa [28]. The FastTrack network struck a compromise between a hybrid P2P network with a centralized index (Napster [19]) and a completely unstructured (Gnutella 0.4 [25]) network. By using a hybrid P2P network with a distributed index in which some peers (directory nodes) were more important than others they combined the best of two worlds. Kazaa however still used a central server for logging in, which meant it was still vulnerable to legal prosecution. When lawsuits loomed the company was quickly sold on by its original developers to an Australian based company called Sharman Networks.

**Hybrid P2P networks with centralized indexes: eDonkey and Bittorrent**
Although the use of a centralized index was the cause of scalability and legal problems in the first generation it is still popular among second generation file sharing applications. However instead of using a single index server that is controlled by a company file sharing applications like eDonkey [1] and Bittorrent [2] now use a vast number of indexing servers. The difference between eDonkey and Bittorrent is mainly the way in which the index servers are accessed.

Bittorrent websites are traditional websites which provide so called .torrent files. A torrent file contains meta data about a set of files, like the filename, size, hash value and most importantly the url to a tracker. A tracker is the location where seeders (uploaders) and leechers (downloaders) register to share a file. Torrents are downloaded in chunks, so leechers are quickly also seeders for parts of the torrent they already downloaded. Peers who upload are also more likely to achieve higher download speeds. There are basically two different types of torrent sites, namely public and private sites. The first are accessible two everyone and offer a lot of files, but often of a lower quality and download speed. A few public torrent sites are specialized and only offer a few big files to download. Basically they use the Bittorrent protocol to lighten the load on the server that big files cause. A good example of such a case would be a public tracker that offers ISO-images of a specific Linux distribution like Ubuntu or Fedora. The second kind of torrent sites, private torrent- sites, are often specialized in like for example music, movies, television series or e-books. Users need to register and often new members are welcome by invitation only. They offer higher download speeds and more high-quality files, but you need to maintain at least a 1:1 upload/download ratio or donate money for the upkeep of the servers.

The eDonkey servers are more like Kazaa, except that the user has a list of servers to which he can connect. He can either let the client-application choose, or he can connect to the servers he wants manually. Clients then register the meta-data of the files they are sharing. Users can search by querying the meta data or by directly searching for a file's network identifier. The network identifier is a unique hash value. One way to find specific files on the network is to visit a

website which hosts a database of such identifiers, retrieve a specific identifier and then start downloading it using an eDonkey client.

### 2.8.3 Third generation: anonymity for all

The third generation of P2P file sharing adds anonymity features. This is achieved by routing traffic through other peers and by using strong encryption methods. Even the network administrators cannot see what is being transferred and to whom. Unfortunately anonymity had its downsides. Due to the rerouting and the encryption downloads are a lot slower than on second generation networks. Furthermore the anonymity causes the network to be abused for exchanging illegal content like child pornography, extremist literature, etc. Because of the overhead third generation file sharing applications are only used on a small scale. Well known third generation P2P file sharing applications include ANts P2P [29] and Freenet [21].

### 2.8.4 Fourth generation: streams over P2P

Most divisions of P2P file sharing applications are limited to two or three generations. Some however also mention a fourth, namely the use of P2P networks to send streams instead of files. For these purposes a swarming technology (similar to Bittorrent) is used instead of a treelike network structure. A swarm is a inter-connected group of peers which all communicate with a central registry and they also communicate directly with each other. Some well known applications that use P2P networks to send video or radio streams are Joost [30], Babelgum [31] and Peercast [32].

# 3 Related work

In this section we discuss previous work as well as some existing approaches to P2P information retrieval.

## 3.1 Routing and storage layer implementations

In this section several P2P overlay networks that are based on distributed hash tables are discussed. DHTs were introduced earlier in sections 2.6.2. The four overlay networks that are discussed here are CAN [33], Chord [34], Pastry [35] and Tapestry [36]. They were created to eliminate the main problem of the first generation of P2P (file sharing) systems, namely the reliance on centralized servers. Some of the overlay networks discussed in this section are used as a base for the newer P2P information retrieval systems, which are introduced further on in section 3.2.

### 3.1.1 CAN

The Content Addressable Network (CAN) [33] is a distributed hash-based lookup protocol that provides fast lookups on an Internet-like scale.

**Naming and structure**

Machines are identified by their IP address and data records are assigned a unique key *K*. CANs design is based around a virtual d-dimensional Cartesian coordinate space on a d-torus. A two-dimensional torus can be represented as a grid or matrix in which if you go left/right in the most left/right square you would end up on the most right/left square. The same holds for the bottom and the top side.

This virtual space is partitioned into many small zones which each machine corresponds to one of the zones. Machines are neighbors if they can be reached in one step in any dimension. For example in a 2-dimensional space the zones directly on the left and the right, as well as above and below a zone are its direct neighbors. Each machine knows its neighboring zones and the IP addresses of the machines in those zones. A node is added by assigning it a zone of its own or by splitting up an existing zone, as illustrated in Figure 3.1.

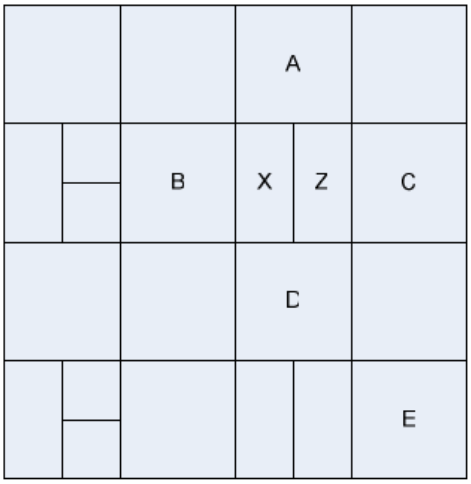

Figure 3.1: CAN after adding node Z

**Locating and routing**

To start the virtual position for a key is calculated. Then the query is passed through neighbors until it finds the machine it is looking for. An example can be seen in Figure 3.2 below. Each machine maintains contact with 2d neighbors on average and with a max of 4d. The average routing path length is equal to $(d/4)n^{1/d}$.The network can achieve *O(logN)* performance on the routing time and the data operations if *d=(logN)/2*.

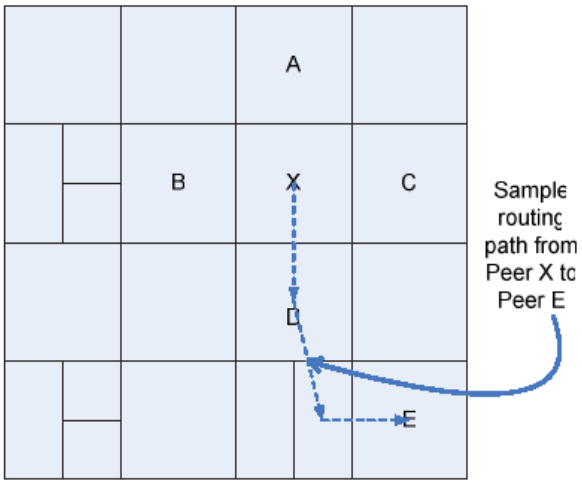

Figure 3.2: Routing example from node X to node E

**Data and topology updates management**
Both data insertion and deletion can be achieved in $(d/4)n^{1/d}$ hops. CAN also supports dynamic joining and leaving of machines. Furthermore it can detect and recover from node failures automatically. While the average cost for machine joining is $(d/4)n^{1/d}$ the costs for machine leaving and failure recovery is constant time.

### 3.1.2 Chord

Chord [34] is a distributed lookup protocol developed at the MIT Laboratory for Computer Science. Like CAN it also scales very well and it offers fast data locating.

**Naming and structure**
Machines are identified by assigning an *m*-bit nodeID, which is based on a hash value of the machine's IP address. Data records consist of a key *K* and a value V and they are also assigned a *m*-bit ID by hashing the key *K*. The location of the data is thus identified by this ID.

Chord uses a one-dimensional circle (a 'chord') to order the machines. Each machine is mapped onto the ring based on their nodeID. The number of machines is limited by *m* because the maximum number of machines $N = 2^m$. The Chord ring is divided into *1+logN* segments, namely itself and *logN* segments with length 1, 2, 4, 8, 16....N/2. The routing table contains not only the boundaries but also the successor (nearest node clockwise) of the virtual node. So each machine only needs *O(logN)* storage space to maintain a structure. Routing a message can also be done in *logN* steps, as shown in Figure 3.3.

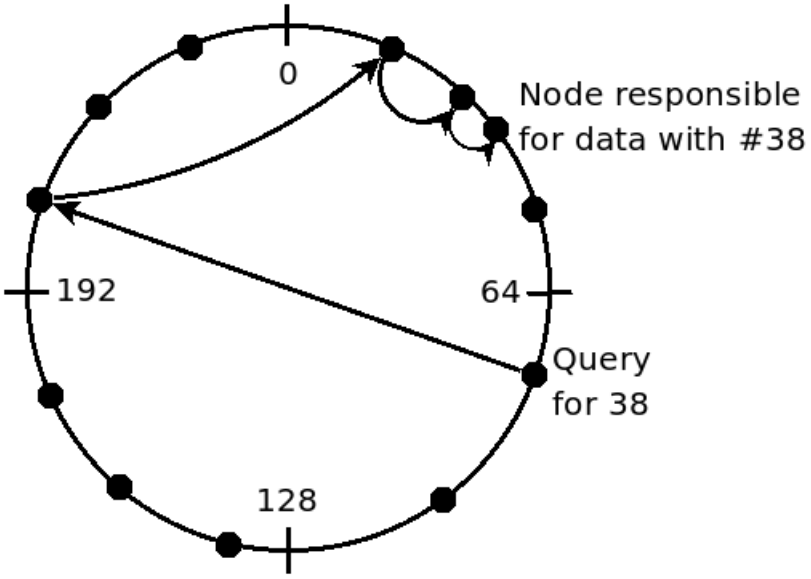


Figure 3.3.: A lookup for the data with ID 38 in a Chord data structure.

**Locating and routing**
To find a specific data record first its *m*-bit ID is calculated by hashing its key *K*. Then the routing table can be used recursively to locate the successor of the segment that contains the target, which is in turn selected to be the next router until the target is reached.

**Data and topology updates management**
Higher availability can be achieved by replicating the data. All operations on the data can be done in *O(logN)* time. Machines can join or leave at any time which will cost $O(\log^2 N)$ with a high probability. In the worst case however it will need *O(N)* time. Chord can also automatically detect and recover from node failures.

### 3.1.3 Pastry

Pastry [35] is an object location and routing system for P2P systems that communicate via the Internet. Like CAN and Chord it offers excellent performance, reliability and scalability. Pastry is actually used in a number of applications, for example:

- Splitstream [37], an application-level multi cast in which peers share the load.
- Squirrel [38], which uses Pastry as a data object location service.
- PAST [39], a large scale P2P persistent storage application.
- Pastiche [40], a P2P backup system.

**Naming and structure**
The nodes in a Pastry network are each assigned a unique *128*-bit nodeID at random. Like the Chord project, Pastry also uses a one-dimensional circle to order the nodes on. NodeIDs are assigned in such a way that the nodes are uniformly distributed over the $2^{128}$-1 spaces.

Nodes maintain this structure by storing a routing table, a neighborhood set and a leaf set. The routing table consists of logN rows which each store *2b-1* entries. The nth row of the table contains nodeIDs and IP addresses of nodes whose nodeIDs are equal in the first n-digits. A neighborhood set only stores the nodeIDs and IP addresses of the closest nodes. And a leaf set contains the nodes with the $|L|/2$ numerically closest larger nodeIDs, as well as with the $|L|/2$ numerically closest smaller nodeIDS.

**Locating and routing**
If a query message needs to be routed then the node first checks to see if the key falls in the range covered by the leaf set. If so, it is forwarded to the closest node in the leaf set. Else the routing table is used to route the message to the node that shares most of the first digits with the target.

**Data and topology updates management**
Pastry can handle both data insertion and deletion, as well as machines leaving and joining, in *O(logN)* time.

### 3.1.4 Tapestry

The fourth P2P application discussed here is Tapestry [36]. Tapestry is in some ways quite similar to Plaxton [41]. A Plaxton mesh is a distributed data structure which is optimized to support a

network overlay for locating and communicating with named data objects. Tapestry however provides improved adaptability, scalability and fault-tolerance (availability) compared to Plaxton.

**Naming and structure**
Each node is assigned a unique nodeID, which are uniformly distributed in a 160-bit SHA-1 [17] identifier space. In Tapestry each node stores a neighbor map consisting of $log_bN$ levels, with each $b$ entries per level. In other words, for each digit $i$ (level) in a nodeID we store entries that point to a nodeID for which digit $i+1$ is one of $b$ options.

**Locating and routing**
When a object needs to be found first a hash function is used to get the objectID of the target. Routing is done by continually hopping one digit closer to the destination. NodeIDs are read from right to left, so a route one could take would for example be ***7 -> **37 -> *437 -> 6437. Routing can thus be accomplished in *O(logN)*.

**Data and topology updates management**
Inserting of data can be accomplished in *O(logN)* time. Since there can be multiple copies of an item the deletion of data takes $O(log^2N)$. Inserting or deleting nodes can also be accomplished in *O(logN)*.

### 3.1.5 Summary
All four of the lookup algorithms described above use a distributed hash table as a foundation. In section 2.3 hardware usage constraints for P2P networks were discussed. The two main constraints are the cost of communication (bandwidth usage) and storage costs. The implementation of the routing and storage layer determines which of the two is considered more important.

For example one could design a system in which each peer knows how to directly contact any other peer. Of course the routing table would be enormous, it would grow as *O(N)*. So the storage costs would be very large, while the communication costs would be very small. The other way around, very small storage costs and very large communication costs, would also not be a very good choice.

Instead most implementations, including the four discussed above, choose to maintain routing tables that grow with *O(logN)* and the communication costs also grow with *O(logN)*. For Chord, Pastry and Tapestry this balance cannot be easily altered. The CAN network however has storage costs of *2d* and data retrieval costs of $O(N^{1/d})$. By choosing $d = (logN)/2$ both the storage and communication costs grow as *O(logN)*.

## 3.2 P2P Information Retrieval Systems
In the last few years P2P file sharing applications have become very popular. This success has however not yet been repeated for other P2P information sharing applications. In this section several research projects on the topic of P2P information retrieval systems are discussed.

### 3.2.1 ALVIS

ALVIS [9] is a large research project funded by the EU in which several institutes, universities and companies in the private sector take part. The project's grand lasted three years, from 1/1/2004 till 31/12/2006. During this period the ALVIS consortium developed a prototype of an open source distributed, semantic-based search engine.

**Network topology (DHT)**

ALVIS uses a improved version of the Content Adressable Network (CAN), which was discussed in section 3.2.1. By using eCAN the logical routing cost is improved to *O(logN)*. Furthermore eCAN chooses routes that are close approximations of the underlying physical topology of the internet.

**Indexing strategy**

The metadata they use is produced by fully automated analysis of the content instead of using the more common coded or semi-automatically extracted metadata. Instead of sharing large posting lists the ALVIS project utilizes an indexing method based on highly discriminative keys. A highly discriminative key is a term, or set of terms, which is globally rare to the document collection. In this way you end up with an index that is more like the index in a book, quite selective and compact, instead of posting lists that contain enormous amounts of redundant information. Most users would only be interested in the top-k results.

The main disadvantage of using term sets is that query terms in ALVIS need to be mapped to keys. In ALVIS this problem is solved by using a simple query mapper and a more advanced method based on distributional semantics. The simple query mapper just tries to find the keys that best match part of the set of terms in a query. The more advanced method uses distributional semantics, which basically looks for semantically related terms to extend the set of query terms. For example if the query is “car windshield” then the query map be extended to “(car OR automobile) AND windshield”. When a query is extended to more terms the likelihood of finding keys that can be mapped to the query terms is increased.

**Scalability and retrieval quality**

By using a approach called Highly Discriminative Keys they achieve a index growth which is linear with respect to the collection size. Retrieval quality (top-k precision) is also comparable to a single term TF.IDF approach.

### 3.2.2 Minerva

Minerva [5], named after the Roman goddess of crafts and wisdom, is a P2P information retrieval system in which the peers each maintain a local database and a local search facility.

**Network topology and overlay networking**

Minerva uses a Chord-style overlay network that is based on a distributed hash table (DHT). In a Chord network each node only needs to store information about *O(logN)* other nodes. Furthermore all lookups are also resolved in *O(logN)* time. So Chord, like most other lookup services that are based on a distributed hash table, provides excellent scalability.

**Indexing strategy**

Peers may share parts of their local index by posting meta-data to the P2P network. This meta-data consists of statistics and quality-of-service information. Minerva maintains this conceptually global, but physically distributed, directory on top of a Chord-like distributed hash table (DHT). Responsibility for a term is shared and replicated among several peers for improved resilience and availability.

ALVIS takes a somewhat similar approach to Minerva [5]; both use a P2P overlay network which contains metadata about the information stored at the peers. Both indices are physically distributed but conceptually global.

**Scalability and retrieval quality**
When a query is executed the peers that are responsible for terms in the query are looked up and the PeerLists are retrieved. For efficiency reasons the query initiator can also choose to just retrieve for example the top-k peers. Using this information the most promising peers are asked to perform the query and the results are eventually combined into a ranked list using the meta-data.

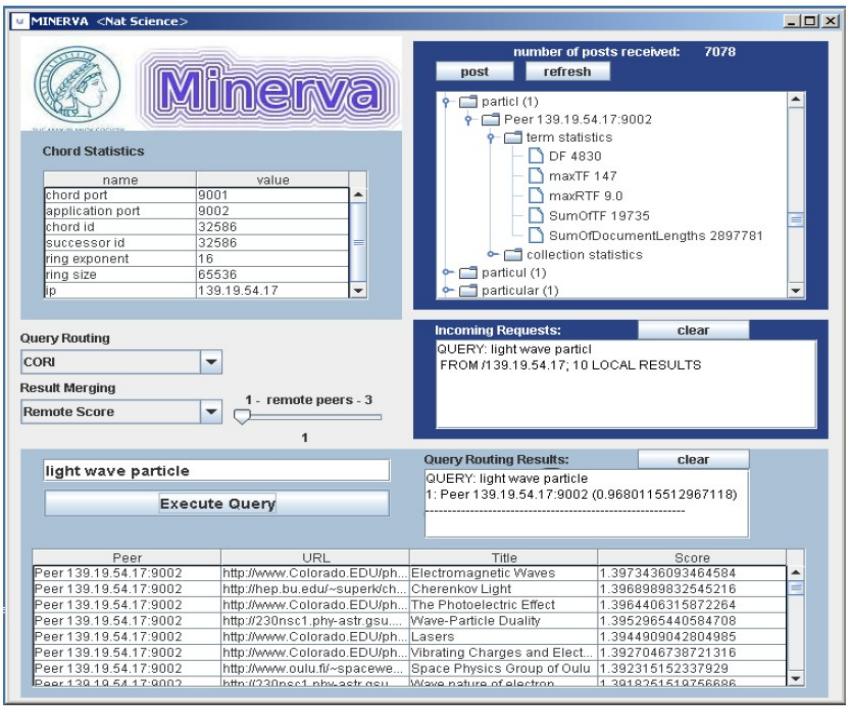

Figure 3.4: The Minerva GUI

### 3.2.3 PlanetP
PlanetP [42] is P2P information retrieval system designed for sharing large sets of text documents between the peers. It was developed at Rutgers, the State University of New Jersey (USA) as a research project into distributed information retrieval.

**Network topology and overlay networking**
PlanetP takes a somewhat different approach then ALVIS or Minerva. Instead of storing a conceptually global index in a physically distributed manner, PlanetP replicates the global directory and a compact summary index at every peer. So each peer stores the names and addresses of all the other peers together with a Bloom [22] filter (see also section 2.4) per peer-entry that summarizes the set of terms that are present in that peers local document collection.

**Indexing strategy**
PlanetP uses a global single-term-to-peer index with federated local indices, just like Minerva. However instead of storing the index in a distributed manner like Minerva does, PlanetP replicates the global index at every peer. The copies of the index are kept uptodate by gossiping between the peers about updates in the network. The information that is being gossiped includes the joining of a new member, a change in a Bloom filter and the rejoining of a previously offline member.

**Scalability and retrieval quality**
Both recall and precision are very close to the performance of a TFxIDF approach that has access to the full inverted index and the word count. PlanetP is also able to scale quite well up to several thousands of peers. After that the index becomes too large to download in a reasonable amount of time for a peer that wants to join the network but has a limited amount of bandwidth available. Although the Bloom filters are a very efficient way to store a set of terms, their sheer number makes the index too large in the end.

### 3.2.4 pSearch
Another interesting P2P information retrieval system is called pSearch [6]. The system supports content- and semantic-based full-text searches.

**Network topology and overlay networking**
pSearch tries to combine the scalability of DHT systems (like CAN) and the accuracy of advanced IR algorithms. Two of the algorithms they use are pVSM (P2P Vector Space Model) and pLSI (P2P Latent Semantic Indexing). The system is actually built on eCAN [43], a hierarchical version of CAN that improves on CAN's logical routing cost to *O(logN)*. ALVIS, which was discussed in section 3.3.1, also uses eCAN.

**Indexing strategy**
pSearch uses two indexing algorithms pVSM and PLSI. VSM represents both queries and documents as term vectors. The weight of an element (term) is often calculated using the *term frequency * inverse document frequency (TF * IDF)* scheme. When a query is executed the query vector is compared to document vectors. Those vectors that are the most similar to the query are returned.

LSI tries to correct problems like synonymy, polysemy and noise in documents. By using singular value decomposition (SVD) semantic relationships can be discovered. For example the words car, vehicle and automobile are semantically quite similar while the words car, toothpick and festival

are not. LSI can transform a high-dimensional term vector into a medium-dimensional semantic vector by discovering the semantic relationships.

**Scalability and retrieval quality**
The pSearch system manages to achieve performance levels that are very close to the non-distributed versions of the algorithms. For example, when comparing pLSI and LSI we find that pLSI only needs to visit 0.4-1.0% of the nodes to achieve 95% of the accuracy of LSI. The system also seems to scale well, as both storage and communication costs do not grow exponentially.

### 3.2.5 Comparison

In the last few sections we discussed some of the most well-known distributed information retrieval systems. Here we compare the four systems to each other to see what they have in common and where they differ.

**Network topology and overlay networking**
Both ALVIS and pSearch use a modified version of the Content Adressable Network (CAN) as their routing and storage layer. Minerva also uses a DHT-based overlay network, but theirs is based on Chord. In fact of the four systems discussed above PlanetP is the only system that takes a different approach. Instead of distributing the index over the peers using a distributed hash table (DHT) the global directory and a compact summary of each peer is stored at each peer. A Bloom filter is used to highly compress the data.

**Indexing strategy**
Again all systems except for PlanetP follow a similar strategy. ALVIS, Minerva and pSearch distribute their index across the network. Although the granularity of the index differs the end results are quite comparable.

**Scalability and retrieval quality**
All systems offer retrieval quality that comes close to a centralized version of the same algorithm. PlanetP is the only system that has serious scalability issues because of the way in which it was designed. In the PlanetP system each peer stores a copy of the global directory and a compact summary of each peer. By using a Bloom filter to highly compress information the system is still able to scale reasonably to several thousands of peers.

The other three systems all use a distributed hash table approach to store and retrieve information. The use of a DHT seems to be almost ideal. Most DHT-based systems balance the cost of storage and communication evenly. Both costs grow with $O(logN)$ so peers only have to store relatively small routing tables and they can still reach any node in the network in $logN$ steps. If there are peers in the network that are willing to offer more services (bandwidth and storage space), this can even be improved upon further. By using a hybrid network in which some peers offer larger routing tables the other peers can find each other more quickly by using the services of those peers.

# 4 Design of the proof-of-concept application

In the previous chapters the theoretical background of P2P networks was discussed, followed by a chapter on related work which discussed P2P file sharing networks as well as distributed information retrieval applications. The ALVIS project was among the distributed information retrieval applications that were discussed. One of the more interesting parts of the project is their indexing approach, which utilizes a concept called 'highly discriminative keys' or HDKs. In this chapter the design for a proof-of-concept application that examines HDK indexing will be presented. Our proof-of-concept application is called the Term Set Indexer (TSI).

## 4.1 Introducing Highly Discriminative Keys (HDKs)

An index entry consists of a term (or a set of terms) and a list of documents or peers where these term (sets) occur. This list of documents or peers is also known as a posting list. So there are basically four options for an index:

1. Global single-term-to-document P2P index.
2. Global key-to-document P2P index.
3. Global key-to-peer P2P index with federated local indices.
4. Global single-term-to-peer P2P index with federated local indices.

These four options were previously explained in section 2.6. The granularity of an index was determined not to have much influence on its scalability. However the extra messages that would have to be send to the peers means an extra delay while the local search on those peers is performed. Therefor a single-term-to-document or a key-to-document index would probably be a more efficient choice.

In a single-term-to-document index there are a few major problems:

1. Since queries often consist of multiple terms we need to combine the results from several keywords. As we do not know beforehand in which documents most of the query terms can be found we need to retrieve the entire posting list for each term.
2. Even if some of the query terms occur in the same document we do not know if they are used in conjunction with each other, like for example in the same sentence or paragraph.
3. There is no limit on the terms that are indexed, nor on the length of the posting lists which are stored for them.

The TSI uses a key-to-document index which tries to solve some of these problems. Sets of terms that occur in a window of words are created, which may answer a query more directly. Furthermore TSI also limits the amount of term sets that are stored in the global index by only storing entries that are a rare combination of non-rare (sets of) terms.

## 4.2 Preprocessing the documents

Before the sets of terms can be generated the data needs to be preprocessed. Preprocessing consists of three steps:

1. **Converting HTML to text:**
   The second step in preparing the document collection is the conversion from HTML to text files. The org.java.util Java Utility Library contains a host of useful classes, including

an HTML to text converter. This converter performs the three following operations on the test data:

1. Strips embedded HTML tags
2. Converts HTML entity codes to appropriate Unicode characters. For example, the string "&" is converted to "&".
3. Converts certain Unicode characters in a string to plain text sequences.

2. **Removing stop words:**
   In every document collection there are a number of words that occur extremely frequently but at the same time are not unique for a specific document at all. Since nearly every document contains these words it would be better to remove them as they will certainly not be used by the indexer. Some obvious examples of stop words in the English language are "a", "of", "the", "and", "it", "you" and "I".
3. **Applying a stemmer:**
   In an effort to further remove any noise from the text we use a stemmer. Stemming is the process of reducing inflected or derived words back to their stem, base or root form. In some cases the stem is not identical to the morphological root of the word, but this doesn't matter as long as related words have the same stem. For example a stemming algorithm for the English language would identify the words "stemmer", "stemming" and "stemmed" as having the same root "stem".

   The most well known stemmer for the English language is the stemming algorithm developed by Martin Porter et al [44]. It was published in July 1980 and it became the de-facto standard stemming algorithm for the English language. Over the years many implementations were developed but a lot of them contained one or more errors. In the year 2000 Martin Porter himself released an official implementation, which was later ported to a number of programming languages [45].

   In our application we use the Java version of the official implementation by Martin Porter. The use of the stemmer further limits the amount of noise in the document collection so the indexer can do its work more efficiently. This final step finishes the pre-processing of the data, so the next step is for the indexer to process the data.

## 4.3 Creating the Highly Discriminative Keys Index

The first step in creating an index is for a peer to create sets of terms as they occur in the local document collection. When a user poses a query he would probably expect the query terms to occur near each other in the document, like in the same sentence or paragraph. So the sets of terms that are created by the TSI always occur in a window of a certain number of words. Furthermore the number of terms in a set of terms is limited. For example, suppose we have a document that consists of just the following words: A B C D E. And we would like to generate all sets of terms in a window of four words, with a maximum of three terms per set. Then in the first window (consisting of the words A B C D) we generate the following sets:

- A
- B
- C
- D

- A B
- A C
- A D
- B C
- B D
- C D
- A B C
- A B D
- A C D
- B C D

Now for each following window (like B C D E) we only have to generate the sets that are a combination of the last word in that window (E) and some of the sets we previously created. Those sets are limited to two terms and they should not contain the word A, as it does not exist in the current window. So that would be the following sets:

- E B
- E C
- E D
- E B C
- E B D
- E C D

In total we just created (4+9+7=) 20 sets of terms. Of course not every set of terms is of equal value, so we split the sets of terms into four different classes:

1. *Kw*, the set of keys that occurs in a window of size w.
2. *Knon-rw*, like the first class, but these keys are classified as none rare.
3. *Krw*, like the first class, but these keys are classified as rare.
4. *Kirw,* like the first class, but these keys are classified as intrinsically rare or i-rare.

The relationship between these different sets of keys is illustrated in Figure 4.1. As one can see the hierarchy of classes limits the amount of keys to a much smaller number, as we are really only interested in the keys that are a member of the class *Kirw*.

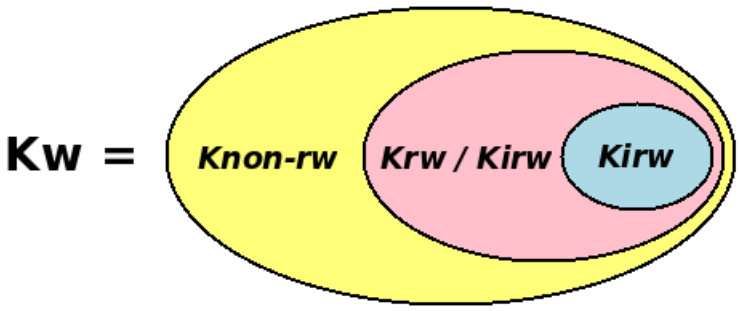


Figure 4.1.: The relationship between non-rare, rare and i-rare keys

At the start of this section we already discussed how we limit all the keys (*K*) to all the keys that appear in a window of a certain size (*Kw*). The remaining keys are then split into two groups, namely rare and non-rare. A key is locally rare if it appears in a no more then *DFmax* documents. The value *DFmax* in TSI is limited to a certain percentage of the number of documents. For example, if a peer has a 200 documents in its local document collection this value could be the minimum of 10% of that (=20 documents). Since every peer has a different number of documents, we tend to use a percentage instead of a fixed value. This percentage is set during experiments between 7,5 and 20 percent.

So now the keys are divided into two classes, namely rare and non-rare. The final set of keys that will be used to update the global index can now be determined by splitting the set of rare keys in two parts. Rare keys are either intrinsically rare or there are not. A key is intrinsically rare if the key itself is rare, but all of its subsets are not. For example, the key A B C is intrinsically rare, or i-rare for short, if the subsets A, B, C, AB, AC and BC are all non-rare. We will call all sets of keys that have a certain number of terms a level. The keys on level one consist of only one term, so here we have a special case, as all rare keys are also i-rare. On higher levels filtering out keys that are not i-rare however removes a lot of redundant keys.

The easiest way to generate all keys in *Kw* is by generating keys on a higher level by building them by using subsets of lower levels. For example, we could build A B C by combining A B and C. Once we have all keys in *Kw* we could split these keys into the classes non-rare and rare. We can then further split the class rare into the classes i-rare and non-i-rare. Unfortunately the easiest way is also the least efficient to build the set of i-rare keys, because less then 10% of the keys in *Kw* are also in *Kirw.* The 10% here is just a rough, but realistic, estimate to indicate the amount of keys that are left after filtering. This is also illustrated by a typical example further on in this section.

Instead the prototype uses an approach in which the filter process is done level by level at the earliest opportunity. This approach is somewhat more costly in terms of time but the memory usage of the application improves dramatically. We will now discuss the algorithm that is used in the proof-of-concept application. The algorithm shown here creates sets of terms up to level three, so with a maximum of three terms per set. The application calculates the *Kirw* in four steps:

1. **Level one:**
   a) Create *Kw* on level one.
   b) Filter level one *Kw* by throwing out all keys that belong in *Knonrw* so only the keys in *Krw* remain.
   c) Create *Kw* on level two.
2. **Level two:**
   a) Create *Kw* on level two.
   b) Filter level two *Kw* by throwing out all keys that belong in *Knonrw* so only the keys in *Krw* remain.
3. **Level three:**

a) Create the term sets on level three, but only store the keys for which all the subsets are non-rare. The set created here thus consists of keys that are possibly i-rare, based on the subsets they contain.

4. **Final filtering step:**
   a) Filter level two, by throwing away non-i-rare keys.
   b) Filter level three, by throwing away keys whoose subsets are all non-rare but are themselves not rare.

The combination of all the filtered sets on each of the levels is now equal to the set that one would get by first creating all keys on all levels and then filtering out non-i-rare keys. The four-step algorithm described above is however far more efficient because a lot of keys are non-i-rare. To illustrate this we will the figures involved with processing a certain document. Certain conditions like the number of (unique) words and the settings for the window size and *DFmax* are ofcourse important. However we just use realistic rough estimates here, as this illustrates the point well enough.

| **# of terms** | **# of keys in *Kw*** | **# of keys in *Kirw*** | **% of *Kw* in *Kirw*** |
|---|---|---|---|
| **1** | 5.000 | 4.000 | 80% |
| **2** | 200.000 | 40.000 | 20% |
| **3** | 1.000.000 | 5.000 | 0,5% |

Table 4.1: An example of the effect of filtering on the amount of multi term keys.

So in this case the easiest (and least efficient) method would first generate 1.205.000 keys and then filter out all non-i-rare keys, to be left with 49.000 i-rare keys. The more efficient four-step algorithm that was discussed above however only generates a modest amount of keys on top of the 49.000 i-rare keys. In the fourth and final step these redundant keys on levels two and three are filtered out.

## 4.4 Updating the global key-to-document index

When a peer has created the set of i-rare keys it is ready to present the keys to the global index and if needed upload the posting list for that key. The index can be stored by using (for example) a Distributed Hash Table (DHT) as discussed in section 2.5. In the case of TSI a centrally stored index is used, but the principle is the same.

Each entry in the index consists of three values:

1. Key; the set of terms.
2. Global document frequency; the number of documents the key appears in globally.
3. Posting list; a list of documents in which the key appears. These documents are stored on one or more of the peers that contribute to the index.

A key stored in the index can become globally non-rare if it appears in more then *DFmax* documents globally. If a key becomes globally non-rare the index will clear the posting list associated with the key, but the key itself and the number of documents it appears in will be kept.

For each locally i-rare key a peer contacts the index to check if the key would (still) be globally rare if its posting list was added to the index. The index increments the global document frequency of that peer by the number of postings that is offered by the peer. The index then gives on of the following replies to the peer:

1. The key is already globally non-rare, don't send your posting list.
2. The key just became globally non-rare, don't send your posting list.
3. The key will remain globally rare, even if your posting list would be added. Please send your posting list.

Only if the peer gets the third answer it sends its posting list. In a distributed environment the global document frequency of a key may change between the time when a part of the index sends a positive reply and the time when the posting list arrives from the peer. If the document frequency has become higher than DFmax such that the key is no longer globally rare, then the posting list should not be added. So before adding the list another check should be made by the index.

## 4.5 Execution of a query

Retrieval also poses a few problems. Given a query Q which consists of {t1, t2, .., tn} terms we need to find the most relevant keys in the HDK index. Unfortunately this is not a trivial task at all. For example, if we have the query A B C then how can we map that set of terms to a number of keys such that the retrieval performance is maximized while the amount of network traffic needed is minimized. If we only try to retrieve a key A B C then we may not get any results at all, since the key may not exist. However if we retrieve all posting lists associated with keys containing at least one of the terms (A, B or C) then we will likely get a lot of results, but the amount of network traffic would be very high. Two possible ways to tackle this problem will be briefly discussed here. In our experiments we only use the first method, namely simple query to key mapping.

The first method tries to solve the problem using a simple mapping. The posting lists for keys that are a subset of the query are retrieved and combined. For example, a query 'A B C', for which no HDK 'A B C' exists, could probably be best answered by the HDK (if it exists) for the subsets of this key: AB, AC, BC, A, B and C. Each of the end results will have contain all three of the terms. This is the method that is used in our experiments.

The second method is known as distributional semantics [46] [47]. In this approach a co-occurence matrix is computed between the terms in the query and all the terms in the documents. Two words co-occur if they are both found in a window of a certain number of words. Terms that occur often with other terms in the vocabulary are more likely to be closely related or even synonyms. For $N$ words a $N*N$ matrix of co-occurence values would have to be calculated. In a very large collection we can assume that certain statistics like the co-occurence of words are about the same as for a sizable subset of that collection. Heaps law [26] basically states that the larger the text, the less new words we will encounter. From this emperical law we can deduce that the larger the amount of text, the less new co-occurences we will encounter.

For example the words car and automobile are synonymous, so their co-occurence value will probably be high. Other words that are related to the word car, like windshield or tire will also co-

occur quite often. In some cases a pair of words (like 'car' and 'the') can have a high co-occurence value while they are not semantically related. In such cases the other word ('the') just occurs a lot in general, so it will also occur a lot in combination with the word 'car'. This can be fixed by either filtering out words that have consistently high co-occurence values from the co-occurence table, or by adding them to the list of stopwords if needed.

# 5 Experimental evaluation

To illustrate the feasibility of an index based on highly discriminative keys experiments were performed on a test collection. In this chapter the setup of these experiments and the results are discussed. The topic of this thesis is the feasibility of distributed web search applications. In chapter two it was determined that the feasibility of such a system mainly depends on its scalability. Scalability alone is however not enough since one also needs to achieve a reasonable level of retrieval quality. In our experiments we compare the scalability and retrieval performance of the HDK indexing approach to a single term index. The test collection and the results of these experiments are presented in the following sections.

## 5.1 Test collection

The test collection that is used to test the indexer is the WT10g collection [8]. This collection closely resembles the characteristics of standard web pages. The collection consists of ten gigabyte of HTML documents from 11.680 different servers. A server contains about 144 documents on average and a minimum of five documents. In total the collection contains about 1.7 million documents. Before the collection could be indexed several steps where needed which will be explained in the following sections.

The documents in the WT10g test collection are distributed randomly over more than 5000 files. To simulate real web servers the documents in the files needed to be resorted based on their IP address. Fortunately each document contains a header with meta-data, including the IP address. An application was written to process all the documents and to store them in separate files based on their address. After resorting we are left with 11.513 files. The difference in the number of servers (11.680) and the number of files (11.513) is most likely caused by servers that serve multiple websites from the same IP address and multiple domain names that are mapped to the same IP address.

From these files a selection was made of files that were about one megabyte in size and which contain one average about 200 documents. All of the test files were preprocessed; HTML code was stripped, about 300 common English stop words were removed and the Porter stemmer was applied. A maximum of three terms per key is used, since the calculation of higher levels becomes increasingly more expensive in terms of the computational load. In section 4.3 we explained that the number of possible term sets increases drastically with the number of terms. In the realistic example given in that section we showed that on level three less then one percent of the keys are in *Kirw*, while a million term sets are created. This trend continues for higher levels, therefore there would only be a few keys on level four.

The Term Set Indexer (TSI) is implemented in Java. The JVM used is version 1.5 (5.0) by Sun Microsystems. All of the calculated term sets and index entries are kept in memory, to speed up the process. All experiments are performed on a single laptop, running Kubuntu Linux 7.04 (Feisty Fawn). The machine is equipped with a dual core processor (with each core running at 1733Mhz) and 1536MB of main memory.

## 5.2 Scalability

The scalability of a distributed information retrieval application depends mainly on the use of disk space and the use of network bandwidth. The use of disk space can be nicely distributed by using a Distributed Hash Table (DHT). The amount of network traffic needed is however mostly determined by the size of the posting lists that need to be send in response to a query. We assume that the amount of network traffic that occurs during the indexing process is less relevant since it occurs very infrequently. Furthermore indexing is not time critical, while an answer to a query has to be delivered as quickly as possible.

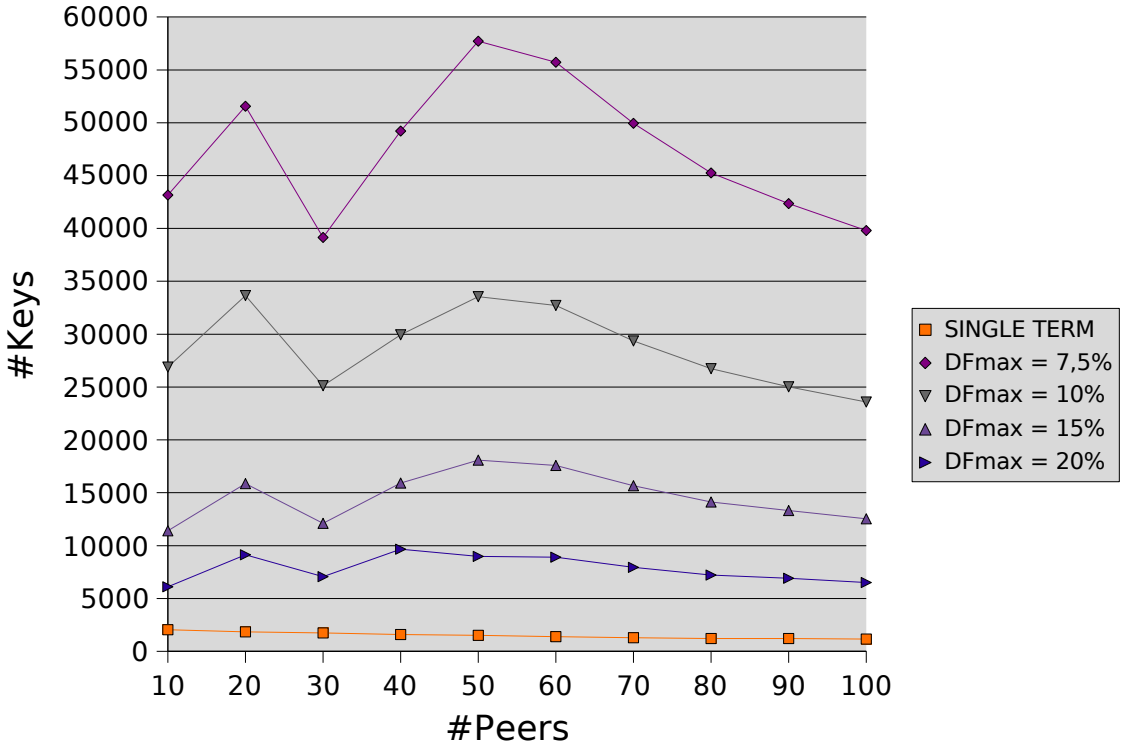


Figure 5.1: The average number of keys per peer

Figure 5.1 shows the average number of keys per peer for various settings of *DFmax* and for the single term index. The average number of keys per peer slowly decreases for large numbers of peers. This is caused by the overlap in keys between peers, so a key can occur on multiple peers. Initially the average number of keys fluctuates a bit because only a small amount of the keys are 'shared'. As the number of peers increases more keys are shared. For very large numbers of peers the average will grow increasingly slower. This is caused by the fact that for large numbers of peers it becomes less likely that we still discover new keys, that were not present in the previously processed peers.

The figure also shows the effect of different values for *DFmax* on the average number of keys per peer. A lower percentage will cause more keys to be intrinsically rare (or i-rare) so the average is also higher. A higher percentage will thus cause less keys to be i-rare. A multi term index with a higher percentage will therefore approach the results for the single term index.

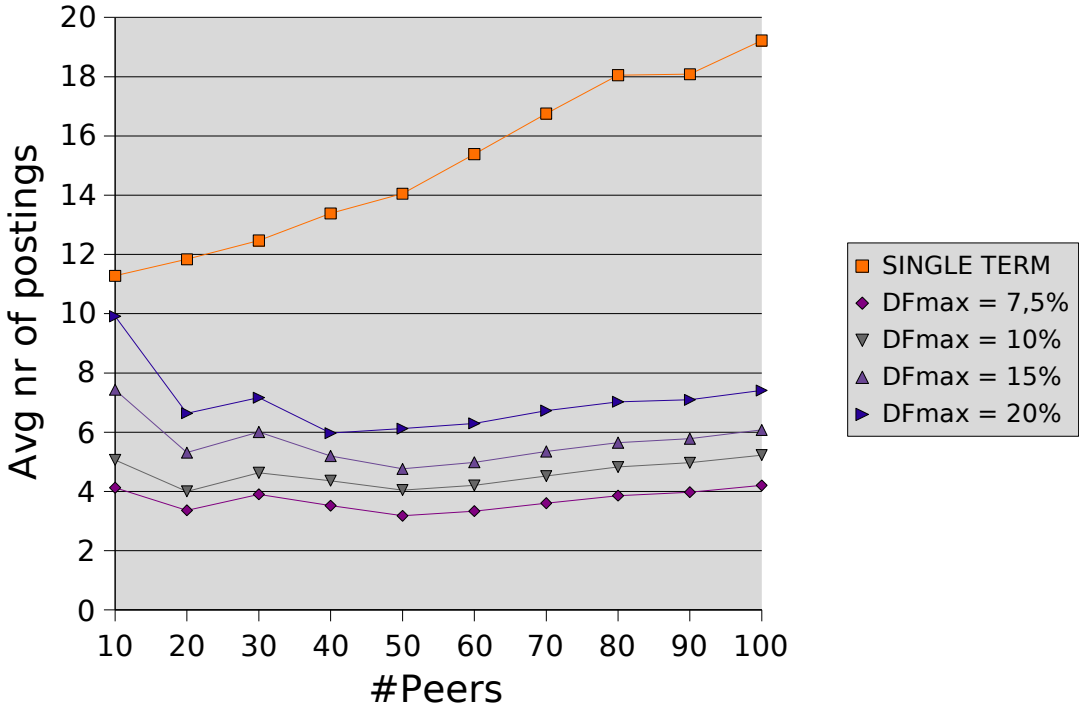


Figure 5.2: Average size of a posting list

Figure 5.2 shows the average size of a posting list. A posting list is a list of documents that corresponds to an index entry. An index entry can either be a term of a set of terms. The figure shows that the average number of postings per key for a multi term index does not fluctuate much.

The single term index on the other hand knows no bound for the number of postings per key so it continues to increase with the number of peers. So the multi term index displays no growth for the average number of postings per key, while the posting list size for a single term index grows linearly with the number of peers.

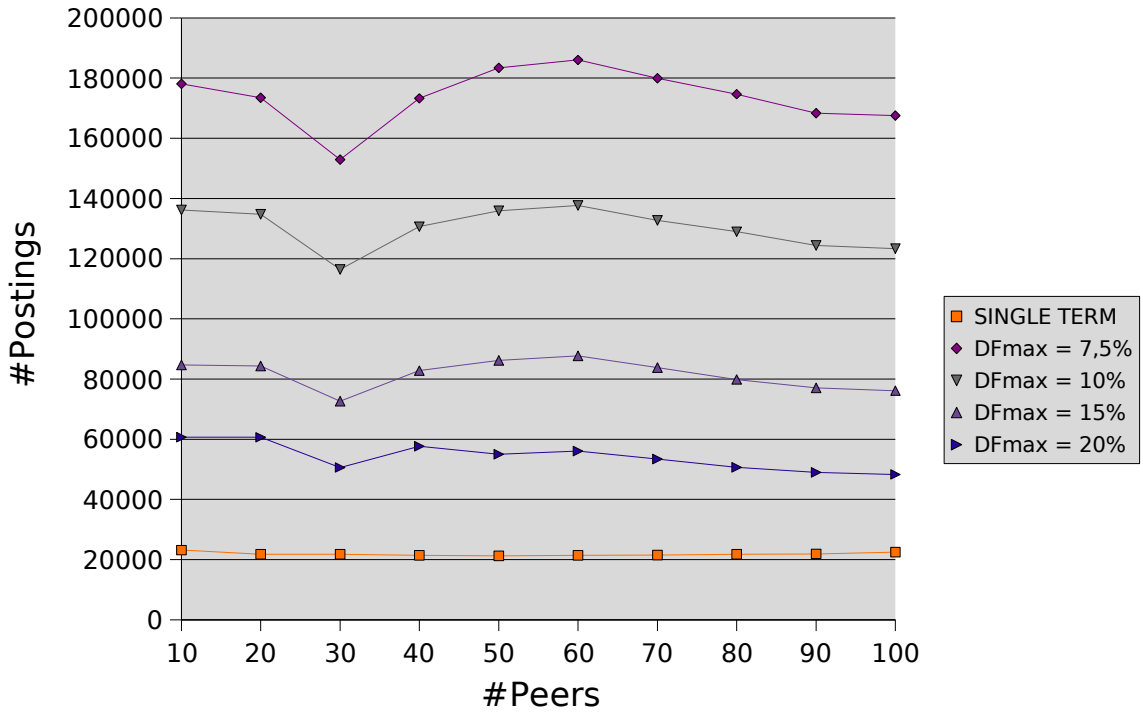


Figure 5.3: Average number of postings per peer

Figure 5.3 shows the average number of postings per peer, again for different values of *DFmax* and for the single term index. The average number of postings per peer for the single term index will display a small but linear growth for large numbers of peers. In Figure 5.3 there appears to be (almost) no growth because the decreasing average number of keys per peer is mostly canceled out by the growth of the average posting list size, which exhibits a linear growth in relation to the number of peers. For larger numbers of peers the average number of postings per peer shows a small constant growth, as is visible in Figure 5.3 for more then 50 peers. This small constant growth is mainly caused by new postings being added to existing keys. For a large, but still growing, document collection we will discover less and less new keys (words) because the dictionary for the English language is limited. In linguistics this is known as Heaps law [26].

As expected the average number of postings per peer is higher for the multi term index with a lower value of *DFmax*. The average number of postings per key for a multi term index doesn't grow as the number of peers increases (see Figure 5.2), so the difference here can be totally attributed to the average number of keys per peer.

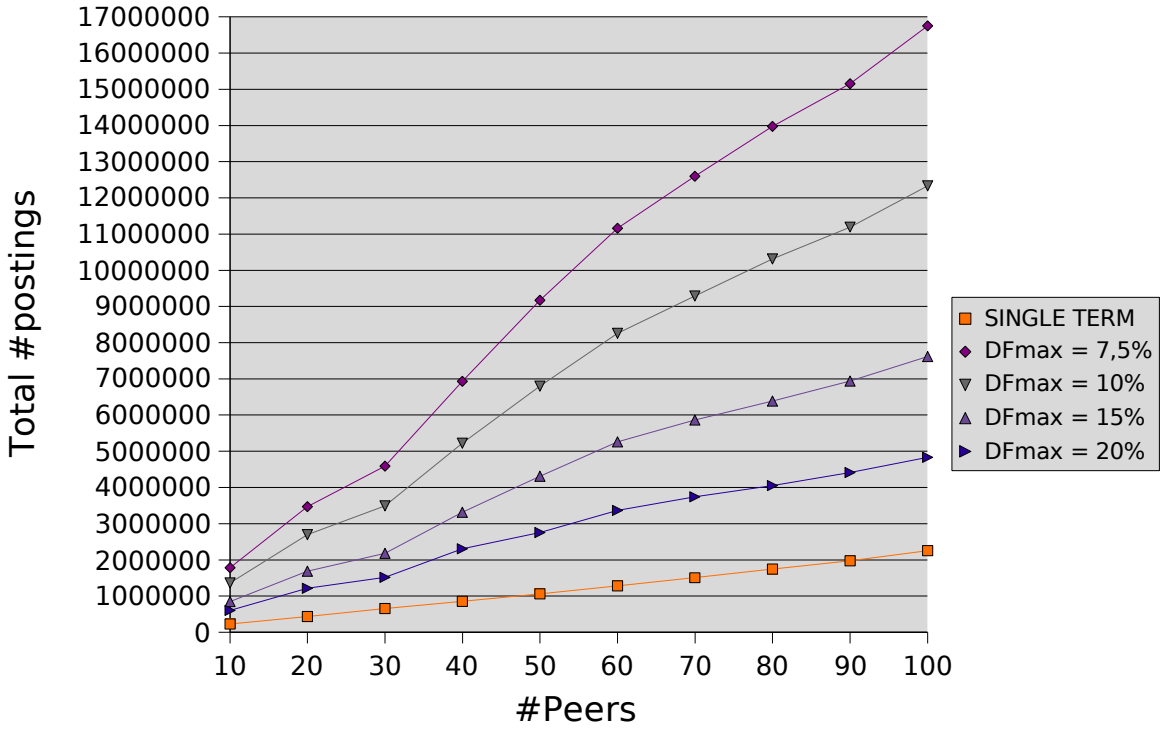


Figure 5.4: Total number of postings in the index

Figure 5.4 shows the total numbers of postings in the entire index. For both the single term index and the multi term index we saw that the average number of keys per peer will level off to a constant. This is caused by the fact that the discovery of new keys will become less likely. The total number of postings is a product of the following values:

- Number of peers, this value will grow linearly.
- Average number of keys per peer, this value will settle around a certain average for each index (see Figure 5.1). For very large numbers of peers this value will grow very slowly.
- Average number of postings per key, this value will increase linearly with the number of peers for the single term index (see Figure 5.2). For the multi term indexes we observed that this value settles around a certain number depending on *DFmax*. In other words, it shows no growth.

So to summarize, we can conclude that the growth rate of the total number of postings in the index is determined by the average number of postings per key. This means that the single term index will grow linearly for large numbers of peers. Increasingly less new terms will be discovered, but new peers will add postings to existing terms for their documents.

The growth rate of the multi term index will also grow like the average number of postings per key. The average number of postings per key for a multi term index levels off so the total number of postings will also level off eventually.

## 5.3 Retrieval performance

In distributed information retrieval systems there needs to be a balance between the scalability of a system and its retrieval performance. A multi term indexing system may scale well, but it should also provide good retrieval performance. In the ALVIS project the retrieval performance of their indexing method is tested by comparing the top-20 overlap between results from a single term index and the multi term index. They use a simple TF-IDF implementation to rank the results. In this thesis we also compare the top-20 overlap however there are also a few things that were done differently:

1. The results were ranked using the Okapi BM25 ranking function, instead of a simpler TF-IDF ranking. BM25 is considered to be a more advanced ranking function. The implementation we used will be discussed in Section 5.3.1.
2. Instead of using just one set of queries we've experimented with different kinds of queries to discover more of the strengths and weaknesses. The results of these experiments are discussed in Section 5.3.2.

### 5.3.1 The Okapi BM25 Ranking function

The ranking function used here is based on the probabilistic model. It is often referred to as Okapi BM25 [48], because the Okapi information retrieval system was the first to implement this ranking function. BM25 is, despite its name, not really a single function since different implementations use different components and parameters. The implementation used here is the following:

$$Score(D,Q) = \sum IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot 1 - b + \frac{b \cdot |D|}{avgdl}}$$

The variables in the equation above are the following:

- $f(q_i, D)$ is the frequency of a query term in a document $D$.
- $|D|$ is the number of words in a document $D$.
- $k_1$ and $b$ are free parameters, chosen here as $k_1 = 1.2$ and $b = 0.75$, which are common settings.
- $IDF(q_i)$ is the inverse document frequency weight of query term $q_i$ where $IDF(q_i)$ is defined as:

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

Here the variable $N$ is the total number of documents in the collection and $n(q_i)$ is the number of document that contain the query term $q_i$.

### 5.3.2 Experimental results

Like the ALVIS project we measure the retrieval performance by comparing the results from the multi term index to those of a single term index, which we use as a baseline. The result of each index, a set of documents containing the query terms, is ranked using the Okapi BM25 algorithm as described in the previous section. We then calculate how many documents occur in the top-20

of both of the indexes. Different values for *DFmax* where used to illustrate the effect on the retrieval quality. Furthermore the size of the posting lists that are send over the network are also compared. All experiments are executed on an index which consists of 50 peers (listed in appendix A) that contain on average 200 documents each.

First we tried to reproduce the results from the ALVIS project using the WT10g test collection. In the ALVIS project the index was tested with a single set of 200 queries. These queries were constructed by randomly choosing two or three terms from the titles of the news articles in the test collection. We use the following approach to create our first queries:

1. Extract the titles from the HTML documents.
2. Filter out duplicate titles. A number of websites uses the same title for every web page. After this step we are left with 2733 unique titles.
3. Remove stop words and punctuation from the titles.
4. Reduce each title to two or three terms.
5. Filter out the duplicate queries, which were created during the previous two steps.

After this process we are left with 632 possible queries (listed in appendix B) which we then executed on both the single term index and the multi term index for different values of *DFmax*. We first determined the number of queries that return at least 10 or 20 results. These results are summarized in Table 5.1.

| | ***Single*** | ***DFmax 7,5%*** | ***DFmax 10%*** | ***DFmax 15%*** | ***DFmax 20%*** |
|---|---|---|---|---|---|
| ***Top-10*** | 335 | 65 | 59 | 44 | 45 |
| ***Top-20*** | 244 | 39 | 33 | 30 | 34 |

Table 5.1: The number of queries with more then top-k results

The results show that a lot of queries do not return more then 10 or 20 results respectively. We also see that the number of results for a multi term index increases for lower settings of *DFmax*. Choosing an even lower setting for *DFmax* is however not a good option because it increases the amount of keys that need to be stored (see Figure 5.1). Furthermore the average number of postings per key would further decrease, which means the index would store a lot of keys with just a few results (see Figure 5.2).

For the remaining queries we tested the top-k overlap between the single term index and the multi term index. We also determined the total number of postings that need to be transmitted to answer each query.

| | *Overlap ratio (average)* | *#Postings on avg for multi term index* | *#Postings on avg for single term index* |
|---|---|---|---|
| ***DFmax 7,5%*** | 7.5% | 33 | 1795 |
| ***DFmax 10%*** | 6.4% | 38 | 1824 |
| ***DFmax 15%*** | 6.8% | 43 | 1945 |
| ***DFmax 20%*** | 4.0% | 38 | 2033 |

Table 5.2: The overlap and posting lists sizes for queries with more then 10 results

Table 5.2 shows that the average overlap ratio for the top-10 results between the single term index and a multi term index is quite low. In general a lower setting for *DFmax* will limit the size of the average posting list, but it also hurts retrieval performance. On the other hand a higher setting limits the amount of queries with enough results (see Table 5.1). The setting for *DFmax = 20%* seems to be over the top. At such a high setting the maximum size of the posting lists (250 documents) becomes a severe limitation, since there are almost no multi-term rare keys.

The results from the ALVIS project show an overlap ratio of 83 to 94% for the top-20 documents [9]. The difference between these two sets of results clearly show that this indexing method does not perform well at all on the WT10g web collection. A number of differences between the two test collections attribute to this extreme difference. In section 5.4 we will make an extensive comparison between the results from the ALVIS project and this master project.

Table 5.2 also shows the number of postings that are transmitted on average during query execution. The number of postings needed to answer a question for the single term index is quite high compared to the same number for the multi term index. For the single term index the posting list of each term needs to be send to the peer that executes the query. After the list are received the intersection of those separate posting lists is calculated. This intersection is then ranked, after which it forms the final result. The result for the single term index contains literally all the documents in the collection that contain all the query terms.

For the multi term index the amount of postings that need to be send is far lower because those intersections have already been determined beforehand. The query for the multi term index is mapped to one or more keys that contain one or more subsets of the query. For example, the query A B C, could be answered by the keys A B and AC. After the final list of documents is determined the results are ranked.

## 5.4 Comparison with the ALVIS project

The proof-of-concept application in this master thesis is based on the work done as part of the ALVIS project. The goal of the ALVIS project was to develop a open source prototype of a distributed, semantic-based web search engine. As part of this research a new and novel idea for indexing was introduced which uses rare sets of terms. This indexing method was researched by building a proof-of-concept application and running some experiments on it. There are a number of differences between the implementations and the experiments which influence the results.

| | Reuters news corpus (ALVIS) | WT10g sub collection (Master Thesis) |
|---|---|---|
| **Average number of words per document** | 170 | 500 |
| **Average number of documents per peer** | 5000 (randomly distributed) | 200 (forming a website) |
| **Number of peers used** | 6 for the queries<br>16 for scalability analysis | 50 for the queries<br>100 for scalability analysis |

Table 5.3: Differences between the Reuters news corpus and the WT10g test collection.

The main difference is the use of another test collection. The ALVIS project used the Reuters news corpus. In this thesis a subset of the WT10g test collection was used. The differences between these two collections are summarized in Table 5.3. In short the ALVIS project uses a small number of peers (6 to 16) which each contain a large number (5000) of randomly distributed documents that have an average length of 170 words. The WT10g sub collection on the other hand consists of a larger number (50 to 100) of peers, which each contain a relatively small number of documents (200) that together form a website. A web page from this collection has an average length of 500 words. In total the amount of information that is used to run the queries on is however about the same size.

There are a number of factors that contribute to the difference in retrieval performance between the two projects. To start we will discuss the factors with the most influence on the results:

1. **ALVIS uses higher quality queries.** In the ALVIS project the titles of the news articles are used as the basis for their queries. The title of a news article is written by a professional journalist who wants it to be as descriptive as possible. After filtering out stop words only a few highly discriminative words would remain. For example, a news article on the French president Sarkozy visiting president Bush while he's on holiday would likely leave us with the query “Sarkozy Bush holiday”. This means that the queries are already highly discriminative themselves as if they were constructed by an expert user. Queries like “Sarkozy Bush holiday”, that contain one or more proper names limit the amount of results because the number of documents the term occurs in is relatively small. If you then intersect one or more of those small lists the end result will be a relatively small set of results. For a smaller result set relatively more documents appear in the top-k results. If only queries that result in more then a certain number of results (10 or 20) are compared then the top-k overlap will be quite large, because both lists could be almost the same. For example the single term index could return 30 results for a very discriminative query, while the multi term index will find 20 results for the same query. The top-20 overlap ratio now already has to be between 50 and 100 percent.
2. **ALVIS uses higher quality documents.** Not only the quality of the titles (and thus the queries), but also the quality of the articles themselves is much higher then those in the WT10g test collection. The texts are written very concise so the important words (the subject of the document) occur close to another. Words within a certain window of words are marked as possible keys, so this is quite important.

3. **ALVIS uses much smaller documents.** On average a document in the test collection of the ALVIS project contains 170 words, while the test set chosen here contains on average 500 words. Even if a web page containing 500 words is written as concise as a news article of 170 words, then the result is still influenced. For example, a web page main contain three paragraphs on three different subjects. The single term index will record the occurrence of these three terms separately, but the multi term index can only record the combination of terms if it appears in a window of a certain number of words.
4. **ALVIS uses randomly distributed documents.** The rare key indexing method as used here and in the ALVIS project uses only local knowledge to filter keys. If the subject of the documents differs a lot on each peer then a lot of subjects (read: term sets) will not occur that often. On the other hand we could have a website (peer) that is largely about one subject. In such a case the term set would not be considered rare because the term set occurs in a lot of documents locally. However globally the term sets may be considered rare if there are not a lot of websites on the subject.
5. **ALVIS has more documents per peer and less peers in total**. In the ALVIS project the queries are tested on a collection of 30.000 documents, distributed randomly and evenly over six peers. In this master project 10.000 documents were used, distributed as one website per peer with 200 web pages on average. The documents here are on average about three times as large so the total amount of text is roughly the same. If a term set occurs in less then 10% of the documents on a peer, then it can occur in almost 500 documents in case of the ALVIS project. But in this project it can only occur in less then 20 documents on average. Therefore a website cannot contain many pages on the same subject.

To illustrate the influence of some of the factors above we take a look at the three best performing queries for the multi term index with *DFmax = 7.5%*. In appendix B you can find the complete list of 632 queries. The three best performing queries are:

- [172] - “genentech leadership” - 4 out of 10 overlap – 13 MTI, 26 STI - P27D261 P27D127 P27D126 P27D88 P27D97 P27D102 P27D235 P27D197 P27D57 P27D18
- [256] - “kathi keller” - 5 out of 10 overlap – 13 MTI, 16 STI - P1D95 P1D93 P1D97 P1D96 P1D98 P1D88 P1D89 P1D85 P1D83 P1D99
- [532] - “steve hinkl” - 10 out of 10 overlap – 10 MTI, 10 STI - P37D60 P37D104 P37D24 P37D109 P37D97 P37D116 P37D51 P37D110 P37D96 P37D94

Each of the lines above represents:

- The number of the query.
- The stemmed query itself.
- The overlap ratio.
- The number of results for the multi term index (MTI) and the single term index (STI).
- The ranked list of results for the multi term index; P stands for the peer number, D stands for the document number.

There are a few remarks that we can make for these queries:

1. The top two queries consist of someone's name, while the other query also contains a proper name (“genentech”).
2. The number of results for both the STI and the MTI is closer when the overlap is higher.
3. For each query the result list contains documents from just a single peer.

Although we cannot draw hard conclusions from such a select number of queries we however can summarize what they at least seem to confirm:

1. First the highest ranking queries all seem to consist or contain proper names, which seems to confirm factor one.
2. Secondly these terms occur near each other in a lot of documents. For a first and a last name this would be very logical. The occurrence of terms near each other is very important as mentioned as factors two and three.
3. Thirdly there are no more then 13 results for the MTI, which is 6,5% of the average number of documents per peer (200). Apparently these lists of results are just below the 7,5% cut-off line, so the keys are considered i-rare. Factors four (random distribution of documents) and factor five (more documents per peer) should improve retrieval performance. Keys will be considered i-rare more often since the number of documents per peer containing the key is lower (factor four) and there can be more documents containing the key per peer (factor five).

There are also a few other differences between the ALVIS project and this master thesis, but these do not influence the results as much. They are presented here in no particular order:

1. **ALVIS uses distributional semantics to improve their results.** This is an interesting approach to query expansion which goes well with the type of indexing that is used. However here it is not implemented so we do not compare the results here with the results from the ALVIS project that use this method of query expansion. Distributional semantics is discussed in more detail in section 6.2 as a suggestion for future work.
2. **ALVIS uses a ranker based on the TF-IDF algorithm, while the algorithm used here is BM25**. BM25 is a more advanced algorithm then a standard TF-IDF implementation. However here and in the ALVIS project only the top-20 overlap between the single term index and the multi term index are compared. Therefore the other factors are likely of far more influence then the ranking algorithm that was used.
3. **ALVIS uses a network of computers, while here only a single laptop is used.** This means that network related issues like insertion time were not researched.

# 6 Discussion and future work

This master project would not be complete without a discussion and an outlook on future work. First the inherent problems with the methods used by the ALVIS project, and therefore also in this project, are discussed. Secondly we discuss a problem that has caused quite a few difficulties during the implementation phase of our proof-of-concept application, namely the memory requirements to build and test such an index. Finally a few suggestions for future work are briefly explored.

## 6.1 Inherent problems with the comparison

Most of the reasons for the difference between the results from this master project and the ALVIS project were already discussed previously in section 5.4. However there are two other factors which make comparing the two difficult:

1. The high amount of filtering.
2. The ranking algorithm.

The rare key indexing method stores only a small subset of the possible term set combinations that can be made in a single document. This is caused by the two filters that are employed:

1. Proximity filter, which only creates term sets in a window of a certain number of words.
2. Redundancy filter, which filters out keys that are not intrinsically rare (see section 4.3).

This means that a huge amount of possible term sets are either never created (1) or filtered out (2). The advantage of this indexing method should be the smaller amount of postings lists it sends during the execution of a query. Although this is exactly what is achieved we also saw that a lot of queries cannot be answered because the multi term index does not contain enough results. Furthermore the queries that can be answered usually have a very low overlap ratio between the top results from the single term index and those of the multi term index.

Another problem is the ranking function that was used. In the ALVIS project a standard TF-IDF implementation is used to rank the results while as part of this project the BM25 ranking function was used. These kind of functions only consider factors like:

- Length of the document.
- Average length of other documents.
- The number of times a term occurs in a document.
- The number of times a term occurs in other documents.

In other words the position of the terms in the document, their proximity to each other and how often they occur together are not considered. A document that contains a set of terms that are used near each other may be a better answer then one that contains the same set of terms spread out over the document. Therefore the top results from the multi term index may be a better answer to the query then those of the single term index.

## 6.2 Problems during implementation

The feasibility of a distributed index depends on a fine balance between scalability and retrieval performance. Each peer in the system needs to be able to (temporarily store) in the order of a few hundred thousand sets of terms. Based on the memory usage of our application we estimate that a

single index entry needs about 500 bytes of storage space in main memory. So if one wants to store 200.000 index entries you would need a hundred megabytes of main memory. The filtering process (as discussed in Section 4.3) needs quick access to the other entries in the index, so storing the index on a hard drive would severely slow down the process. The amount of keys we need to store depends mainly on the following factors:

- The number and length of the documents.
- The size of the window in which words that form a key can appear.
- The maximum number of documents that can contain a key (=*DFmax*).

An average website in the WT10g test collection contains 144 documents. For such a website a few hundred thousand keys to work with in main memory is a realistic estimate. However websites that are a lot larger need a more efficient storage structure. In the proof-of-concept we therefore store the index as a HashMap<String key, byte[] postings>. The IndexEntry object can be reconstructed from the key and the postings list, but the storage space needed is only about 100-150 bytes. It may be possible to limit the amount of storage space even further, but the index entries will still have to be quickly accessible.

## 6.3 Suggestions for future work

The research done as part of this master project yielded some interesting results. However more research is needed into distributed information retrieval so it can mature. In light of the research done here there a few interesting issues to research:

1. Distributional semantics
2. Query adaptive indexing

The first research area, distributional semantics [46] [47], could be used to improve the query expansion process. When a query is entered it needs to be mapped to one or more keys in the index. In a best case scenario we have a query that is equal to a key. In most cases however the query terms will have to be mapped to different keys.

For example, if we have the query A B C then in absence of the key A B C other keys like A B, A C, B C, A, B and C may be (if they exist) combined to form the list of resulting documents. There may also be other keys that contain a subset of ABC, but that do not solely consist of this subset. An example here would be the key ABX, which contains the subset AB, but also contains X. Now suppose that the query is ABC and the index happens to contain keys like ABD, ABE, ABF ... ABZ. Retrieving the posting lists for all these keys would generate a lot of network traffic, which is costly and also decreases the scalability of the application. Distributional semantics tries to solve this by determining which keys we should retrieve if we want to find ABC. This is done by calculating a co-occurrence matrix of $N*N$ words for a representative subset of the global document collection. In a co-occurrence matrix the amount of times a word like D occurs within a certain window size near C is stored. So to select the best keys we look at which other words occur the most near C. To save storage space you could throw away all references to terms that do not co-occur more then a certain number of times, since we are only interested in co-occurrences that occur often. Using distributional semantics thus increases the chance of success while lowering the amount of network traffic.

In recent years the use of query adaptive indexing techniques for peer-to-peer networks have also become an interesting topic [49] [50]. An index based on highly discriminative keys basically

tries to guess the query a user would enter to find that document. Words in a window of a certain size that also don't occur very often together are combined to form a key. This approach limits the amount of postings that need to be retrieved at the time a query is executed. However the rareness of a key does not seem to be the best indicator of its usefulness. If we could determine the usefulness of keys by looking at previously executed queries then the retrieval performance may improve, while at the same time the storage and therefore the communication costs could be kept down.

# 7 Conclusions

The contribution of this thesis is twofold; first the current state of the field of distributed information retrieval is presented. In the second part an application was presented and evaluated that implemented a recently introduced indexing method based on rare sets of terms (or keys) [9].

**Which systems for distributed information retrieval already exist?**
In the last few years a number of distributed information retrieval systems have been implemented. In this thesis we looked at ALVIS [4], Minerva [5], pSearch [6] and PlanetP [42] which are some of the most well known distributed information retrieval systems. We examined the structure of the network they use, their indexing strategy as well as their scalability and retrieval performance. ALVIS, Minerva and pSearch all use a distributed hash table (DHT) [7] as the basis of their overlay network, which means that also their index is distributed over the peers. These three information retrieval systems all use existing DHT approaches like CAN [33][43] (ALVIS and Minerva) or Chord [34] (pSearch). These DHT systems can reach any peer in the network in *logN* steps, while maintaining an routing table that grows as *O(logN)*.

**What are the differences among them?**
There are no major differences between ALVIS, Minerva and pSearch if we look at the network structure (DHT-based), their indexing strategy (distributed over the peers) or their scalability (quite good, because of the DHT approach). PlanetP is the only system that takes a totally different approach. Instead of distributing the index over the peers the PlanetP system stores a copy of the entire index on each peer using a Bloom filter [22] to achieve a high compression rate. However despite the high compression rate the index still becomes far too large for large numbers of peers.

**How do they achieve a balance between scalability and retrieval performance?**
In order to find a good balance between excellent retrieval performance (which means a large index) and excellent scalability (which means a small index) the more successful systems use a distributed hash table as their basis. This means a distributed index is stored on a number of peers. The number of peers that need to be contacted is limited. In the ALVIS project this is done by mapping the query terms to a relatively small number of keys. Minerva uses meta data to select the most promising peers and pSearch compares the query vector to retrieve the closest matching document vectors. So by distributing the index and querying only a select number of peers the systems remain scalable and achieve good retrieval performance. As said before the approach used by the PlanetP system doesn't scale well so the balance is lost.

**What are the advantages or disadvantages of the approach they use?**
The main advantage of the DHT-based systems is their excellent scalability. Except for PlanetP all of the systems can grow to very large numbers of peers. The main disadvantage of a distributed information retrieval system in general is the cost of communication. Since the information needs to be send over the (relatively slow) internet instead of a (relatively fast) local network the amount of communication needs to be kept down. Without a (near) perfect knowledge of the global document collection a distributed information retrieval system can only approach the retrieval performance of a centralized system. Most systems claim very high

retrieval performance, however these systems really should be tested on a number of collections with a number of query sets to assess their performance. For example, in the case of the ALVIS project our experiments show that their indexing method doesn't perform well at all when it is tested using a realistic collection of webpages (WT10g [8]).

In the second part of this thesis the feasibility of using rare key indexing for distributed web search was researched. This indexing method is based on research which was done as as part of the ALVIS project. Basically sets of terms are considered rare if they occur near each other in a limited number of documents. In our experiments we researched the scalability and the retrieval performance of the indexing method using the WT10g test collection. During these experiments we came to the following conclusions.

**How does the average HDK vocubulary per peer scale?**
An important factor of scalability is the average number of entries per peer. The scalability analysis of our implementation shows that the average number of keys per peer for both the single and the multi term index will grow increasingly slower. For large numbers of peers the discovery of new (sets of) terms becomes less likely.

**How does the average posting list size scale?**
The results also show that the average number of postings per key is a steady value for a multi term index, while it continues to increase linearly for a single term index. For large numbers of peers the posting lists for the single term index become extremely large in comparison to the result set of a multi term query.

**How does the average number of postings per peer scale?**
For large numbers of peers the posting lists for the single term index become extremely large in comparison to the result set of a multi term query. The maximum size of the multi term indexes created here is about 8.5 times the size of the single term index. I can also conclude that the growth of a multi term index will slow down eventually, while the single term index will continue to grow linearly. Overall the scalability of a multi term index still seems to be reasonable for use in a distributed setting.

**What is the retrieval quality of the system compared to a centralized system, when using top-k retrieval as a measurement?**
After the scalability analysis we performed several experiments to measure the retrieval performance of the multi term index compared to a single term index. The results from the ALVIS project promises excellent retrieval performance but that has not been true in this case. The top-10 overlap ratio between results from the single term index and a multi term index was found to be a meager 7.5% in the best case. The huge difference in retrieval performance between the two projects has two major causes. The first is the use of highly discriminative queries by the ALVIS project. This causes the result sets to be quite small, while each result set needs to contain at least twenty results. So the result sets for the single term index and the multi term index will be of a similar size since they are bounded. Therefore the top-20 overlap ratio is quite high.

The second cause of the difference in retrieval performance is the test collection and its properties. These statistics are summarized in Table 5.3. In the ALVIS project six peers are used

that each contain 5000 randomly selected Reuters news articles of on average 170 words. In this thesis a subset of the WT10g test collection was used. We tested the queries using 50 peers and each peer contains on average 200 web pages of on average 500 words each that together form a website. The use of a relatively large number of peers with each a relatively low number of related web pages seems to be a difficult combination for the rare key indexing method. The reasoning behind this is discussed in depth in section 5.4.

**Final remarks**

Based on the outcome of the experiments I have to conclude that the rare key indexing method as first introduced by the ALVIS project is unsuitable to index and search real websites. The concept of beforehand calculating term sets to limit the size of posting lists however remains an interesting one. Distributional semantics [46] [47] may be a good way to improve the query expander so it includes the best keys that do not fully match (subsets of) the query. Furthermore the use of only local knowledge to determine if a key is useful may result in storing a lot of keys that are not so useful at all. An approach using query adaptive indexing [49] [50] could look at previous queries from users to make a more informed choice about the value of a key.

# Bibliography


[1] Wikipedia, eDonkey network, http://en.wikipedia.org/wiki/EDonkey_network, 2007

[2] Bittorrent Inc., Bittorrent website, http://www.bittorrent.com, 2007

[3] Gnutella, Gnutella2 website, http://www.gnutella2.com, 2007

[4] ALVIS Consortium, ALVIS - Superpeer semantic search engines, http://www.alvis.info/alvis/, 2007

[5] Max Planck Institute, Minerva website, http://www.mpi-inf.mpg.de/departments/d5/software/minerva/, 2007

[6] Tang, C.; Xu, Z.; Mahalingham, M., pSearch: Information retrieval in Structured Overlays, *ACM HotNets-I*, 2002

[7] Wikipedia, Distributed Hash Tables (DHT), http://en.wikipedia.org/wiki/Distributed_hash_table, 2007

[8] Information Retrieval department (Glasgow University, United Kingdom), TREC Web Corpus : WT10g, http://ir.dcs.gla.ac.uk/test_collections/wt10g.html, May 2003

[9] Podnar, Ivana; Rajman, Martin; Luu, Toan; Klemm, Fabius; Aberer, Karl, Deliverable D4.1 Report on abstract model and P2P protocols, *ALVIS - Superpeer Semantic Search Engine*, 2006

[10]Markoff, John; Hansell, Saul, Hiding in Plain Sight, Google Seeks More Power, *The New York Times*, 14th of June 2006

[11]Google, Webmaster Help Center - What is a Sitemap file and why should I have one?, http://www.google.com/support/webmasters/bin/answe, 2007

[12]Wikipedia, RSS, http://en.wikipedia.org/wiki/RSS_(file_format), 2007

[13]Li, Jinyang; Loo, Boon Thau; Hellerstein, Joseph M.; Kaashoek, M. Frans; Karger, David R.; Morris, Robbert, On the Feasibility of Peer-to-Peer Web Indexing and Search, *International Workshop on Peer-to-Peer Systems (IPTPS) 2003*, 2003

[14]Lu, J.;Callan, J., Content-based retrieval in hybrid peer-to-peer networks, *Proceedings of the 12th international conference on Information and knowledge management*, 2003

[15]Pourebrahimi, B.;Bertels, K.;Vassiliadis, S., A Survey of Peer-to-Peer Networks, *ProRISC (Program for Research on Integrated Systems and Circuits) 2005 Proceedings - NWO-STW*, 2005

[16]Wikipedia, Hash functions, http://en.wikipedia.org/wiki/Hash_function, 2007

[17]The Internet Engineering Task Force (IETF), US Secure Hash Algorithm 1 (SHA-1) - RFC 3174, http://tools.ietf.org/html/rfc3174, 2001

[18]The Internet Engineering Task Force (IETF), The MD-5 Message Digest Algorithm - RFC 1321, http://tools.ietf.org/html/rfc1321, 1992

[19]Wikipedia, Napster, http://en.wikipedia.org/wiki/Napster, 2007

[20]Gnutella, Gnutella2 website, http://www.gnutella2.com, 2007

[21]The Free Network project, Freenet project, http://freenetproject.org/, 2007

[22]Bloom, Burton H., Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM*, July 1970

[23]Google Inc., The Google search engine, http://www.google.com, 2007

[24]Yahoo Inc., The Yahoo search engine, http://www.yahoo.com, 2007

[25]RFC-Gnutella website, Gnutella v0.4 protocol specification, http://rfc-gnutella.sourceforge.net/developer/stable/index.html, 2007

[26]Heaps, H.S., *Information Retrieval - Computational and Theoretical Aspects*, Academic Press, 1978

[27]Wikipedia, File Sharing, http://en.wikipedia.org/wiki/File_sharing, 2007

[28]Sharman Networks, Kazaa website, http://www.kazaa.com, 2007

[29]ANts P2P Project, ANts P2P Website on Sourceforge.net, http://antsp2p.sourceforge.net/, 2007

[30]Baaima NV, Joost website, http://www.joost.com/, 2007

[31]Babel Networks Ltd., Babelgum website, http://www.babelgum.com/, 2007

[32]Peercast, Peercast website, http://www.peercast.org/, 2007

[33]Ratnasamy, S.; Francis, P.; Handley M.; Karp, R., Shenker, S., A Scalable Content-Addressable Network , *In proceedings of the Special Interest Group on Data Communication (SIGCOMM) '01*, 2001

[34]Stoica, I.;Morris, R.;Karger, D.;Kaashoek, M.F.;Blakrishnan, H., Chord: A scalable peer-to-peer lookup service for internet applications, *In proceedings of the Special Interest Group on Data Communication (SIGCOMM) '01*, 2001

[35]Rowstron, A.; Druschel, P., Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, *In proceedings of Middleware'01*, 2001

[36]Zhao, B.;Duan, Y.;Huang, L.;Joseph, A., Tapestry: An infrastructure for fault-resilient wide-area location and routing, *Techical Report UCB//CDS-01-1141 U.C.Berkeley*,

[37]Castro, M. ;Druschel, P.; Kermarrec A.M.; Nandi, A.; Rowstron A.; Singh A., Splitstream: high-bandwidth multicast in cooperative environments, *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003

[38]Iyer, S.; Rowstron, A.; Druschel, P., Squirrel: A decentralized peer-to-peer web cache, *Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC), Monterey, California,USA*, 2002

[39]Rowstron, A.; Druschel, P., PAST: A large-scale, persistent peer-to-peer storage utility, *Proceedings of the 8th Workshop on HotTopics in Operating Systems (HotOS-VIII). Schloss Elmau, Germany:IEEECompSoc*, 2001

[40]Cox, L. P.; Murray, C. D.; Noble, B. D., Pastiche: making backup cheap and easy, *Special Interest Group on Operatings Systems - Operating Systems Review*, 2002

[41]Plaxton, C.G.; Rajaraman, R.; Richa, A.W., Accessing nearby copies of replicated objects in a distributed environment, *ACM Symposium on Parallel Algorithms and Architectures*, 1997

[42]Rutgers University (New Jersey, USA), PlanetP website, http://www.panic-lab.rutgers.edu/Research/planetp/, 2007

[43]Xu, Z; Zhang, Z, Technical Report HPL-2002-41, *HP Laboratories Palo Alto*, 2002

[44]van Rijsbergen, C.J.; Robertson, S.E.; Porter, M.F., New models in probabilistic information retrieval, *London: British Library. (British Library Research and Development Report, no. 5587)*, 1980

[45]Martin Porter, The 'official' implementations of the Porter Stemming algorithms in a number of programming languages, http://tartarus.org/~martin/PorterStemmer/, Jan 2006

[46]Kolb, P., Distributionelle Semantik - Automatisch Lesartengenerierung durch Erkennen unterschiedlicher Gebrauchskontexte, *Master Thesis, University of Potsdam (Germany)*, 2003

[47]Rajman, M.; Bonnet, A., Corpora-Base linguistics: New tools for Natural Language Processing, *1st Annual Conference of Association for Global Strategic Information*, 1992

[48]Wikipedia, Okapi BM25, http://en.wikipedia.org/wiki/Okapi_BM25, 2007

[49]Klemm, F.; Datta, A.; Aberer, K., A Query-Adaptive Partial Distributed Hash Table for Peer-to-Peer Systems, *Lecture notes in computer science*, 2004

[50]Balke, W.; Nejdl, W.; Siberski, W.; Thaden, U., Progressive distributed top-k retrieval in peer-to-peer networks, *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, 2005

# Appendix A – List of peers used in experiments

The following list contains the IP addresses of all the peers used in the experiments. For the retrieval performance experiments the first 50 peers (P1...P50) were used. And for the scalability analysis all 100 peers (P1...P100) were used. A peer contains on average 200 documents. For each peer its number is listed and its IP address.

| | | | |
|---|---|---|---|
| P1: 132.198.2.99 | P26: 152.2.44.1 | P51: 206.161.79.25 | P76: 131.95.98.200 |
| P2: 198.95.204.3 | P27: 205.182.53.51 | P52: 204.156.149.58 | P77: 198.107.235.4 |
| P3: 137.229.33.63 | P28: 192.216.245.8 | P53: 152.138.5.3 | P78: 134.205.165.120 |
| P4: 199.1.61.158 | P29: 192.87.7.4 | P54: 207.158.226.136 | P79: 205.163.84.69 |
| P5: 207.33.42.248 | P30: 140.190.65.12 | P55: 207.70.107.20 | P80: 207.60.86.242 |
| P6: 203.21.84.108 | P31: 204.162.147.197 | P56: 206.161.8.114 | P81: 204.140.220.228 |
| P7: 206.158.146.51 | P32: 206.54.38.105 | P57: 205.162.38.113 | P82: 204.178.72.78 |
| P8: 199.45.246.34 | P33: 207.126.101.90 | P58: 206.127.196.124 | P83: 208.194.65.10 |
| P9: 204.164.76.131 | P34: 206.86.52.4 | P59: 199.211.123.12 | P84: 203.111.77.64 |
| P10: 198.115.182.11 | P35: 206.65.84.166 | P60: 155.198.125.46 | P85: 198.62.160.12 |
| P11: 206.31.73.128 | P36: 130.70.46.129 | P61: 194.73.169.207 | P86: 203.15.58.8 |
| P12: 207.158.201.127 | P37: 136.159.130.50 | P62: 198.115.182.8 | P87: 130.226.166.167 |
| P13: 194.217.105.1 | P38: 129.12.200.19 | P63: 205.229.48.168 | P88: 206.30.242.28 |
| P14: 204.107.211.167 | P39: 192.217.82.137 | P64: 207.112.0.10 | P89: 204.62.160.251 |
| P15: 206.169.12.100 | P40: 24.129.0.69 | P65: 155.88.25.10 | P90: 206.86.48.91 |
| P16: 205.212.126.12 | P41: 136.210.100.51 | P66: 132.161.33.70 | P91: 199.182.71.101 |
| P17: 207.25.209.36 | P42: 199.18.207.26 | P67: 206.124.192.202 | P92: 208.131.64.137 |
| P18: 194.219.32.70 | P43: 207.60.134.110 | P68: 192.195.26.12 | P93: 207.31.82.101 |
| P19: 128.138.165.99 | P44: 204.233.138.5 | P69: 193.123.133.18 | P94: 137.82.170.200 |
| P20: 192.234.213.1 | P45: 204.141.224.193 | P70: 137.132.19.215 | P95: 194.88.132.152 |
| P21: 195.40.65.89 | P46: 134.84.174.20 | P71: 38.247.71.7 | P96: 143.216.21.6 |
| P22: 205.199.139.3 | P47: 198.246.244.55 | P72: 128.192.22.84 | P97: 206.171.10.14 |
| P23: 207.86.226.162 | P48: 206.96.72.123 | P73: 206.161.77.75 | P98: 137.82.194.23 |
| P24: 205.177.145.61 | P49: 207.60.110.71 | P74: 128.138.108.74 | P99: 192.132.206.7 |
| P25: 130.160.88.109 | P50: 206.98.169.214 | P75: 204.180.227.49 | P100: 192.107.39.3 |

# Appendix B – List of queries used in experiments

The following list consists of 632 queries. These queries were constructed by using the titles of the web pages. For the exact method please see section 5.3.2. The table has the following format:

| 1.Nr | 2. Stemmed Query | 3. STI | 4. MTI-075 | 5. MTI-010 | 6. MTI-015 | 7. MTI-020 |
|---|---|---|---|---|---|---|

The first column contains the number of the query. The list is sorted alphabetically on the second column, the query itself. The query is already stemmed using the Porter stemmer and it consists of either two or three words. The third column indicates if the query produced more then ten results when executed on the Single Term Index (STI). Columns four, five, six and seven list if any of the Multi Term Indexes (MTIs) returned more then ten results. Since the results from a MTI will be a subset of those of the STI, an STI entry will always be present when an MTI entry is. The *DFmax* percentage for each MTI is noted, for example MTI-075 stands for the Multi Term Index with *DFmax* set to 7,5% of the local number of documents.

All of the queries were executed on an index consisting of 50 peers. Of the 632 queries 335 returned ten or more results on the STI. The MTI returned the following number of queries with ten or more results for different settings of *DFmax*:

- 7,5% - 65 queries
- 10% - 59 queries
- 15% - 44 queries
- 20% - 45 queries

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | abstract syntax | STI | | | | |
| 1 | academ calendar | STI | | | | |
| 2 | academ visitor signal | | | | | |
| 3 | acadian louisiana lesson | STI | | | | |
| 4 | adapt signal process | STI | | | | |
| 5 | administr train cours | STI | | | | |
| 6 | advanc elect | STI | | | | |
| 7 | advantag melbourn assist | STI | | | | |
| 8 | advantag melbourn centr | STI | | | | |
| 9 | advantag melbourn introduct | | | | | |
| 10 | aerob studio | | | | | |
| 11 | alabama graduat school | STI | | | | |
| 12 | albania observ liber | | | | | |
| 13 | alleg victim iranian | | | | | |
| 14 | american colleg | STI | MTI-075 | | | |
| 15 | american studi | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 16 | anaskophsh kyproi | | | | | |
| 17 | anglo turkish associ | | | | | |
| 18 | anion polymer initi | STI | | | | |
| 19 | anion polymer propagat | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 20 | anion vinyl polymer | STI | | | | |
| 21 | annot visit | STI | | | | |
| 22 | annual review | STI | MTI-075 | MTI-10 | MTI-15 | |
| 23 | annual survei | STI | | | | |
| 24 | asoci access | | | | | |
| 25 | assist mission | STI | | | | |
| 26 | attack breast endeavor | | | | | |
| 27 | attack writer suggest | | | | | |
| 28 | attribut grammar | STI | | | | |
| 29 | award innov | STI | | | | |
| 30 | barbara masser | | | | | |
| 31 | basic econom model | STI | | | | |
| 32 | belgian refuge threaten | | | | | |
| 33 | biograph inform bruce | | | | | |
| 34 | biologi deptart facil | | | | | |
| 35 | borrow natur | STI | | | | |
| 36 | breath easier pulmozym | | | | | |
| 37 | bring histori endeavor | STI | | | | |
| 38 | broadcast chipset slash | | | | | |
| 39 | bruce biographi | | | | | |
| 40 | california assembl public | STI | | | | |
| 41 | camera store | STI | | | | |
| 42 | cameroon urban agricultur | | | | | |
| 43 | canada graduat school | STI | | | | |
| 44 | carbon fiber | | | | | |
| 45 | career singapor | | | | | |
| 46 | carolina endeavor | STI | | | | |
| 47 | casio contest winner | | | | | |
| 48 | cation polymer chain | STI | | | | |
| 49 | cation polymer initi | STI | | | | |
| 50 | cation polymer propag | STI | | | | |
| 51 | cation polymer termin | STI | | | | |
| 52 | cation vinyl polymer | STI | | | | |
| 53 | centr alloi solidif | | | | | |
| 54 | challeng project descriptor | | | | | |
| 55 | check survei thank | STI | | | | |
| 56 | chesapeak incid | | | | | |
| 57 | chief report | STI | | | | |
| 58 | choos adventur | STI | | | | |
| 59 | christma humor | | | | | |
| 60 | citcom servic | STI | | | | |
| 61 | citcom servic acknowledg | | | | | |
| 62 | citcom servic overview | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 63 | citnet acknowledg | | | | | |
| 64 | class descript | STI | MTI-075 | MTI-10 | | |
| 65 | class inform announc | STI | | | | |
| 66 | class requir | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 67 | class schedul | STI | MTI-075 | MTI-10 | | |
| 68 | comic newslett | STI | | | | |
| 69 | commun garden vancouvc | | | | | |
| 70 | commun internet | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 71 | commun multimedia market | STI | | | | |
| 72 | compani profil | STI | | | | |
| 73 | compil supercombin | | | | | |
| 74 | configur freeppp | | | | | |
| 75 | connecticut graduat school | | | | | |
| 76 | consolid balanc sheet | STI | | | | |
| 77 | consolid statement equiti | STI | | | | |
| 78 | consolid statement incom | STI | | | | |
| 79 | consortium project | STI | | | | |
| 80 | consum product | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 81 | contact assist | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 82 | contact postcard promot | STI | | | | |
| 83 | continent internet captain | STI | | | | |
| 84 | continent internet custom | STI | MTI-15 | | | |
| 85 | convent techniqu | STI | | | | |
| 86 | convert mobil telephoni | | | | | |
| 87 | corpfinet career center | | | | | |
| 88 | corpfinet interview | | | | | |
| 89 | corpor financ updat | STI | | | | |
| 90 | cours signal process | STI | | | | |
| 91 | cover endeavor | STI | | | | |
| 92 | cover sheet | STI | | | | |
| 93 | cowboi biographi | | | | | |
| 94 | cowboi junki | | | | | |
| 95 | crawler search | | | | | |
| 96 | creat applic proxi | STI | | | | |
| 97 | crimin justic | STI | | | | |
| 98 | crystallin polym | | | | | |
| 99 | cwuaa championship result | | | | | |
| 100 | cyber adventur | STI | | | | |
| 101 | cypriot cultur | | | | | |
| 102 | cyprriot costum | STI | | | | |
| 103 | debug fault simul | | | | | |
| 104 | decemb profil | STI | | | | |
| 105 | delawar graduat school | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 106 | delphi nortech softwar | STI | | | | |
| 107 | depart contact | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 108 | depart directori | STI | | | | |
| 109 | depart directori vwxyz | | | | | |
| 110 | descript program languag | STI | | | | |
| 111 | design dimens perform | STI | | | | |
| 112 | desktop publish | STI | | | | |
| 113 | develop resourc | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 114 | diana reichardt | | | | | |
| 115 | differ instanti | | | | | |
| 116 | dilut solut viscometri | | | | | |
| 117 | dimitar homepga | | | | | |
| 118 | disappear javad rouhani | STI | | | | |
| 119 | discount comic check | STI | | | | |
| 120 | discuss analysi | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 121 | disson record | | | | | |
| 122 | district columbia graduat | | | | | |
| 123 | dollar dungeon | | | | | |
| 124 | domin abram | STI | MTI-10 | | | |
| 125 | donna jessop | | | | | |
| 126 | download logotron softwar | | | | | |
| 127 | drive endeavor | STI | | | | |
| 128 | dynam transform | STI | | | | |
| 129 | earli synthet polym | | | | | |
| 130 | electr extend speech | | | | | |
| 131 | elfman interview | | | | | |
| 132 | empir success object | STI | | | | |
| 133 | employ opportun check | STI | | | | |
| 134 | endeavor content | STI | MTI-20 | | | |
| 135 | endeavor magazin april | STI | | | | |
| 136 | energi servic | STI | MTI-075 | MTI-10 | | |
| 137 | enhanc elmhurst homepag | | | | | |
| 138 | environment scienc | STI | MTI-10 | | | |
| 139 | epoxi resin | | | | | |
| 140 | erowid cannabi experi | | | | | |
| 141 | erowid dream | STI | | | | |
| 142 | erowid entheogen disclaim | STI | | | | |
| 143 | erowid guestbook | STI | | | | |
| 144 | erowid guestbook addit | | | | | |
| 145 | erowid hippi | | | | | |
| 146 | erowid ketamin articl | STI | | | | |
| 147 | erowid magic mushroom | | | | | |
| 148 | erowid mushroom cultiv | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 149 | erowid mushroom scienc | | | | | |
| 150 | erowid salvia divinorum | | | | | |
| 151 | erowid tobacco nicotin | | | | | |
| 152 | event calendar | STI | | | | |
| 153 | explor vastli prefer | | | | | |
| 154 | factori construct | STI | | | | |
| 155 | famou student athlet | | | | | |
| 156 | financi highlight | STI | | | | |
| 157 | financi inform | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 158 | financi overview | STI | | | | |
| 159 | florida graduat school | STI | | | | |
| 160 | foreign graduat school | STI | | | | |
| 161 | franki borison | STI | | | | |
| 162 | frequenc domain kalman | | | | | |
| 163 | futur scienc | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 164 | garbag collect method | STI | | | | |
| 165 | garbag sound video | | | | | |
| 166 | gedistribueerd systemen | | | | | |
| 167 | geffen record | STI | | | | |
| 168 | geffen vintag | STI | | | | |
| 169 | geffen vintag audio | | | | | |
| 170 | geffen vintag video | | | | | |
| 171 | genentech annual report | | | | | |
| 172 | genentech leadership | STI | MTI-075 | | | |
| 173 | genentech market todai | STI | | | | |
| 174 | gener comment | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 175 | gener descript | STI | MTI-075 | MTI-10 | MTI-15 | |
| 176 | gener macintosh inform | STI | | | | |
| 177 | gener tourist introduct | STI | | | | |
| 178 | georgia graduat school | STI | | | | |
| 179 | german commun garden | STI | | | | |
| 180 | gertrud endeavor | | | | | |
| 181 | gettysburg address format | | | | | |
| 182 | gettysburg address unformat | | | | | |
| 183 | gillian biographi | | | | | |
| 184 | gillian photograph | | | | | |
| 185 | gillian reviv | | | | | |
| 186 | gillian welch | | | | | |
| 187 | glass transit | STI | | | | |
| 188 | global facil urban | | | | | |
| 189 | govern polit | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 190 | grabber shell script | | | | | |
| 191 | graduat field studi | STI | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 192 | graduat school | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 193 | graham chapman biographi | | | | | |
| 194 | grant contract applic | STI | | | | |
| 195 | granular visit function | | | | | |
| 196 | guitar record | STI | | | | |
| 197 | guitar record classifi | | | | | |
| 198 | guitar record column | STI | MTI-20 | | | |
| 199 | guitar record contact | STI | | | | |
| 200 | guitar record essenti | STI | | | | |
| 201 | guitar record interview | STI | | | | |
| 202 | guitar record label | STI | | | | |
| 203 | guitar record power | STI | | | | |
| 204 | guitar record search | STI | | | | |
| 205 | guitar record tabplu | | | | | |
| 206 | guitar record undiscov | STI | | | | |
| 207 | hacker challeng break | | | | | |
| 208 | handl stress | STI | | | | |
| 209 | hawaii graduat school | | | | | |
| 210 | hayden displai | | | | | |
| 211 | health medicin | STI | MTI-075 | MTI-10 | | |
| 212 | health scienc | STI | MTI-075 | MTI-20 | | |
| 213 | higher order attribut | STI | | | | |
| 214 | highlight interest project | STI | | | | |
| 215 | histor conserv studi | STI | | | | |
| 216 | human interact | STI | MTI-075 | MTI-10 | | |
| 217 | human resourc philosophi | STI | | | | |
| 218 | illinoi graduat school | STI | | | | |
| 219 | implement method | STI | MTI-075 | MTI-10 | | |
| 220 | increment evalu perform | STI | | | | |
| 221 | index multimedia inform | STI | | | | |
| 222 | indiana graduat school | | | | | |
| 223 | industri capabl develop | STI | | | | |
| 224 | industri labor relat | STI | | | | |
| 225 | inform ethic | STI | | | | |
| 226 | inform resourc | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 227 | innov support assist | STI | | | | |
| 228 | innov support framework | STI | | | | |
| 229 | innov support inform | STI | | | | |
| 230 | innov support network | STI | | | | |
| 231 | innov support technic | STI | | | | |
| 232 | instant collect section | STI | | | | |
| 233 | institut review board | STI | | | | |
| 234 | institut servic | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 235 | integr support | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 236 | interest research develop | STI | | | | |
| 237 | intern advisori panel | | | | | |
| 238 | intern affair | STI | | | | |
| 239 | internet relat | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 240 | internet total account | STI | | | | |
| 241 | internet winner loser | | | | | |
| 242 | intern linkag | STI | | | | |
| 243 | intern linkag bilater | | | | | |
| 244 | intern linkag multilater | | | | | |
| 245 | introduc cypru | | | | | |
| 246 | introduct minist | | | | | |
| 247 | introduct overview | STI | MTI-075 | | | |
| 248 | iranian writer kidnap | | | | | |
| 249 | israel egypt | STI | | | | |
| 250 | janic biographi | | | | | |
| 251 | jennif bookmark | | | | | |
| 252 | jewel sandov | | | | | |
| 253 | joshua muravchik | | | | | |
| 254 | kasten algorithm | | | | | |
| 255 | kathi keenan | STI | MTI-10 | MTI-15 | | |
| 256 | kathi keller | STI | MTI-075 | | | |
| 257 | kathi koerper | | | | | |
| 258 | kentucki graduat school | | | | | |
| 259 | kevin homepag | | | | | |
| 260 | kevin resum | STI | | | | |
| 261 | klima feedback | | | | | |
| 262 | lambda express | | | | | |
| 263 | laura wigod | | | | | |
| 264 | letter stockhold | | | | | |
| 265 | letter support faraj | | | | | |
| 266 | librari environ | STI | MTI-075 | MTI-10 | | |
| 267 | light cyberspac | STI | | | | |
| 268 | listen email phone | | | | | |
| 269 | lobbi guidelin employe | | | | | |
| 270 | logic famili | STI | | | | |
| 271 | logotron catalogu | STI | | | | |
| 272 | logotron press releas | | | | | |
| 273 | lyric garbag | | | | | |
| 274 | macintosh instal | STI | MTI-075 | MTI-10 | MTI-15 | |
| 275 | macintosh modem initi | | | | | |
| 276 | macintosh onlin public | STI | | | | |
| 277 | macintosh resourc internet | STI | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 278 | manag success product | STI | | | | |
| 279 | manpow develop assist | | | | | |
| 280 | manufactur macintosh hardwar | STI | | | | |
| 281 | maria biographi | | | | | |
| 282 | maria carri | STI | | | | |
| 283 | maria everybodi | | | | | |
| 284 | maria human | STI | | | | |
| 285 | maria listen | | | | | |
| 286 | maria mckee | | | | | |
| 287 | maria perfect dress | | | | | |
| 288 | maria scarlov | STI | | | | |
| 289 | maria smarter | | | | | |
| 290 | mariu usher | STI | | | | |
| 291 | market chang environ | STI | | | | |
| 292 | maryland graduat school | | | | | |
| 293 | massachusett graduat school | STI | | | | |
| 294 | mehdi rouhani | | | | | |
| 295 | membership applic | STI | MTI-20 | | | |
| 296 | membership inform | STI | MTI-075 | MTI-20 | | |
| 297 | memori model engin | STI | | | | |
| 298 | metallocen catalysi | | | | | |
| 299 | metropoli worknet | | | | | |
| 300 | michael palin biographi | | | | | |
| 301 | michigan graduat school | STI | | | | |
| 302 | mississippi graduat school | STI | | | | |
| 303 | modem semiconductor revenu | | | | | |
| 304 | modular build white | | | | | |
| 305 | molecular weight | | | | | |
| 306 | murrai biographi | | | | | |
| 307 | nader afshar | | | | | |
| 308 | nairobi urban agricultur | | | | | |
| 309 | nation patent inform | STI | | | | |
| 310 | nation scienc award | STI | | | | |
| 311 | nation scienc technolog | STI | | | | |
| 312 | nation sport center | STI | | | | |
| 313 | nation technolog award | STI | | | | |
| 314 | nation undergradu research | STI | | | | |
| 315 | natur polym | | | | | |
| 316 | network structur | STI | | | | |
| 317 | newsmak april | | | | | |
| 318 | newswir august | | | | | |
| 319 | newswir decemb | | | | | |
| 320 | newswir novemb | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 321 | newswir octob | | | | | |
| 322 | newswir septemb | | | | | |
| 323 | nonlinear polym | | | | | |
| 324 | north carolina graduat | STI | | | | |
| 325 | north eastern graduat | | | | | |
| 326 | nuclear magnet reson | | | | | |
| 327 | nylon synthesi | | | | | |
| 328 | object testabl member | | | | | |
| 329 | object veloc | | | | | |
| 330 | offic board director | STI | | | | |
| 331 | offici california legisl | | | | | |
| 332 | offic inform commun | STI | MTI-20 | | | |
| 333 | offic research servic | STI | | | | |
| 334 | offic technolog develop | STI | MTI-075 | | | |
| 335 | olefin metathesi polymer | | | | | |
| 336 | olymp nation sport | STI | | | | |
| 337 | opportun person product | STI | | | | |
| 338 | opposit activ | STI | | | | |
| 339 | optic group homepag | | | | | |
| 340 | oragan school check | | | | | |
| 341 | organiz chart research | | | | | |
| 342 | osteopath medicin | STI | | | | |
| 343 | overview citnet | | | | | |
| 344 | packet filter applic | | | | | |
| 345 | paint decor | STI | MTI-15 | MTI-20 | | |
| 346 | parent consent letter | | | | | |
| 347 | partner index | STI | | | | |
| 348 | pennsylvania graduat school | STI | | | | |
| 349 | pertin figur | | | | | |
| 350 | peter gabriel biographi | | | | | |
| 351 | peter mayer | | | | | |
| 352 | peter stoyanov presid | | | | | |
| 353 | philip chipset | STI | | | | |
| 354 | photo album | STI | | | | |
| 355 | photo galleri | STI | | | | |
| 356 | physic infrastructur | STI | | | | |
| 357 | physic scienc | STI | MTI-075 | MTI-10 | MTI-20 | |
| 358 | pinpoint servic | | | | | |
| 359 | pinpoint survei advic | | | | | |
| 360 | pinpoint train | | | | | |
| 361 | poland feedback | | | | | |
| 362 | poland journei | | | | | |
| 363 | polym composit | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 364 | positiv reductiv | | | | | |
| 365 | possibl simul | STI | MTI-075 | MTI-10 | MTI-15 | |
| 366 | postcard promot | STI | | | | |
| 367 | postgradu train initi | STI | | | | |
| 368 | power endeavor | STI | | | | |
| 369 | pragati grover | | | | | |
| 370 | presidenti elect bulgaria | | | | | |
| 371 | press crucibl | | | | | |
| 372 | press insid knowledg | STI | | | | |
| 373 | press peter rabbit | | | | | |
| 374 | press releas | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 375 | press releas octob | STI | | | | |
| 376 | press releas septemb | STI | | | | |
| 377 | press victorian crime | | | | | |
| 378 | produc hospic | | | | | |
| 379 | product extra | STI | | | | |
| 380 | professor derek rutter | | | | | |
| 381 | program applic | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 382 | proton adipoyl chlorid | | | | | |
| 383 | prototyp gofer | STI | | | | |
| 384 | proxim market | STI | | | | |
| 385 | prune optim | | | | | |
| 386 | psycholog centr research | STI | | | | |
| 387 | psycholog centr studi | STI | | | | |
| 388 | psycholog development psycholog | STI | | | | |
| 389 | psycholog handbook | STI | | | | |
| 390 | psycholog neuropsycholog cognit | | | | | |
| 391 | psycholog public | STI | MTI-10 | MTI-15 | MTI-20 | |
| 392 | public relat | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 393 | public research institut | STI | | | | |
| 394 | public skate schedul | | | | | |
| 395 | quarterli report | STI | | | | |
| 396 | questionnair cover letter | | | | | |
| 397 | rachel modena barasch | | | | | |
| 398 | raleigh freeman | | | | | |
| 399 | recent activ | STI | MTI-075 | MTI-10 | MTI-15 | |
| 400 | recreat leisur | STI | | | | |
| 401 | reduc primit count | | | | | |
| 402 | regist product | STI | MTI-075 | | | |
| 403 | relat formal | STI | MTI-075 | MTI-10 | | |
| 404 | religi arrest continu | | | | | |
| 405 | remov inherit attribut | | | | | |
| 406 | report instruct | STI | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 407 | report manag | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 408 | repositori white paper | STI | | | | |
| 409 | request graduat admiss | | | | | |
| 410 | research confer center | STI | | | | |
| 411 | research graduat studi | STI | MTI-10 | | | |
| 412 | research innov | STI | MTI-075 | MTI-10 | MTI-15 | |
| 413 | research interest | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 414 | research lighter | | | | | |
| 415 | research seminar programm | STI | | | | |
| 416 | research servic | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 417 | research staff section | STI | | | | |
| 418 | research subject | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 419 | research support april | STI | | | | |
| 420 | research support biolog | STI | | | | |
| 421 | research support march | STI | | | | |
| 422 | research support newslett | STI | | | | |
| 423 | research support physic | STI | | | | |
| 424 | research support social | STI | | | | |
| 425 | resourc research | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 426 | resultaten groupwar evaluatieproject | | | | | |
| 427 | review afronet | | | | | |
| 428 | review american heart | STI | | | | |
| 429 | review bookwir | | | | | |
| 430 | review brettnew | | | | | |
| 431 | review careerweb | | | | | |
| 432 | review cdnow | | | | | |
| 433 | review channel | STI | | | | |
| 434 | review charg | STI | | | | |
| 435 | review cnnfn | | | | | |
| 436 | review comedi central | | | | | |
| 437 | review consum world | STI | | | | |
| 438 | review crayon | | | | | |
| 439 | review cyberwalk | | | | | |
| 440 | review dejanew | | | | | |
| 441 | review directori servic | STI | | | | |
| 442 | review discoveri channel | STI | | | | |
| 443 | review disnei | | | | | |
| 444 | review electr postcard | | | | | |
| 445 | review entertain weekli | STI | | | | |
| 446 | review epicuri | | | | | |
| 447 | review espnet sportszon | | | | | |
| 448 | review exploranet | | | | | |
| 449 | review famili | STI | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 450 | review familiar quotat | | | | | |
| 451 | review fedex | | | | | |
| 452 | review fedworld | | | | | |
| 453 | review firefli | | | | | |
| 454 | review gigaplex | | | | | |
| 455 | review global network | STI | | | | |
| 456 | review hotwir | | | | | |
| 457 | review hypermod | | | | | |
| 458 | review industri | STI | MTI-075 | MTI-10 | MTI-15 | |
| 459 | review librari congress | STI | | | | |
| 460 | review mayaquest | | | | | |
| 461 | review mercuri center | | | | | |
| 462 | review metavers | | | | | |
| 463 | review mississippi review | STI | | | | |
| 464 | review nation institut | STI | | | | |
| 465 | review netscap commun | STI | | | | |
| 466 | review networth | | | | | |
| 467 | review njonlin weather | | | | | |
| 468 | review onlin | STI | MTI-075 | MTI-10 | MTI-20 | |
| 469 | review onlin health | STI | | | | |
| 470 | review place | STI | MTI-075 | MTI-10 | MTI-15 | |
| 471 | review project galileo | | | | | |
| 472 | review scholast central | | | | | |
| 473 | review seniorcom | | | | | |
| 474 | review sharewar | STI | | | | |
| 475 | review sionlin | | | | | |
| 476 | review sonicnet | | | | | |
| 477 | review space | STI | | | | |
| 478 | review terraquest | | | | | |
| 479 | review thoma | STI | | | | |
| 480 | review travel channel | STI | | | | |
| 481 | review tripod | | | | | |
| 482 | review uroulett | | | | | |
| 483 | review voyag | STI | | | | |
| 484 | rhode island graduat | | | | | |
| 485 | sabina aharpour | | | | | |
| 486 | sampl deriv | STI | | | | |
| 487 | scapp english | | | | | |
| 488 | scapp french | | | | | |
| 489 | scheme local postgradu | | | | | |
| 490 | scholarli endeavor | STI | | | | |
| 491 | school adopt scheme | | | | | |
| 492 | scienc technolog | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 493 | scienc technolog promot | STI | | | | |
| 494 | search power monei | STI | | | | |
| 495 | secular model | | | | | |
| 496 | septemb content | STI | MTI-20 | | | |
| 497 | servic check | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 498 | signal process digit | STI | | | | |
| 499 | simcopi shell script | | | | | |
| 500 | simul answer econom | STI | | | | |
| 501 | skill enhanc | STI | MTI-075 | | | |
| 502 | skill manpow | | | | | |
| 503 | sober endeavor | | | | | |
| 504 | social scienc | STI | MTI-075 | | | |
| 505 | solectron appoint david | | | | | |
| 506 | solectron complet elect | | | | | |
| 507 | solectron complet purchas | | | | | |
| 508 | solectron corpor | STI | | | | |
| 509 | solectron corpor quarter | | | | | |
| 510 | solectron corpor second | | | | | |
| 511 | solut centr | STI | | | | |
| 512 | solut smart product | STI | | | | |
| 513 | south carolina graduat | STI | | | | |
| 514 | southern cultur | STI | | | | |
| 515 | southern cultur nashvil | | | | | |
| 516 | southern cultur throw | | | | | |
| 517 | special clientel | | | | | |
| 518 | special featur | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 519 | special featur sqlwindow | | | | | |
| 520 | special featur visual | STI | | | | |
| 521 | special offer | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 522 | spectral estim signal | | | | | |
| 523 | speed fault simul | | | | | |
| 524 | sport leisur | STI | | | | |
| 525 | staff directori research | STI | | | | |
| 526 | staff signal process | STI | | | | |
| 527 | starch cellulos | | | | | |
| 528 | start point internet | STI | | | | |
| 529 | static detect | STI | | | | |
| 530 | static optim | STI | | | | |
| 531 | steve farnsworth | | | | | |
| 532 | steve hinkl | STI | MTI-075 | | | |
| 533 | stock stockhold inform | STI | | | | |
| 534 | strang travel stori | STI | | | | |
| 535 | streamwork player | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 536 | streamwork server price | | | | | |
| 537 | streamwork server public | | | | | |
| 538 | streamwork server specif | | | | | |
| 539 | streamwork server support | | | | | |
| 540 | streamwork transmitt | | | | | |
| 541 | streamwork transmitt enhanc | | | | | |
| 542 | streamwork transmitt price | | | | | |
| 543 | streamwork transmitt specif | | | | | |
| 544 | strong intern linkag | STI | | | | |
| 545 | structur thesi | STI | | | | |
| 546 | student athlet | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |
| 547 | student athlet award | | | | | |
| 548 | student evalu | STI | MTI-10 | MTI-15 | MTI-20 | |
| 549 | student signal process | STI | | | | |
| 550 | stuffit expand | STI | | | | |
| 551 | submiss resum | | | | | |
| 552 | subscrib servic | STI | MTI-075 | MTI-10 | MTI-20 | |
| 553 | success highlight | STI | | | | |
| 554 | sunnit cleric murder | | | | | |
| 555 | support terror | STI | | | | |
| 556 | suspend restart simul | | | | | |
| 557 | sustain develop | STI | MTI-075 | MTI-10 | | |
| 558 | switchmod power suppli | | | | | |
| 559 | syntax semant | STI | | | | |
| 560 | tabriz execut | STI | | | | |
| 561 | target achiev | STI | | | | |
| 562 | techmonth diari event | | | | | |
| 563 | techmonth techmonth award | STI | | | | |
| 564 | techmonth techmonth organis | STI | | | | |
| 565 | technic report | STI | MTI-075 | MTI-10 | | |
| 566 | technolog centr | STI | MTI-075 | MTI-20 | | |
| 567 | technolog knowledg infrastructur | STI | | | | |
| 568 | teenag fanclub | | | | | |
| 569 | teenag fanclub photo | | | | | |
| 570 | tennesse graduat school | STI | | | | |
| 571 | termin silli | | | | | |
| 572 | terraweb gener inform | | | | | |
| 573 | terraweb guestbook | | | | | |
| 574 | theolog school | | | | | |
| 575 | theolog school admiss | | | | | |
| 576 | thermoplast elastom | | | | | |
| 577 | track field record | STI | | | | |
| 578 | train cours | STI | MTI-075 | MTI-10 | MTI-15 | MTI-20 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 579 | train resourc materi | STI | | | | |
| 580 | trent affair | | | | | |
| 581 | troubleshoot freeppp | | | | | |
| 582 | univers calgari dinosaur | STI | | | | |
| 583 | univers calgari field | STI | | | | |
| 584 | univers calgari track | STI | | | | |
| 585 | univers calgari tumbl | | | | | |
| 586 | univers forest | STI | | | | |
| 587 | univers gazett octob | | | | | |
| 588 | univers research council | STI | | | | |
| 589 | upcom event | STI | MTI-075 | | | |
| 590 | vantag endeavor | | | | | |
| 591 | variou poetri | STI | | | | |
| 592 | vendor macintosh product | STI | | | | |
| 593 | vermont graduat school | STI | | | | |
| 594 | veterinari medicin | STI | | | | |
| 595 | vibrant industri | STI | | | | |
| 596 | victorian societi summer | STI | | | | |
| 597 | vinyl polym | | | | | |
| 598 | violent demonstr report | | | | | |
| 599 | virginia graduat school | STI | | | | |
| 600 | virtual endeavor | STI | | | | |
| 601 | visit function | STI | MTI-10 | | | |
| 602 | visit function optim | STI | | | | |
| 603 | visit subsequ | STI | | | | |
| 604 | walter herzog | | | | | |
| 605 | webpoint onlin | STI | | | | |
| 606 | weezer pinkerton | | | | | |
| 607 | weezer pinkerton album | | | | | |
| 608 | weezer pinkerton biographi | | | | | |
| 609 | weezer pinkerton sound | | | | | |
| 610 | welcom zycad | | | | | |
| 611 | wellcom multiplex | | | | | |
| 612 | whizz mexico | | | | | |
| 613 | wisconsin graduat school | STI | | | | |
| 614 | xingmpeg encod | STI | | | | |
| 615 | xingmpeg encod enhanc | | | | | |
| 616 | xingmpeg encod price | | | | | |
| 617 | xingmpeg encod specif | | | | | |
| 618 | xingmpeg player | STI | | | | |
| 619 | xingmpeg player enhanc | | | | | |
| 620 | xingmpeg player price | | | | | |
| 621 | xingmpeg player specif | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 622 | xingpartn benefit | STI | | | | |
| 623 | xingpartn program applic | | | | | |
| 624 | xingpartn requir | STI | | | | |
| 625 | zycad announc increas | | | | | |
| 626 | zycad compani newslett | | | | | |
| 627 | zycad custom support | | | | | |
| 628 | zycad document search | | | | | |
| 629 | zycad employ opportun | | | | | |
| 630 | zycad incid submiss | | | | | |
| 631 | zycad offic locat | | | | | |