A Distributed Directory Service

by

HONGBING LI
B. Sc. Shenyang Polytechnic University, China, 1984
M. Sc. Shenyang Polytechnic University, China, 1987

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

January, 1994

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date April 28, 1994

Abstract

Fast response time, large amounts of directory information, and friendly user interfaces are key criteria for a good distributed directory service. Although there are some Internet directory services developed, many of them have failed to achieve these goals. The work presented in this thesis is motivated by the importance of an Internet directory service, specifically to provide network users' email addresses, and the current development of such services, which is neither adequate nor effective.

We designed and implemented a light weight directory (lwd) service system, which provides an Internet white pages service, with the following features: simple architecture, easy incorporation of new sources of information, and quick response time to users' queries. The system avoids requiring global cooperation, as most of other directory service system do by storing all of the white-pages information locally. Thus response time to users' queries are much faster. The system provides a mechanism to collect and exchange information among different sites in the system. Two information exchange protocols have been implemented, one guaranteeing all information maintained on one site can be transferred to another, and the other more intuitive and easier to implement.

In comparison of the lwd system to several important Internet white pages services, we found that lwd has simple query structure, fast response time, and friendly user interface. In the lwd system, only a person's name is required to look up instead of hierarchical structure information or a set of keywords many other existing or proposed directory services demand. Also, the system provides error-tolerance capability to users' queries by conducting "approximate matches" to user submitted partial names.

Contents

List of Tables

List of Figures

Acknowledgement

I would like to thank Gerald Neufeld, my supervisor, for his guidance, encouragement and patience throughout my work on this thesis. I would also like to thank Norm Hutchinson for his reading through the draft of this thesis.

Many thanks to Zheng Zhu for discussing the draft with me, to Christopher Healey and Tim Spurrell for their proofreading, to Runping Qi for his suggestions and comments, and to Niels Maretti for his help in Latex and Gemacs. Thanks also to other staff and graduate students in the Department of Computer Science at UBC, Barry Brachman, John Demco, Peter Phillips, Michael Sanderson, Yinchun Xu, and Ying Zhang.

I owe thanks to my great husband, Gang Li, for his encouragement, and for taking more responsibility to care for our son, Yaoyao Li, during the final stage of my thesis. I am very grateful for Yaoyao's understanding when instead of staying home to tell him stories, I went to school to work in the evenings.

Last but not the least, I would like to express my gratefulness to my grandma, who has helped my parents to raise me and is always a source of care and encouragement to me.

Chapter 1

Introduction

Computer to computer communication enhances functionality and usability of computers by sharing data and resources. The research presented in this thesis discusses the design and implementation of a distributed directory service, which provides services to network users who want to find other network users' information, such as email addresses, mailing addresses, and phone numbers.

1.1 The Problem and Motivations

As the use of computer network and electronic mail has been increasingly integrated with every aspect of activities in the society, a frequently asked question is,

> "I know someone's name, and I think he must have an electronic mail address somewhere. How can I find it?"

A directory service provides an answer to this question. In this thesis, a *directory service* refers to an on-line computer system, centralized or distributed, which holds directory information and provides its users with services to access information. A distributed directory service consists of one or more cooperating subsystems. A subsystem contains some directory information and executes a set of defined directory protocols.

From a user's perspective, a directory service is an integrated one, capable of providing various services. These services allow users to lookup and update the directory information.

Directory services have become increasingly important due to the growth of computer networks and their penetration into the daily life of our society. There are millions of users in the world who are using computer networks that communicate with each other. What is needed is a directory service that identifies computer system users by name. The directory services are sometimes known as *white pages* services because their function is similar to the white pages of a telephone book. Given a person's name, computer white pages may supply information such as email address, telephone number, and mailing address.

The importance of such a service can be exemplified by imagining the chaotic situation of a large and modern city without a telephone directory. Unlike telephone directory services, which are provided by a centralized authority, computer directory services are complicated by thousands of autonomous organizations in a network such as Internet, a world-wide computer network connecting over one million computers with over 10 million users. Within the Internet there is no central authority that provides directory services.

Currently research has been done on some Internet directory services. However, all of them have the limitation of the directory information sources. That is, they only contain information about a small set of network users. In many cases it is still not possible to

locate the electronic mail address for a given Internet user, even though a significant effort has been made to achieve this goal. Moreover, accessing a collection of available directories is time-consuming and requires knowledge of each directory's user interface. This severely damages the usability of such services.

The work reported in this thesis is motivated by the importance of an Internet directory service, especially providing network users' email addresses, and the current development of such services, which is neither adequate nor effective.

1.2 Requirements

To provide a fast directory service system, the following requirements should be accomplished:

- Local/fast lookup. Although quick response time for a look up is a key measurement of the effectiveness of every directory service system, many systems have failed to achieve this goal. Currently, a good directory service system can respond a user's query in a few seconds. On the other hand, some systems may take up to 200 seconds to answer a query. Delayed responses may be caused by a lack of cooperation between local and remote servers. To solve this problem, a query which is fulfilled locally can reduce lookup time significantly. This requires that directory information in a system be kept locally in each cooperating subsystem.

- Containing large amounts of directory information from a variety of sources. Directory information usually comes from two types of sources: public and local. Public sources include the information which people can easily obtain from the Internet such as USENET and other network directory services, such as WHOIS

[Harrenstien85], Netfind [Schwartz89]. There is a huge amount of information located at various local sources. The local information sources, such as the *userinfo* database created and maintained at the Computer Science Department of the University of British Columbia, contain locally complete and up-to-date information. To obtain as much, and as recent user information as possible, a directory service system must be able to integrate local sources and public sources into a directory information collection mechanism.

- Providing the service at a reasonable cost. This cost includes consumption of network bandwidth, data storage cost, computational expense for processing user queries, and the cost of operating such a service. Currently, most distributed directory service systems require network-wide cooperation, such as dedicated directory service servers throughout the network, in order to provide services. Therefore, in addition to minimizing other costs such as data storage, network bandwidth, it is important to consider the cost effect in operating a distributed directory service system.

- Providing a friendly user interface. A friendly user interface usually possesses two major capabilities: easy access to directory information, and error-tolerance regarding user's queries. Providing an error-tolerant query processing mechanism is important because users commonly misspell names in queries, due to mistyping or incorrect spellings. Therefore, a friendly user interface should be able to "guess" what a user actually means and find a solution which closely matches a user's query.

1.3 General Description of the System

The light weight directory (lwd) system is a distributed directory service system. We designed and implemented this system to achieve the goals outlined in the previous section. Figure 1.1 shows the architecture of the lwd system. In the figure, each dotted box represents a subsystem (local directory service) in the lwd system. Each subsystem consists of a database and provides functionalities for information storage, information lookup, and information maintenance. Subsystems exchange directory information according to an information exchange protocol.

- Information storage. Directory information is stored in databases. The data structure for an entry is organized to be a key/content pair, with a person's name taken as the key and his email address along with other properties stored as content under the key. The system processes duplicate names when storing and updating information in a database.

  An auxiliary key file is used to enchance the performance of the system. It stores all keys in a database and is used to search for a partial name or a regular expression instead of searching a complete database. Since the key file is much smaller than the database, the search can be accomplished quickly.

- Information lookup. Directory information in a database can be looked up in one of the following three ways:

  - Exact full name match.
  - Regular expression.
  - Approximate match for partial name.

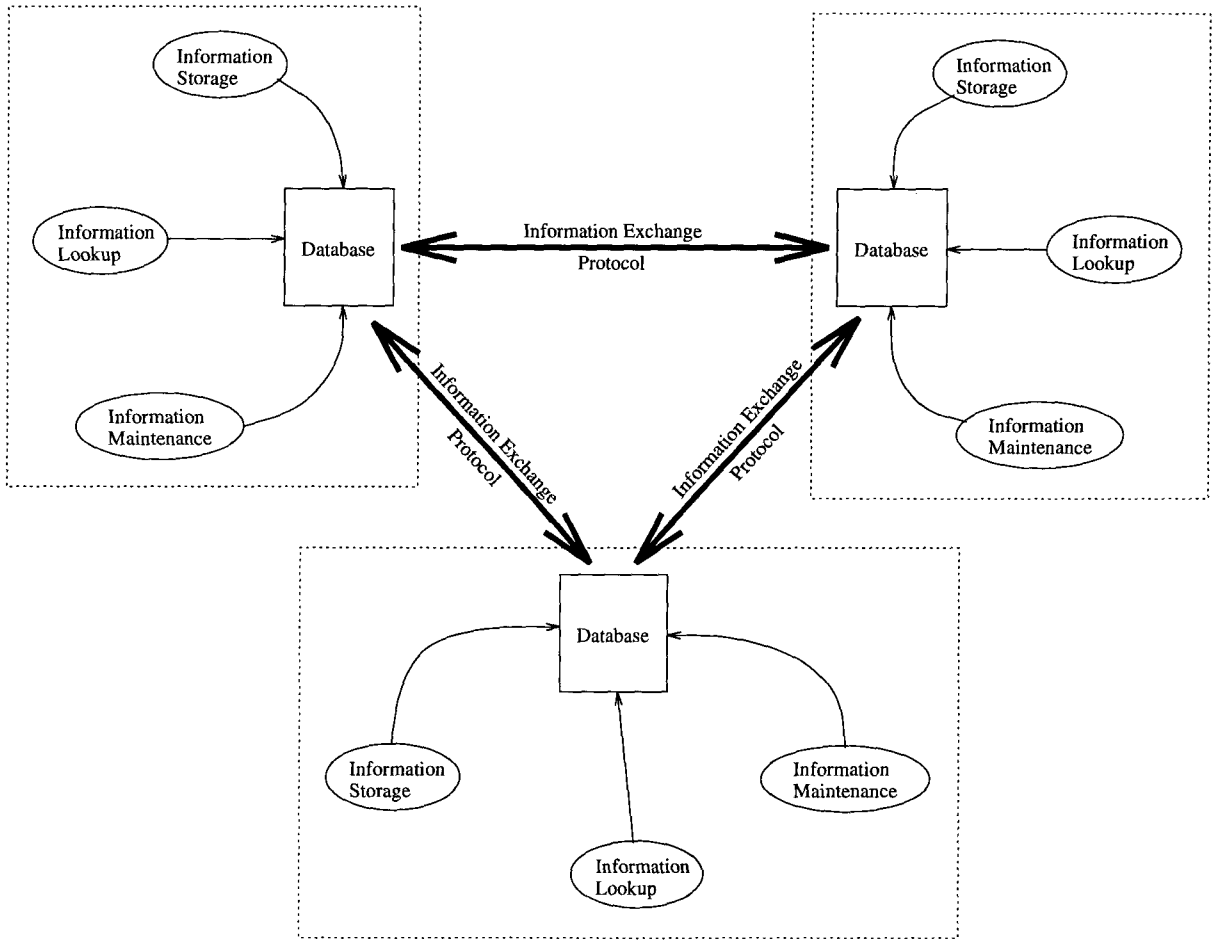


Figure 1.1: The Architecture of the Lwd System

- Information maintenance. Information about people are collected from different sources. The system also provides *add* and *delete* operations to directly update directory information.

  The system keeps track of the usability of its directory information. For example, if an entry has not been accessed in a long period of time, the entry is considered outdated and is removed from the database.

- Information exchange. In order to store as much information at a local site as possible, the system provides a mechanism to exchange information between different sites in a distributed environment. In Figure 1.1, the information exchange protocol defines how sites exchange information.

The rest of this thesis is organized as follows. Chapter 2 introduces the local directory service of the lwd system, which consists of information storage, information lookup, and information maintenance at a local site. Chapter 3 focuses on two information exchange protocols, *the hash method protocol* and *the key method protocol*, and their implementations. Analysis and discussions of the two protocols are presented. Chapter 4 surveys other directory service systems. The advantages and disadvantages of the lwd system, in comparison with others, are highlighted. Chapter 5 concludes the thesis by a summary and discusses future work.

Chapter 2

The Local Directory Service

Lwd is a distributed directory system. Sites can be located on any system in the world reachable through the Internet. A *site* is defined as a place where a database, which stores directory information about people, is held. Figure 2.1 shows the configuration of the system. A user can submit a request and expect to get a response from the nearest database. Instead of transferring a request to a more distant site, the system keeps all the information locally. Sites exchange information with each other periodically.

This chapter discusses the organization of a local directory service of the lwd system, the ways in which information is collected from different sources and stored in databases, and lookup and update services the system provides to its users. The next chapter addresses the issue of information exchange between different sites.

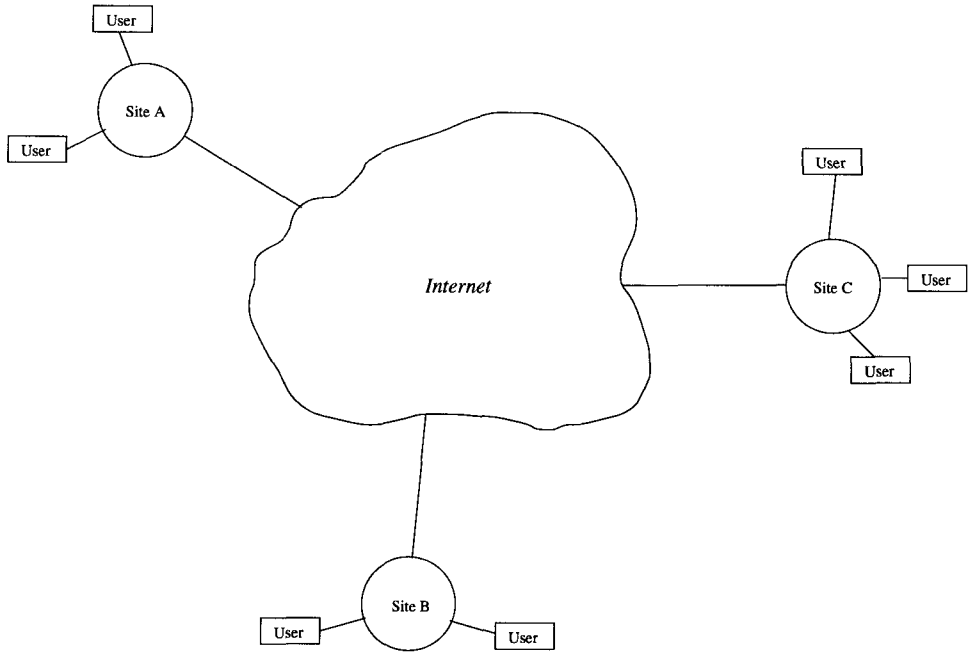


Figure 2.1: The Lwd Configuration

2.1 Overview of the Local Directory Service

The local directory service provides three kinds of operations to users. A user can add, delete, or lookup an entry in the system. In theory, a user can find information about anyone in the world, if the person has an email address. Information about people can be obtained from many sources. Currently we obtain information from EAN [Neufeld85] (an implementation of CCITT X.400), USENET [Tanenbaum88], and a local directory maintained by the Department of Computer Science at UBC.

The architecture of one site is shown in Figure 2.2. Due to the limited information available at one site, a site needs to exchange information with other sites that have databases. Basically, the architecture of one site consists of three parts.

- Information storage. How directory information is stored in a database and how the data structures for the entries are organized.
- Information lookup. The system allows users to look up directory information in three ways.
  - Exact full name match.
  - Regular expression.
  - Approximate match for partial name.
- Information maintenance. It deals with collecting information about people from different sources and converting them into a standard format. Users are provided *add* and *delete* services to update directory information.

In the lwd system, information about people is stored and retrieved. Therefore, one of the basic objectives of the system is to provide a convenient user interface. In

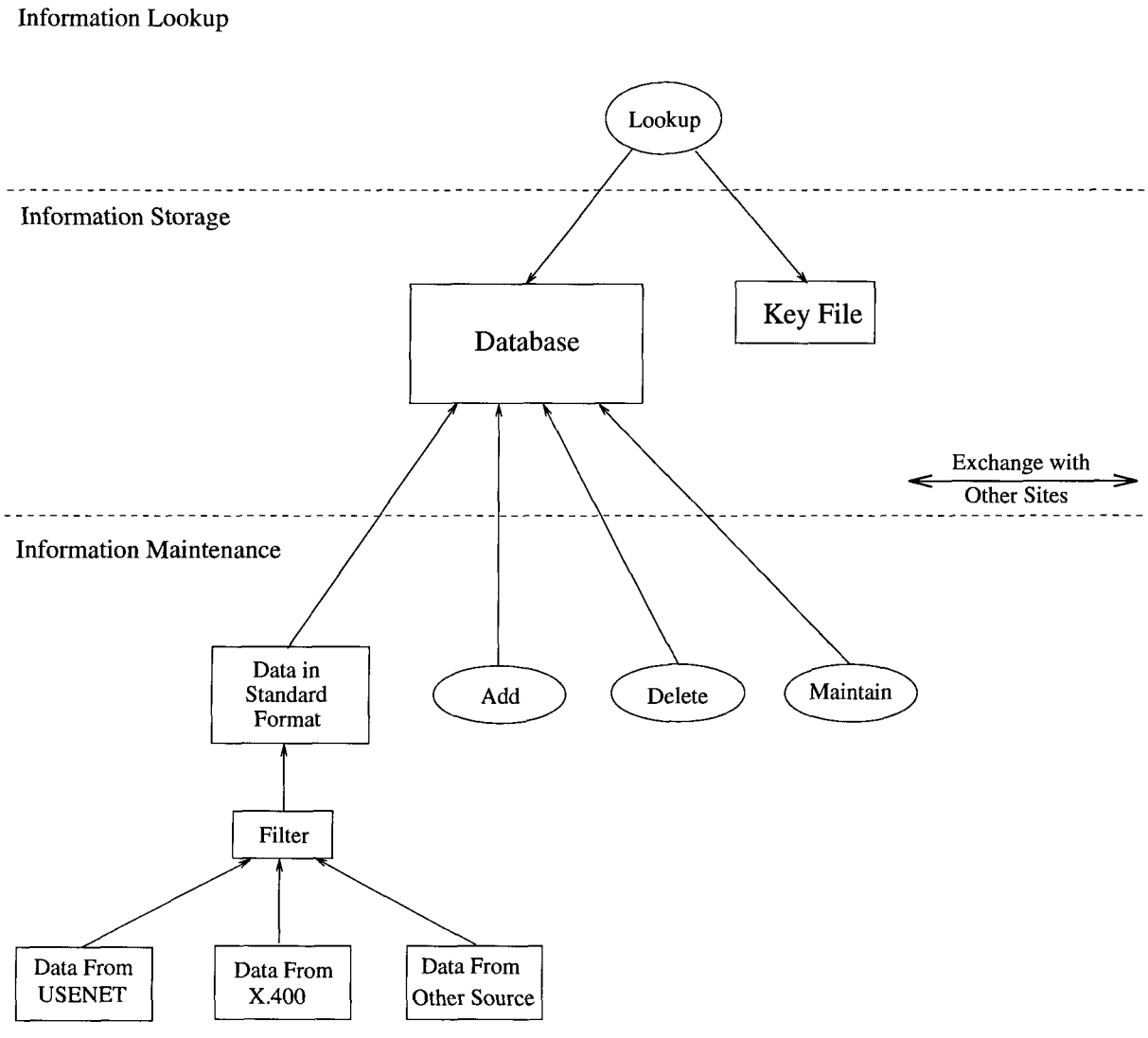


Figure 2.2: The Architecture of One Site

order to retrieve information, the user should only be required to submit information about a person's name. To meet this objective, we use people's names as keys and store other information under the corresponding names in a database[1]. We now discuss each individual part in detail.

2.2 Information Storage

2.2.1 The Structure of an Entry

In the database, a person's information is stored in an entry based on a key/content pair. A person's name is taken as the key and his email address along with other properties such as phone number, facsimile number, organization, and address are stored as content associated with the key. The date when the information was created is considered a special property and is also stored. All the information, flattened in a string format and separated by a delimiter ";", is stored as content. For example, *Hongbing Li*'s information as established on October 4, 1992 is as follows:

Name: *Hongbing Li*
Email: *lihong@cs.ubc.ca*
Phone: *822-3731*
Addr: *2366 Main Mall, Vancouver, B.C. V6T 1Z4*
Date: *Oct. 4, 1992*

In the database, the key for the entry is *Hongbing Li*. The content is *Email=lihong@cs.ubc.ca;Phone=822-3731;Addr=2366 Main Mall, Vancouver, B.C. V6T 1Z4;Date=4Oct1992*.

[1]Currently we use an efficient database library *Tdbm* (*dbm* with transactions) which was developed by Barry Brachman and Gerald Neufeld at the University of British Columbia [Brachman92].

From now on, we will assume that the content of an entry contains an email address only, and other properties are omitted for simplicity of discussion. Figure 2.3 shows the structure of a key and its content.

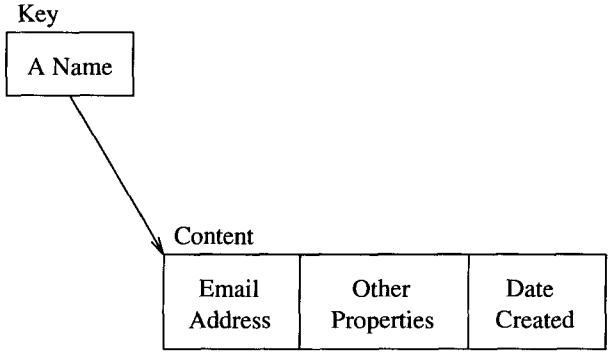


Figure 2.3: The structure of an entry

2.2.2 Storage

Before information is stored in the database, we see whether there is already an entry for the person in the database. If there is no such entry with the person's name as its key, we save the entry. If there is an entry, the two email values are compared. If they are the same as well, we assume the two entries are for the same person. In this case, we simply update the date property of the content. If the two email addresses are different, we save the new email address into the database under the duplicate name. It is possible that more than one email address of a person is stored under a duplicate name. Possibly the person has more than one valid email address or there are two people with the same name. It is also possible that one or two email addresses of the person are invalid. In the later case, the problem can be solved by the *maintain* program. By comparing the current date to the date stored in an entry, the *maintain* program can remove entries

which are considered inactive.

Currently, entries are stored in a *tdbm* [Brachman92] database library. *Tdbm* provides nested atomic transactions, volatile and persistent databases, and support for very large objects and distributed operation. While providing the new features, *tdbm* performs well in comparison with *ndbm* [Berkeley86], the most widely-used extensible hashing library under UNIX.

2.2.3 The Processing of Duplicate Names

In the lwd system, name duplication has to be handled properly in order to store information correctly. There are two cases where a duplicated name may occur. One is where different people have the same name. The other is where the same person has more than one email address. For example, there may be two people who have the name *Hongbing Li*, with email addresses being *lihong@cs.ubc.ca* and *hongbing@phoenix.princeton.edu*. But *Gerald Neufeld* may have two email addresses *neufeld@cs.ubc.ca* and *gneufeld@ee.ubc.ca*.

As discussed in section 2.2.1, we use a person's name as a key and the person's email address as a content. Information about a person is saved as a key/content pair in our database. When we encounter a duplicate name, there will be more than one content associated with the same name. It is possible for multiple contents to be physically stored under the same name key, but it complicates operations on the structure of the content. This is because all information for an entry is stored in a flattened way, so we would have to define another delimiter to separate multiple contents for different people. Moreover, the majority of names have no duplicates. If we define a new delimiter, we have to check every content to see whether it consists of more than one name.

We use a two level key/content structure to solve the above problem. In the first

level a duplicate name is taken as a key. Instead of the property content as shown in Figure 2.3, a special content is stored under the duplicate name key. The special content starts with a string "Duplicate Name" followed by internal keys for the property contents. In the second level, a random number is generated for each email address. It acts as a key, with the email address stored as a content under it. The structure of a duplicate name is shown in Figure 2.4.

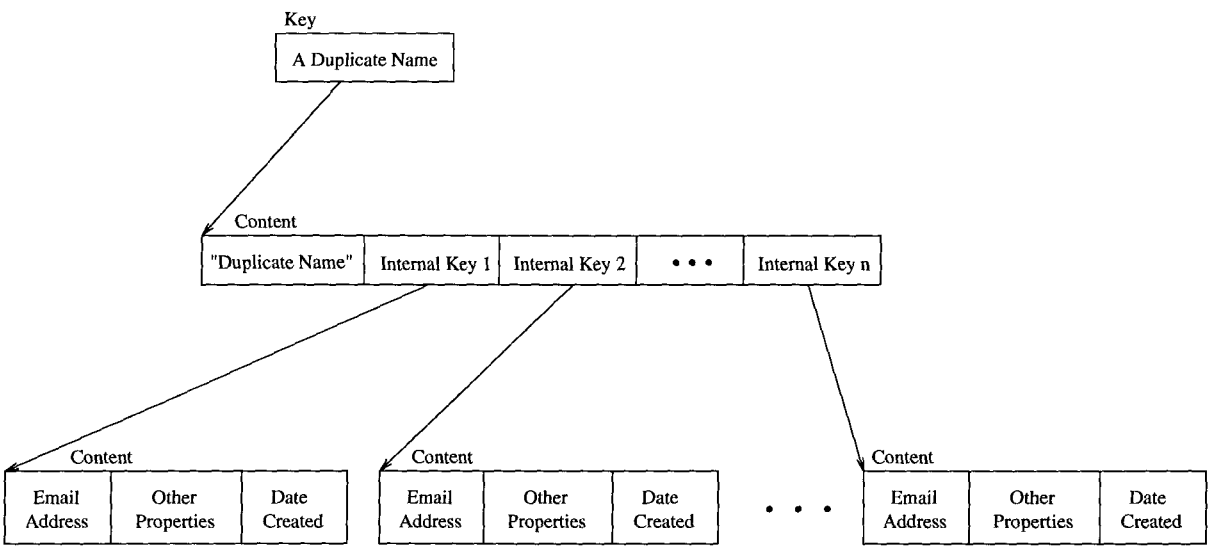


Figure 2.4: The structure of a duplicate name

The internal key is generated by a random number generator. To guarantee uniqueness when a random number is generated, an attempt is made to store the entry in the database. The insert mode of tdbm will not allow an entry to be stored using a key if an entry already exists for that key. Therefore, if it is determined that the storage attempt has failed, another key must be generated.

2.2.4 The Key File

Initially, we considered storing partial names as keys in the database. By providing a partial name, a user can expect to get the corresponding full name and then retrieve information by using this full name. Unfortunately, this method would make the database very large, since each name can be split into many partial names. Retrieving information becomes time-consuming, moreover, much of the information in a database is redundant. In order to solve this problem, we use a key file which stores all keys. When we store a key with its content in a database, the key (person's name) is added to the key file. The key file is an ordinary text file. Whenever there is a request for information retrieval based on a partial name, or even a regular expression, a fast and efficient algorithm *agrep* [Wu91] (approximate grep) is used to search for the person's name in the key file, rather than searching the complete database. Since the key file is much smaller than the database, the search can be accomplished quickly. This gives us increased performance and a database of reduced size.

2.3 Information Lookup

2.3.1 Lookup

The system provides a convenient and friendly user interface for looking up information. To look up a database, only information about a person's name is needed. Other information, such as affiliation, country, and so on, is not required.

Users have multiple choices for submitting different forms of a person's name. A person's full name, first name, last name, and a regular expression are all valid forms. If a user provides a person's full name, the system searches the database for an entry

which has the given name as its key. If such an entry is found and it is not a duplicate name, the system returns the person's email address. If a duplicate name is found, the system returns a list of email addresses by looking up every internal key stored under the duplicate name. If a user provides either a person's partial name or a regular expression, a list of matched full names is returned. The user can use one of the names to get the desired entry as in the case of looking up an entry by full name.

A regular expression is often useful when one is uncertain about the precise spelling of a name. Consider the case where a user would like to obtain the directory information of a person named *neufeld* but only remembers the first four letters in the name. The regular expression

'neuf[a-z]*'

can be used to express a pattern which starts with a string 'neuf', followed by an arbitrary string consisting of lower case letters 'a,...,z'. What follows is a script of a lookup operation when the user provided the regular expression 'neuf[a-z]*':

Enter name: 'neuf[a-z]*'

Matched names:
andrew neufeld
christopher neufeld
eric neufeld
gerald neufeld
kathryn neufeld

At this moment, the user is able to provide a full name to the prompt for a name:

Enter name: gerald neufeld

Email=neufeld@cs.ubc.ca

The syntax of regular expressions accepted by the system is the same as for *agrep*. Partial names are always enclosed in quotes to distinguish them from full names.

2.3.2 A Fast and Efficient Algorithm - *agrep*

Finding a person's name according to a regular expression can be considered a general string-searching problem. String searching in Unix is often performed by functions in the *grep* family. In our system a new tool *agrep*, which is similar to *grep* (or *egrep* or *fgrep*), is used for string searching. *agrep* was developed by Sun Wu and Udi Manber at the University of Arizona [Wu91]. It supports many kinds of queries, including arbitrary wild cards, sets of patterns, and general regular expressions. In addition to several new features, *agrep* supports most of the options available in the *grep* family. However, it is much more general, and usually gives better performance.

agrep has two significant features that the *grep* family does not support:

1. The ability to search for approximate patterns. This feature allows a user to make a query based on an inexact pattern, such as a misspelled name. For example, assume that a file foo contains a name hongbing. The command

   agrep -1 hongbin foo

   searches the file and returns hongbing, as well as other key words that can be obtained from hongbin with at most 1 substitution, insertion, or deletion.

2. Multiple patterns with AND (or OR) logical queries. For example, the command

```
agrep -d '^From' 'burger,pizza' mbox
```

returns all mail messages which contain either keyword **burger** or **pizza**. In the command, **-d '^From'** defines mail messages as records. *agrep* matches each record separately. The comma between the words **burger** and **pizza** denotes a logical OR operator. The command

```
agrep -d '^From' 'good;pizza' mbox
```

returns all mail messages which contain both keywords **good** and **pizza**. The semi-colon between the two keywords denotes the logical AND operator.

agrep not only supports a large number of options, but is also very efficient. Sun Wu and Udi Manber conducted performance analysis and concluded that *agrep* is competitive with the best exact string-matching tools that they could find at that time, and in many cases, it is one to two orders of magnitude faster than other approximate string-matching algorithms [Wu91]. We also conducted some simple experiments on *agrep*, *grep*, *egrep*, and *fgrep*. We collected 16 megabytes of data consisting of email addresses, associated names, and dates from news headers on the USENET. We first looked up string "Neufeld" from the data file, then regular expression 'N*eu[a-z]'. Each set of tests was run 10 times to get the experiment results. They are summarized in Tables 2.1 and 2.2.

In the tables, each number indicates the time (in seconds) spent by an *agrep* command or a command from the *grep* family on machines based on various architectures. The machines, a SUN 4/75, a MIPS M2000-8, and an HP 720, were under normal loads[2]. There is no data for *fgrep* in Table 2.2 because *fgrep* patterns are fixed strings and regular expression metacharacters are not allowed.

[2]The SPECint ratings for the machines are 20.24, 19.27, and 39.00 respectively.

	SUN 4/75	MIPS M2000-8	HP 720
agrep	2	9	1
egrep	11	8	6
grep	15	8	7
fgrep	18	9	6

Table 2.1: [agrep | egrep | grep | fgrep] Neufeld mid

	SUN 4/75	MIPS M2000-8	HP 720
agrep	10	23	10
egrep	23	18	35
grep	37	19	35

Table 2.2: [agrep | egrep | grep] 'N*eu[a-z]' mid

From the results in the tables, we can see that *agrep* is fastest except on the MIPS architecture, where the difference is not significant. These results provide supporting evidence for Sun Wu and Udi Manber's claim that *agrep* is very efficient for both exact and approximate pattern searching.

Because *agrep* provides more general functionalities as well as enhanced performance, it was chosen to perform pattern searching in our system.

2.4 Information Maintenance

2.4.1 Information Collection

The lwd system targets an environment that provides as many information sources as possible. Currently we use three sources: USENET, EAN, and our on-campus source. Other sources which become available can be easily incorporated into the system. Now we describe how we collect data from the three information sources.

Data from USENET

USENET [Tanenbaum88], started at Duke University and the University of North Carolina, offers a service called *network news*. Network news is divided into hundreds of newsgroups on a variety of topics. There are groups for popular programming languages, for common microcomputers, and for several operating systems, as well as groups for people offering or seeking jobs, people wanting to buy or sell things, and groups for many recreational activities and sports. USENET users can subscribe to whatever groups they are interested in.

Users can also post messages to news groups. Messages are broadcasted to all the machines in the world that carry the news groups. A user with a question or an opinion on some subject can post a message that may start a discussion eventually involving hundreds or thousands of people all over the world. The number of subscribers to USENET is enormous. Thus, USENET is a good source for getting information about people.

We can obtain information about a person, including the person's name and email address, from the header of a news message in USENET. A news message header looks like:

> Path: cs.ubc.ca!stephen.cs.ubc.ca!not-for-mail
> From: neufeld@cs.ubc.ca (Gerald Neufeld)
> Newsgroups: cs.systems
> Subject: THURSDAY SEMINAR
> Date: 16 Jun 1993 14:45:47 -0700
> Organization: Computer Science, University of B.C., Vancouver, B.C., Canada
> Lines: 32
> Distribution: cs
> Message-ID: <1vo4abINN7e2@stephen.cs.ubc.ca>
> NNTP-Posting-Host: stephen.cs.ubc.ca

In a news message header only two lines have useful information for us: the "From" line and the "Date" line. From these we can get the sender's name, email address, and the date when the message was sent. The date information will be used later to check whether a database entry is out of date.

By going through each news message in every news group, we can obtain a large amount of information about many people. We can then use a filter, which will be

described in the next section, to transform the information into a standard format such that each entry looks like:

Date=16Jun1993
Name=Gerald Neufeld
Email=neufeld@cs.ubc.ca

When we collect data from USENET, we need to choose an interval between two consecutive collections. In our news host, lifespans of messages vary in different news groups ranging from 7 to 21 days. We decided to perform a data collection every 7 days. In order to avoid collecting information from the same message more than once, messages which are older than 7 days are not processed. This allows us to save both time and space.

Data from X.400

One of the major design objectives of the lwd system is to store people's email addresses. On the Internet, the volume of electronic mail among people continues to increase. A well known advantage of email service is that it is very fast. People can expect a message delivered from coast to coast in a few seconds. Moreover, electronic mail has the speed of the telephone without requiring that both parties be available at the same instant. It leaves a written copy of the message that can be filed away or forwarded. Furthermore, a message can be sent to many people at once. Electronic mail messages are highly structured documents. In many systems, each message has a large number of header fields in addition to its contents. These include the sender's name and address, the recipient's name and address, the date and time of delivery, a list of people to receive carbon copies, the expiry date, the importance level, and so on.

Many telephone companies and PTT's are interested in offering electronic mail as a standard service to companies and individual subscribers. To prevent worldwide chaos, CCITT defined a series of protocols in 1984 for what it calls Message Handling Systems (MHS) in its X.400 series of recommendations [MHS88].

A distributed message system based on the CCITT X.400 recommendations, called EAN [Neufeld85] , was implemented at the University of British Columbia. In the EAN system there are log files which record information about all incoming and outgoing messages. Part of this information is electronic mail addresses of EAN users who send or receive messages, the users' names, times when the messages are sent or received, and machines where the messages are delivered to or received from. From the log files, we can get the names and email addresses of both the sender and receiver, and the time when a message was sent or received.

Data from Other Sources

In the department of Computer Science of the University of British Columbia, a database called "userinfo" has been developed. The database contains directory information of all people affiliated with the department. We have included the directory information in our system. There is another on-line database for all faculty and staff members on the UBC campus. We also plan to store this related information into our system. Other sources, from other universities or industrial organizations, can be easily incorporated into our system.

2.4.2 Information Conversion

A Standard Format

Since information is collected from different sources, we need to convert it into a standard format. By doing this the lwd store function is simplified since only one generic function is needed to store information from different sources into a database. In the standard format, each entry has three fields:

Date=
Name=
Email=

For example, the entry for *Hongbing Li* in the standard format is as follows.

Date=2Oct1992
Name=Hongbing Li
Email=lihong@cs.ubc.ca

Date indicates the time when the entry was first created, **Name** indicates the person's name, and **Email** is the electronic mail address of that person.

Information Conversion

The filter used for converting collected information into the standard format is implemented in Perl [Wall90]. Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It is very concise and convenient to use. The filter used to produce data in the standard format described above needs to take care of a few special cases.

- Processing of special characters.

  Generally the characters result from people misusing news groups or mail systems. This causes unknown, unexpected, or meaningless information. The special characters are:

  ^ { } @ ; * ? | < > ! /

  : _ " # ~ & % [ ] $ + =

  There are other cases in a name or an email address, such as more than one consecutive period "." or consecutive comma ",", several digits of numbers which go along with a name to represent the person's telephone number, multiple consecutive brackets, and so on. They should all be ignored by the system.

- Processing incorrect format.

  The correct format of collected information about a person is:

  "name" <email address> or

  email address (name)

  Take *Hongbing Li* as an example, the correct format should be:

  "Hongbing Li" <lihong@cs.ubc.ca> or

  lihong@cs.ubc.ca (Hongbing Li)

  The following information shows an example of incorrect format of information.

  "Q 8)̂" <castro@alm.admin.usfca.edu>

  diwadkva@ctrvx1.vanderbilt.eduVaibhav A. Diwadkar)

- Processing aliases.

  Some people like to use personal aliases rather than names in their messages. These aliases act like a person's name but can not be used for our purposes. For example,

"The Puzzled" , "The Black Eagle", and "A hopeless person", have to be removed manually, because there is no way to detect them from message syntax.

2.4.3 Maintenance

Information stored in the directory is maintained by a program called *maintain*. A special property, the date property, is used to check whether this entry is out of date. The date property is automatically updated whenever there is a lookup operation on the entry. Inactive entries, that is, those which have not been accessed for a certain period of time, are removed from the directory. There are two advantages which justify removing inactive entries:

- It keeps the directory at a reasonable size;
- It is very likely that an inactive entry contains out of date information.

2.4.4 Add

To collect information and enrich the database, a function *add* is provided to allow a user to add entries into the database.

In order to add an entry about a person to a database, a user must provide the name and the email address of the person. A user is also asked if other additional properties about the person are to be entered. The properties are described in section 2.2.1 as *other properties*. They are:

Phone: the person's telephone number;

Fax: a number to which a facsimile can be sent;

Org: the organization to which the person is affiliated;

Addr: the postal address of the person

The *add* function works as follows. The system first prints the prompt **Enter name:** to ask for a name to which properties are to be added. It should be pointed out that only full names are accepted by the *add* command. After receiving a person's name, the system prompts **Enter email:** for the person's e-mail address. The system asks whether the user wants to add other properties into the entry, in addition to the existing ones. A user can also modify an existing property by providing a new one to the system after confirming the option. What follows is a script of how Hongbing Li's information is added to a database. In the script, system prompts are in bold font, (*i.e.* **Enter Name:**). User input to the system is in sans serif font, (*i.e.* Hongbing Li).

Enter name: Hongbing Li

Enter email: lihong@cs.ubc.ca

Enter other properties: (y/n?) y

Phone: 822-3731

Fax: 822-5485

Org: Dept of Computer Science, UBC

Addr: 2366 Main Mall, Vancouver, B.C., Canada V6T 1Z4

After accepting the input, the system asks the user to confirm that information is to be stored in the database:

Add or not: (y/n?)

The user can respond to the prompt with `n` to avoid storing in the database, if an error in the input is detected. Otherwise, the new entry is written into the database.

Since we don't store entries with partial names as keys in a database, the full name of the person should be provided. It is recommended that a person's full name be given in the order of first name, middle name, and last name. However, the system is able to accept a full name given in any format.

There are no fixed formats for other properties, but we recommend users choose some reasonably acceptable format. For the organization where a person works, a user can provide either rough or detailed information. For example,

Univ. of British Columbia or
Dept. of Computer Science, Univ. of British Columbia or
System Group, Dept. of Computer Science, Univ. of British Columbia

2.4.5 Delete

To delete an entry from a database, a person's name is required. The system first checks whether the given name is a full name (first name and last name) or only a partial name. If the given name is a full name, the system looks in the database to find an entry with the given name as its key. If such an entry is found, the system determines whether the name is a duplicate key. If it is a duplicate key, the system asks the user to provide the email address of the person in order to decide which person's information the user wants to delete. After the entry is uniquely identified, it is deleted from the database. The following script shows the system and user interaction when the *delete* operation finds a duplicate key:

Enter Name: Hongbing Li
Duplicate Name
Please specify the email address: lihong@cs.ubc.ca

If the system is given a partial name, it searches in the key file to find a list of matched full names. The user chooses the name to be deleted. The user can then submit it to the system and expect that the corresponding entry will be deleted from the database, according to the algorithm of deleting an entry by full name described above.

Since a separate key file is used to store all the keys in a database, whenever an entry is deleted from the database, the key should also be deleted from the key file. In the case of a duplicate name, the name in the key file isn't removed, since there are at least two entries stored under one duplicate name.

An important point to remember is when there are only two entries under a duplicate name and one of them is deleted, the other one becomes unique. The duplicate name entry of two levels is changed into a normal entry of one level to store the unique entry.

A key is deleted from the key file using the following procedure: first, *agrep* is used to get the line number of the key in the key file. Then the key file is updated by removing the line from the original file.

Chapter 3

Information Exchange in a Distributed System

The previous chapter discussed site architecture and operations provided by the system. The architecture defines a local site in the system. All the sites are interconnected by the Internet to provide a distributed environment for the directory service system. This chapter discusses issues related to acquiring and organizing information in databases at different sites in the lwd system.

This chapter is organized as follows: Section 3.1 discusses the general issues involved in the lwd system. Section 3.2 describes and examines the hash method used in one implementation of the lwd system, to exchange information between different sites. Finally, Section 3.3 discusses another information exchange method, the key method, which is used in another implementation of the system.

3.1 General Introduction

Accompanied by the rapid growth of communications, computers have been interconnected by various local and wide area networks. People are not satisfied with centralized systems which have limitations such as speed and resources. It is imperative to develop distributed systems and applications in order to keep up with the rapid development of computer networks.

The lwd system is a distributed directory service designed and implemented to provide an Internet white pages service. A distributed system such as lwd possesses the following properties:

- *Extensibility.* A distributed system can be extended as the demand for service grows without replacing any of the existing components. In the lwd system, there may be an arbitrary number of sites connected into the system. It is much easier to extend a system which is distributed.

- *Sharing of resources.* For example, in a distributed system, each workstation may be diskless or have only a small disk (10-20 Mbytes) for temporary storage. Access to permanent files on a large disk can be provided to all of the workstations by a single file server. In the lwd system, a database at each site usually contains a large amount of information, for example, 36 Mbytes for 200,000 entries in a database implemented using *tdbm*. It can be installed on a central file server to allow other workstations in the distributed system to access it.

- *Uninterrupted availability.* When one of the components in a distributed system fails most of the work in progress need not be interrupted. Only the jobs performed on the failed component need to be moved to another available component. For

example, a user may move to another workstation if the one being used breaks down.

For a distributed directory service, a user expects that a request can be fulfilled locally or remotely if the request can't be done locally. Most of the distributed directory service systems currently available (*e.g.*, X.500) maintain a portion of information at each server in the system. To answer a query, such a system tries to locate the information locally. If such an attempt fails, the system may need to access a remote database. It usually takes much longer to conduct a remote lookup. In the lwd system all lookups are local, no attempt is made to do a distributed lookup. This implies that the local database must be as complete as possible.

In the lwd system, a large number of *sites* in the system are connected by the Internet, as shown in Figure 2.1. Each site in the system holds its own database. Since each site in the system collects data individually, different sites may contain different entries. To avoid remote accesses, the system tries to store as much information as possible in the local database. So, every site in the system may hold almost the same information as any other site in the system. Since every site in the system contains as much information as possible, a query to the system can be done locally and hence improves the response time of the system.

To store as much information at one site as possible, the system has to provide a mechanism which allows exchange of information among different sites. The general framework for information exchange is shown in Figure 3.1. In the figure, three sites are used to show how sites exchange information with each other. When one site sends a request to another, it expects to receive entries which do not exist locally but which are available at other sites. Each site can retrieve and send the information requested by other sites in the system, if such information is available locally. In other words,

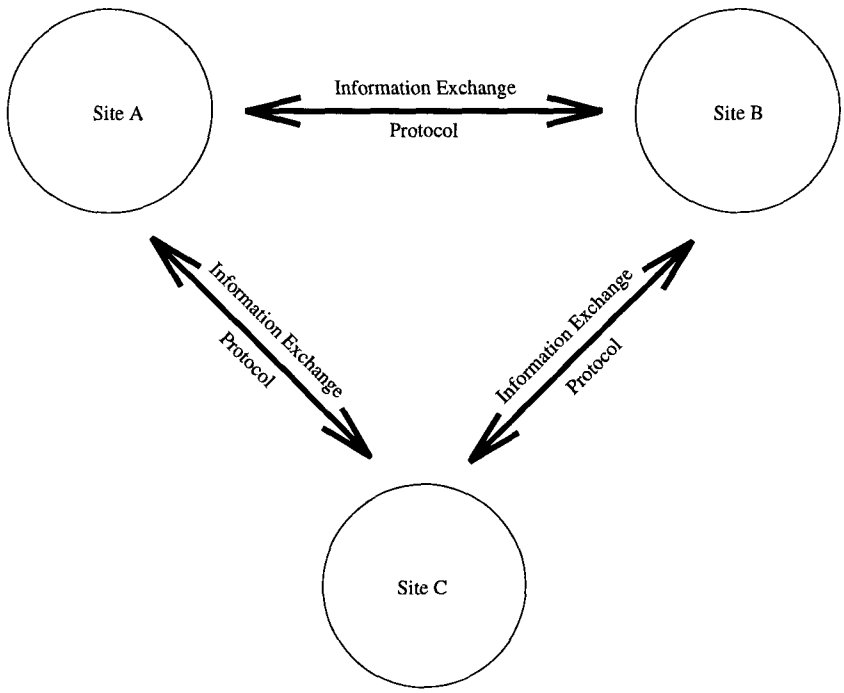


Figure 3.1: Information exchange between multiple sites

every site can act either as a client or a server. The major sequence of this client-server communication is described in Table 3.1.

Client	*Network*	*Server*
sends request for information exchange		
waits for reply	transmitting request	
.		receives request
.		executes request
.		sends reply
.	transmits reply	
receives reply and continues		

Table 3.1: The sequence of client-server communication.

In the lwd system, two different methods have been implemented for information exchange between different sites. They are described separately in the following sections.

3.2 The Hash Method

The hash method uses a bit map table to record the availability of entries at each site in the lwd system. After receiving a request for information from another site, this bit map table is used to decide whether the requested information is available locally. Since indices of the table are hash values of persons' email addresses, this method is called *the hash method.*

3.2.1 The Hash Method Protocol

As mentioned at the beginning of Chapter 2, each site in the lwd system holds its own database. In addition, each site also maintains an array of bits, called *the bit map table*, to record the availability of its information. The indices of the table are hash values of persons' email addresses. Every entry in the bit map table is a boolean value: the ith bit of the table is 1 if an entry whose email address' hash value is i exists in the local database. Otherwise the ith bit of the table is 0. When a site, say site A, requests information from another site, say site B, the following protocol, shown in Figure 3.2, is applied: Site A first sends its bit map table to site B. When site B receives the bit map table, it compares the table against its own. Site B only sends back those entries whose email address' hash values correspond to value 1 in its local bit map table but value 0 in the received bit map table.

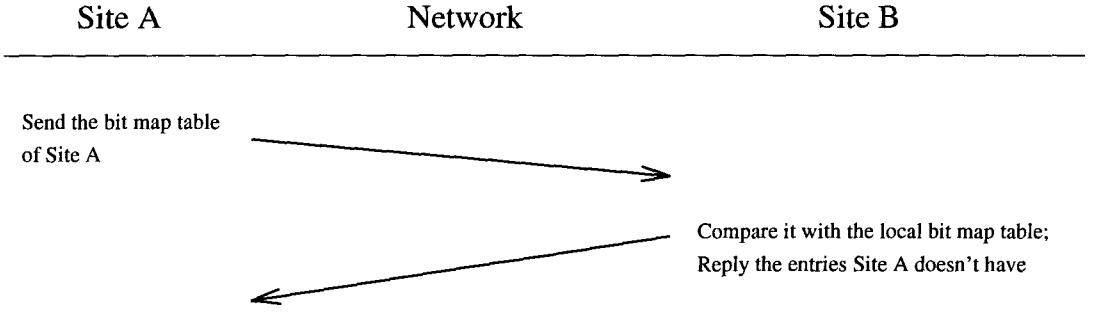


Figure 3.2: The hash method protocol

3.2.2 An Implementation of the Hash Method Protocol

The hash method is implemented as follows: Each site in the system holds a database and maintains a bit map table. The size of the bit map table (number of bits) equals

the number of different hash values available. Figure 3.3 shows an example of two sites (UBC site and SFU site) where each site's bit map table is of size 12.

First, a person's email address is converted to a natural number according to a hash function from *sdbm* [Yigit90], an ndbm-like hashed database library [Berkeley86]. The hash function is given in Appendix B. Initially, every bit in both bit map tables has the value 0. As the system evolves, entries may be added to either of the local databases. When an entry is added to a database, its email address is "hashed" to obtain a natural number i. The ith bit of the bit map table is updated to 1 to indicate that the entry is available at the site. The reason for choosing an email address, rather than a persons' full name, to obtain a hash value is due to the fact that different people may have identical names. Using email addresses increases the possibility that a natural number (index of the bit map table) corresponds to a unique entry in the database.

Twelve email addresses are shown in Figure 3.3, with respective suffixes (*e.g.* @cs.ubc.ca) omitted for simplicity. For example, email addresses *lihong* and *neufeld* in the figure are only a shorthand of *lihong@cs.ubc.ca* and *neufeld@cs.ubc.ca* respectively. Assume that the hash values of the email addresses *lihong*, *neufeld* and *brachman* are $1, 2$, and 3 respectively. From the bit map table at the UBC site, we can conclude that the entry of *lihong* is available at the UBC site, while that of *neufeld* is not, because the first bit in the bit map table is 1 and the second bit in the table is 0.

Suppose the UBC site makes a request to exchange information with the SFU site. It first sends its bit map table to the SFU site. After receiving the table, the SFU site performs an exclusive-OR on the received table and its local one to obtain a *the difference table*. A 1 in the table indicates the corresponding entry exists at one of the two sites but not at both. The Figure 3.3 details the result of this operation. Next, the SFU site applies a logical-AND operation to the difference table and its own information table.

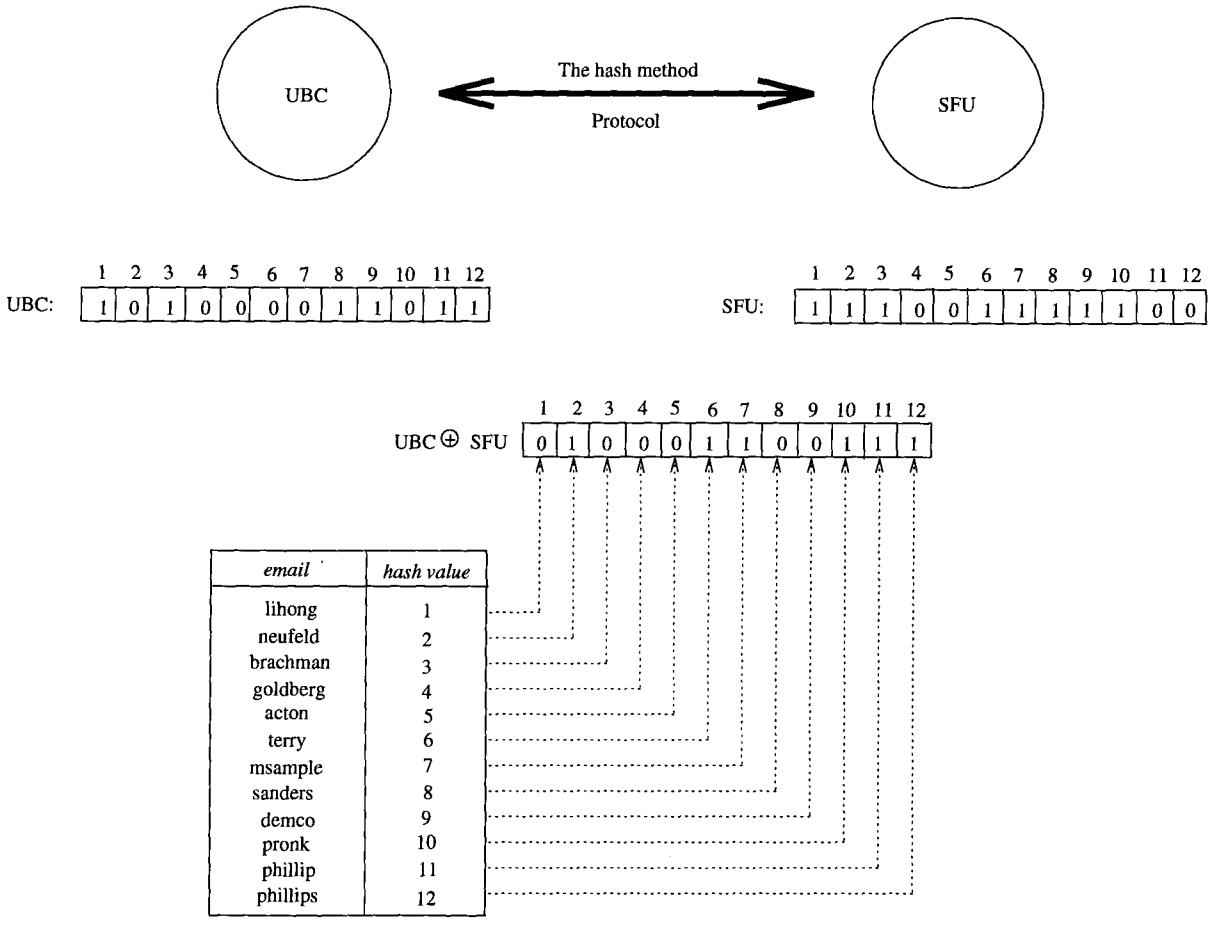


Figure 3.3: Bit Map Tables

This operation results in a table of size 12, where every bit which has value 1 means that the corresponding entry exists at the SFU site but not at the UBC site. Figure 3.4 shows the final table which indicates the entries available at the SFU site but unavailable at the UBC site.

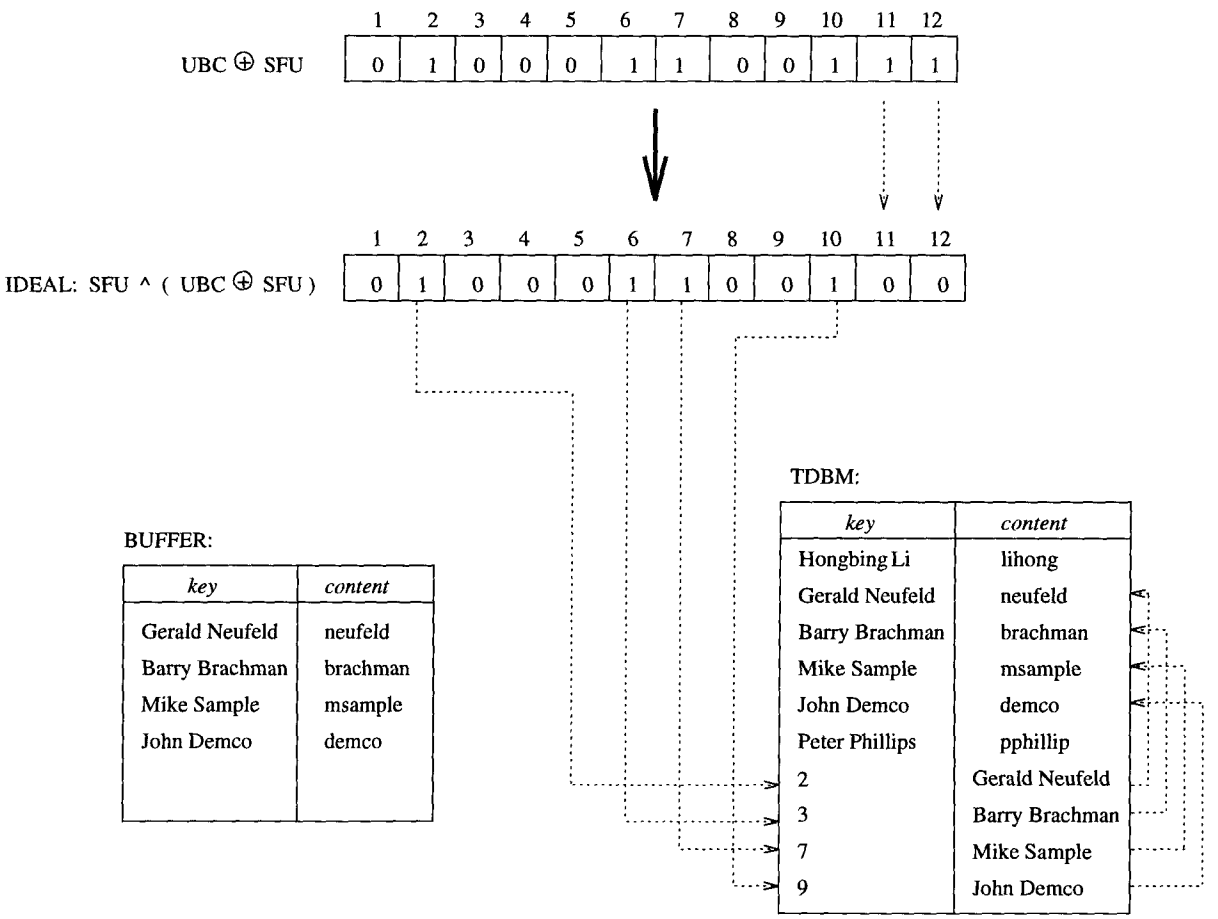


Figure 3.4: Hash method

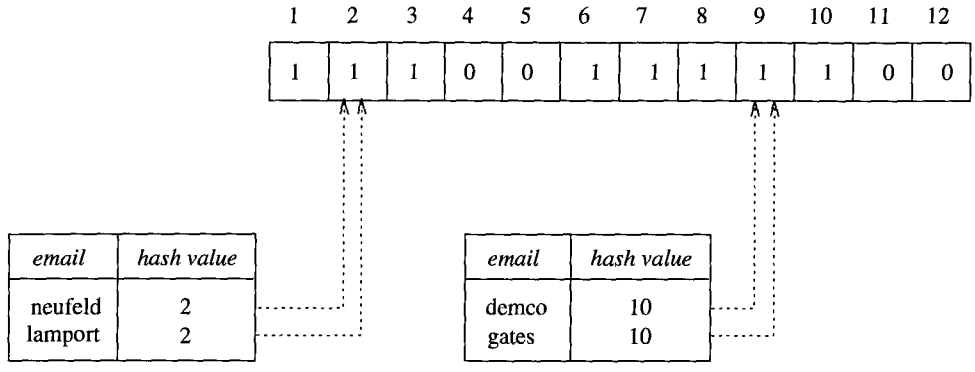
In the hash method, a local database needs to store another kind of entry, in addition to the entries with people's names as their keys. The entries use hash values as keys and

people's names as their contents to establish a relationship between hash values (of email addresses) and people's names. The purpose of doing this is to make retrieving an entry from the database easier. In Figure 3.4, the dotted lines pointing to the block TDBM illustrate the problem. Take *Gerald Neufeld* as an example: two entries related to him are stored in the database. One uses *Gerald Neufeld* as a key, with his email address as a content. The other uses the hash value 2 as a key and the name *Gerald Neufeld* as a content. Hence, when the site retrieves information from the database about people who's email address hash value is 2 according to the bit set in the final bit map table, it can first find the person's name *Gerald Neufeld* by retrieving the hash value key 2 and then use *Gerald Neufeld* as a key to find the content needed.

3.2.3 Analysis of the Hash Method

Associated with the adoption of hash methods is the problem of collisions. A collision means that there are two (or more) different email addresses which hash to the same value, so they share an identical bit in the bit map table. This means the lwd system may not be able to identify unique entries in the local database which should be sent to the remote site. The number of collision depends on the size of the hash table.

One solution to the collision problem is to create a *collision entry table* which records all the entries whose email addresses cause collision. When a request for information exchange occurs, this collision entry table is sent along with other *normal entries* (entries whose email addresses don't cause collisions) to the requesting site. The requesting site then decides which entries it does not have. The advantage of this solution is that it guarantees that every entry in a local database, which doesn't exist at other sites, will be exchanged. A disadvantage is that maintaining a collision entry table may consume a significant amount of storage and network bandwidth if the collision rate is high.



Collision Entry Table:

key	*content*
Gerald Neufeld	neufeld
John Lamport	lamport
John Demco	demco
Mike Gates	gates

Figure 3.5: Collision

To illustrate the problem in detail, let us take the bit map table at the SFU site in Figure 3.3 as an example. Suppose there are two email addresses *neufeld* and *lamport* which have a hash value of 2, and *demco* and *gates* which have a hash value of 10 as shown in Figure 3.5. There are four possibilities for a collision as shown in Table 3.2. The first is indicated by the first row in the table, indicating that there is no entry, whose email addresses are *neufeld* or *lamport* existing in the database. The second row shows there is an entry whose email address is *lamport* in the database. The third row means that there is a entry of *neufeld* in the database. Finally, the last row describes the case when both entries are stored in the database. All cases except the first will set the bit value to 1.

The problem becomes complicated since even if the collision bit is 1 in the bit map table of a requesting site, we cannot predict if the first, the second or both collision entries are stored in the database. To get exact information needed at one site, we generate a collision entry table. Whenever there is a collision, the system records both the entries in the collision entry table. The collision bit in the bit map table is set to 0. When a site, say SFU, receives a request for information exchange from another site, say UBC, it compares the bit map tables at the two sites and sends back the normal entries which the UBC site needs. In addition, SFU sends the contents of its collision entry table. At the UBC site, after storing the normal entries SFU sent, it saves the contents of the collision entries table in the database. At same time, it concatenates the contents with its own collision entry table.

To reduce the probability of collisions, one could use a larger set of hash values. In the lwd system, it means that a larger bit map table should be used. In the hash algorithm we adopt, the maximum hash value can reach 2^{32}. That means that 2^{32} should be chosen as the size of the bit map table. However, this causes a significant overhead in both speed and storage space. First, whenever information exchange occurs, a very large

neufeld	*lamport*	the bit value
n	n	0
n	y	1
y	n	1
y	y	1

Table 3.2: The bit value when collision occurs

bit map table, containing 2^{32} bits, is transferred between two sites. This causes heavy traffic and slows down the performance of the network. Also, a local database typically contains around 250,000 entries, therefore, only 0.6% of the bits in an bit map table are used.

A solution for reducing network traffic and unnecessary storage space is a smaller bit map table. However, smaller bit map tables inevitably lead to an increased probability of collisions. An experiment was conducted to study the relationship between the size of the bit map table and probability of collisions. In the experiment, the original hash algorithm was used to generate a hash value h for a given email address. This hash value is in the range $[0, 2^{32} - 1]$. The index (hash value) to the bit map table is the number

$$h \textit{ modulo } p$$

where p is the prime number closest to 2^{24}, 2^{23}, and so on. Table 3.3 shows the result of this experiment.

In the experiment, there are a total of $50,000$ entries in a local database. The entries, consisting of email addresses associated with person's names, are obtained from three different sources: USENET, EAN, and our UBC local directory as discussed in Section 2.4.1. The last column of Table 3.3 shows the minimum number of bytes of data

n	nearest prime number (p) to 2^n	number of collision entries (c)	collision rate	number of bytes to be transferred
24	16,777,213	1,061	2.1%	2,203,252
23	8,388,617	1,142	2.3%	1,162,776
22	4,194,301	1,327	2.65%	656,988
21	2,097,143	1,645	3.29%	426,644
20	1,048,573	2,392	4.78%	370,272
19	524,287	3,578	7.16%	423,336
16	65,537	17,803	35.6%	1,788,492

Table 3.3: Using Different Ranges of Hash Values

to be transferred during an exchange of information. These numbers are obtained by the formula:

$$p \div 8 + c \times 100 \tag{3.1}$$

In (3.1), $p \div 8$ is the size of a bit map table in bytes. We assume each entry occupies 100 bytes on average. $c \times 100$ is the size of the collision table in bytes. According to the previous discussion, when a site requests an information exchange, it sends its bit map table, which has $p \div 8$ bytes, and its entire collision table, which has $c \times 100$ bytes. We use the numbers in the last column to approximate network traffic in the situation. As shown in the table, collision rates range from a little over 2% to over 35%. This experiment reveals that when the number of entries in a database is $50,000$ and the average number of bytes per entry is 100, choosing the nearest prime number to 2^{20} as the size of bit map table requires the least network traffic during information exchanges. It should be pointed out that the conclusion may change if the number of entries in a database, or the average number of bytes per entry, changes. Nevertheless,

the significance of the experiment is to provide a method for choosing a bit map table size during further development of the lwd system.

3.3 The Key Method

The key method is implemented by using keys to decide on the availability of entries at a site in the lwd system. After receiving a request for information exchange, a site transmits its key file to the requesting site. The requesting site then decides whether the information is available at its own site. For simplicity, we call this the *key* method.

3.3.1 The Key Method Protocol

The protocol for exchanging information between two sites works as shown in Figure 3.6. First, site A sends a request to site B. After receiving the request, site B sends its key file to site A. Site A checks the received key file against its local database to find those keys, called *non-existing keys at site A*, which are in site B's key file but not in site A's local database. Site A sends the list of keys back to site B. Site B retrieves all the entries on the list and sends them back to site A. When site A receives these entries from site B, it saves them in its local database. The protocol gives site A all the entries it does not have which exist in site B's database.

3.3.2 Analysis of the Key Method

The key method solves the problem of information exchange between two different sites in a distributed system. The data used in the analysis of the key method is identical to the data used in the hash method. The size of the key file is 720,604 bytes when the

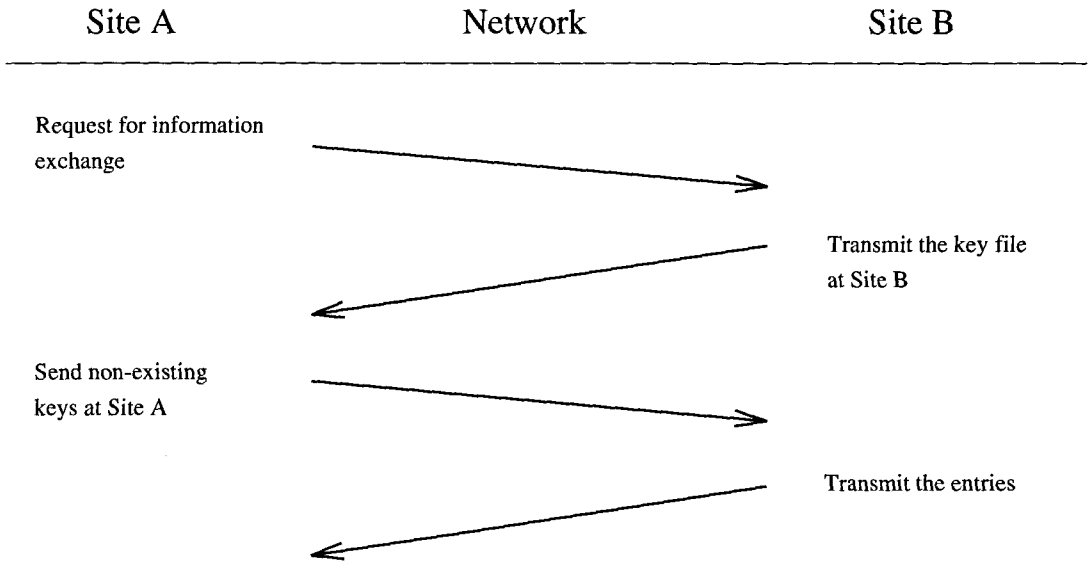


Figure 3.6: The key method protocol

number of entries in a database is 50,000. A disadvantage of the key method is, due to duplicate names, it does not guarantee complete information exchange. The following example highlights this issue: suppose that there are two different individuals who share the identical name *Hongbing Li*. The database at UBC contains the entry for one, and the database at SFU contains the entry for another. For the sake of easy identification, the individuals at UBC and SFU are identified as *Hongbing Li-U* and *Hongbing Li-S* respectively. Due to the identical name, UBC and SFU sites use the same key (the individual's name *Hongbing Li*) to point to different entries for different individuals. When the UBC site requests an information exchange with the SFU site, it expects to receive the entry *Hongbing Li-S* since it does not exist in its database. However, when the UBC site receives the key file from the SFU site, it is not able to identify that *Hongbing Li-S* is different from *Hongbing Li-U*. Therefore, it does not request *Hongbing Li-S* by putting the key *Hongbing Li* in the non-existing key list. The consequence of

this is that the SFU site will not retrieve and send *Hongbing Li-S*'s entry back to the UBC site.

Based on the discussion above, we can see that the key method is suitable for a system where the duplicate name rate is very low or lack of complete information exchange is considered insignificant.

3.3.3 ASN.1 Definition for the Protocol

Communication between any communicating entities requires a clear specification of the format and type of information to be exchanged. The lwd system is designed to serve a very large and diverse user community in a heterogeneous environment. The complexity of the data structures exchanged by distributed applications in a heterogeneous environment necessitates a general tool, called an *external data representation language*. In a heterogeneous environment, interconnection of different systems requires:

- Interconnection of different computer environments. This addresses different representations of the same data on different computers. For example, an integer may be represented in 2's complement format on one computer and in 1's complement format on another. During an information exchange between two different computers in the system, it is necessary to ensure the information is properly conveyed, regardless of how the information is represented on both sides of the exchange.

- Interconnection when different communication applications are implemented in different programming languages. To support this type of heterogeneity, it is necessary to ensure that the external data representation language is not dependent on a particular programming language.

- Interconnection of different applications systems. Although different application systems may have been written in the same programming language and on the same computer and operating system, they may be vastly different in the way in which they represent messages. An example is two different proprietary email systems.
- Interconnection of heterogeneous networks. In this case, the form of the data should be able to be altered without changing the type information. This means the form should be decoupled from the type.

Abstract Syntax One (ASN.1) [ASN188] with its encoding rules is an external data representation language defined by ISO and CCITT, which supports the forms of heterogeneous interconnection mentioned above. It provides a standard format of information which can be understood by different computers in a heterogeneous environment. In the *Key Method* implementation of the lwd system, ASN.1 was chosen to define the protocol for information exchange. The rest of this section is devoted to a brief introduction to the ASN.1 definitions.

In the lwd system, one site accepts a request from another site, performs the required operations, and sends back the requested information. The types, operations and their arguments between two sites are defined below in ASN.1:

```
LwdDirectory DEFINITIONS::=BEGIN

 -- the ASN.1 definitions describe types, operations and their arguments
 -- between two sites to exchange information in the lwd system.

 -- operations supported by the system
```

```
TransmitKey  OPERATION
             ARGUMENT  ReqMsg
             RESULT  AllKeys
             ERRORS  {noKeyFile, ServiceError}
     ::= 1

SendEntry    OPERATION
             ARGUMENT  KeysNeeded
             RESULT  Entries
             ERRORS  {ServiceError}
     ::= 2

 -- argument types required by operations

ReqMsg ::= PrintableString
KeysNeeded ::= PrintableString

 -- result types returned by operations

AllKeys ::= PrintableString
Entries ::= SET OF Entry

 -- Entry in directory

Entry ::= SEQUENCE {
             key    PrintableString,
             dptr   INTEGER,
```

```
            desc    OCTET STRING
            }

 -- errors returned by operations

noKeyFile     ERROR PARAMETER None ::= 1
ServiceError ERROR PARAMETER None ::= 2

END
```

Sometimes it is necessary for a site to return an error, rather than a result, to the requester if an anomaly is detected. The defined error types are:

`noKeyFile`: indicates a problem with the key file at the requested site, for instance, there was no key file found.

`ServiceError`: either a database problem or some communication problem, for example, after the requesting site A receives the key file from site B, site A asks for entries it doesn't have locally by sending the non-existing key list to site B. However, some entries can't be found in the database at site B.

3.3.4 Implementation

The key method is implemented by means of key files at different sites. Each site in the system holds a key file as discussed in Section 2.2.4. It includes all the entry name keys in a local database. Instead of transferring a bit map table, as in the hash method, a site transfers its key file to the requesting site, as shown in Figure 3.7.

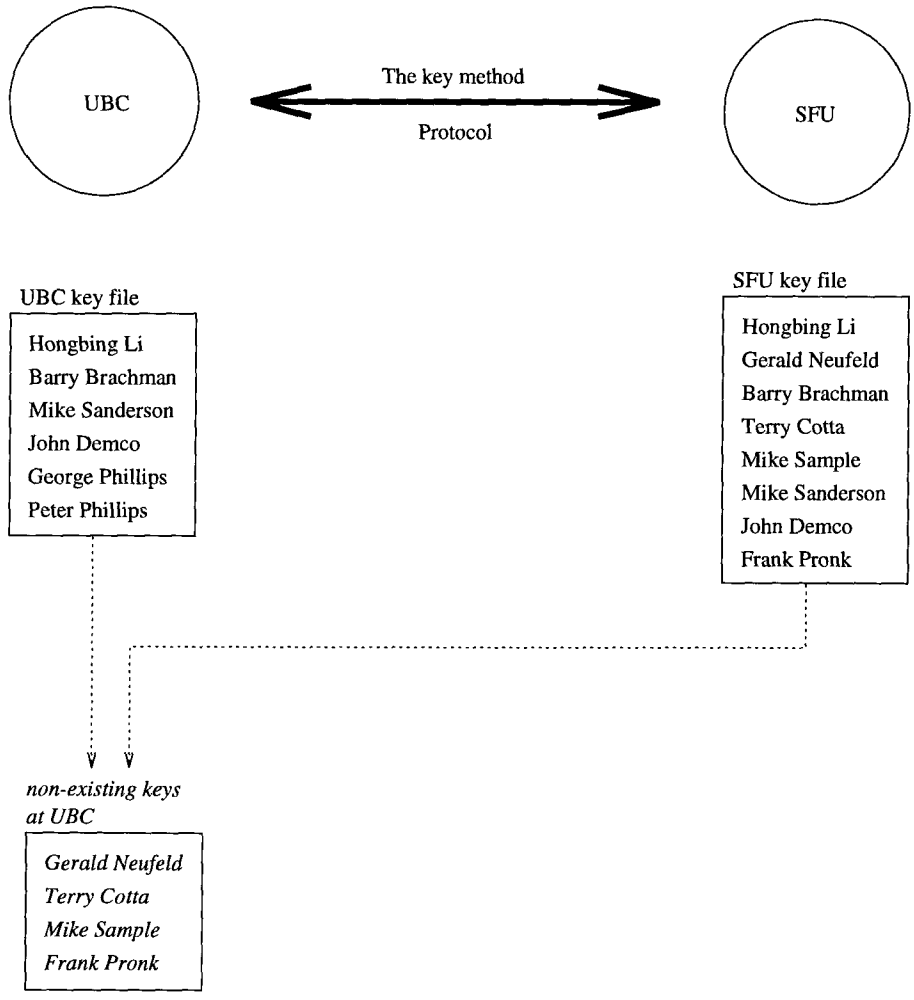


Figure 3.7: The key method

When site A receives a key file from site B, it needs to find those keys from the key file whose corresponding entries do not exist in site A's database. The lwd system uses a feature of DbmStore() in *tdbm*, which allows efficient detection of whether an entry corresponding to a given key exists in a database. Then the requesting site sends the non-existing keys to the requested site. After receiving the list, the requested site sends back the data needed by the requesting site. All requests and responses for information exchange are implemented in a *client-server* model, with the requesting site acting as a *client* and a receiving site acting as a *server*. Two sites communicate through TCP/IP.

3.4 Summary

This section discusses issues related to information exchange between different sites. The goal of the lwd system is to store as much information as possible at one site. This can make the response time for a user's query very fast. Two methods for information exchange, the hash method and the key method, were implemented. The hash method guarantees that all the information requested can be transferred to a requesting site. However, to get reasonable performance it requires choosing the size of a bit map table carefully. The key method is more intuitive than the hash method. The disadvantage is that, in the case of duplicate names, it does not guarantee complete information exchange.

Chapter 4

Related Work

In this chapter, several important projects and systems related to directory services are surveyed. Advantages and disadvantages of each system are briefly discussed, in comparison with the lwd system.

4.1 WHOIS Service

The WHOIS service [Harrenstien85] is a centralized TCP-based Internet directory service, developed by the SRI Network Information Center (NIC) in 1985. It provides netwide directory service to Internet users. The WHOIS server is accessible across the Internet from user programs running on local hosts. It collects and delivers information about registered DDN users. Such user information includes a user's full name, U.S. postal mail address, telephone number, and network email address.

The WHOIS system collects user information by means of user registrations. Therefore, it contains information about only a small fraction of Internet users, i.e., those who

have registered with NIC. This has prevented the system from providing a reasonable service to meet the demand of Internet white page service.

4.2 X.500 Pilot

The CCITT X.500 [DS88] defines a new OSI application and represents a milestone in the development of large-scale distributed systems. It provides a specification for a global on-line directory which holds data to be used to facilitate communications. It is designed to hold information on "objects" which are of interest in various areas of communication. Examples of such objects include people (*e.g.*, name, address, telephone number, facsimile number, and electronic mail address); organizations (*e.g.*, organization name, address, phone number, telex number, and facsimile number); and OSI application entities (*e.g.*, name and presentation address).

NYSERNet Inc. implemented a White Pages Pilot Project [Rose89] which is now managed by Performance Systems International. The project adopts X.500 as the basis for its prototype implementation. This facility is seen as the eventual replacement for the WHOIS service [Schwartz89]. X.500 involves a hierarchical collection of servers running at participating sites, each of which maintains directory information about that site. Browsing and searching operations are supported. Distributed operation of the Directory is achieved through a complex combination of protocols and the knowledge of network nodes known to each component system. Although X.500 is considered to be an advanced design for distributed directory, it is still under development and needs the sites' cooperation to run. It is not widely used now because of various constraints such as resources, and cooperation of organizations.

4.3 Profile

Larry L. Peterson at the University of Arizona developed a system called *Profile* [Peterson88]. *Profile* is an attribute-based naming service used to identify users and organizations. It supports queries over general types of objects, based on the Universal Naming Protocol defined in [Peterson88]. However, *Profile* focuses its effort more on the structure of query mechanisms for supporting directory services, and is not expected to be a practical directory service system available to a wide range of network users.

4.4 Netfind

Michael F. Schwartz et al. at the University of Colorado developed an Internet white pages directory tool called *Netfind* [Schwartz89]. The tool does not require the type of global cooperation many existing or proposed directory services demand. Global cooperation usually means running special directory servers at many sites around the Internet. At present, the tool utilizes information from USENET news messages, the Domain Naming System, the Simple Mail Transfer Protocol, and the "finger" protocol. Since they use a very large, administratively decentralized collection of information sources, such wide distribution increases the number of different failure modes the system can experience, because of the fluctuations in availability of the Internet and the sites being searched. For example, some sites disallow "SMTP" or "finger" protocols for security reasons or privacy concerns.

4.5 Knowbot Information Service

Ralph E. Droms at Bucknell University built a tool called a Knowbot Information Service (KIS) [Droms90]. KIS provides an Internet white pages facility by utilizing existing sources of information including *Profile*, *WHOIS*, the *CSNET* name server, *finger*, *MCI* Mail, the *MIT* white pages, and the *X.500 Pilot*. Since Droms' system is oriented towards providing a front-end to a variety of different information sources, it provides a translation between query formats used on different systems. It fulfills user queries by invoking one, or a combination, of services mentioned above. An advantage of this system is that it utilizes many information sources and thus has a high success rate in fulfilling users' query. One disadvantage is that it relies heavily on the global network and performance of other computing facilities in the network. Therefore, its response time to user queries is relatively long.

4.6 Comparison of Internet White Pages Facilities

In the Table 4.1, six directory service systems are compared. Some of the data used in the table is taken from Schwartz's report [Schwartz89]. What follows is a brief explanation and discussion of each row listed in the tables.

- Query Structure. This refers to support for making meaningful queries. To submit a query to the lwd system, it is sufficient to provide only a person's name, whereas other systems require more information such as a hierarchical structure (X.500 Pilot) or a set of keywords (*Netfind*). The lwd system also provides error-tolerance to partial names by conducting "approximate matching". This means that the system may find the correct answer to a query, even if the query uses a misspelled

Metric	LWD	Netfind	WHOIS	X.500_Pilot	Profile	KIS
Query Structure	User name, partial names, soundx names	User name near matches; wide variety of organizational keywords	User name substrings	Regular expressions over typed fields	Regular expressions + matching operators over typed fields; path preferences	Regular expressions over typed fields
Response Time (seconds)	1-2	3-45	8-45	14-182	1-2	10-102
Information Sources	USENET, EAN, UBC Directory	USENET, Domain Naming System, The Simple Naming Protocol, finger	Registration		Administra-tive Data	WHOIS, Profile, X.500 Pilot, the CSNET Name server, finger, MCI Mail, the MIT White Page
Individuals Reached	185,569	1,147,000	70,000	100,000	5,000	390,000

Table 4.1: Comparison of Internet White Pages Facilities

name. The complexity of the query structure reflects the complexity of a user query from a user's perspective.

- Response Time. The response time depends on various circumstances. For the X.500 Pilot, the low end represents a local query directed to a specific individual, while the high end represents a query to a non-existent individual by providing a flattened organization string. For KIS, the low end represents querying a single fast information source, and the high end represents querying a set of 9 different information sources. Although the circumstances are different, the numbers in this row show the significant advantage of the lwd system, which has the quickest response time. This is due to the fact that the lwd system stores all the information locally, so queries are performed at a local site which does not need to search for other resources and rely on network traffic and performance. Numbers in this row indicate that X.500 Pilot and KIS perform more slowly because of this.

- Information Sources. This row shows the information sources each system is currently using.

- Individuals Reached. This is the number of individuals each system can reach at the time of the survey. Numbers in this row indicate that the *Netfind* system reaches the largest number of users and KIS ranks second. This is because of the information sources they used (see the row "Information Sources".) Although the lwd system uses only USENET, EAN and a UBC local directory, it has reached a reasonably large number of individuals. This row measures an aspect of the effectiveness of a directory service system.

Chapter 5

Conclusions

5.1 Summary

This thesis presents the lwd system, a distributed directory service system, developed at the University of British Columbia. The lwd system provides an Internet white pages service.

Several major issues concerning the lwd system are discussed in this document: Chapter 2 presents the lwd system's local site architecture, information collection, organization and operations for database management and user services. Chapter 3 focuses on the network-wide architecture of the lwd system, and the methods used to exchange information among different sites within the system. Chapter 4 briefly surveys several directory service systems related and compares a few important aspects of such systems to show the advantages and disadvantages of different systems and techniques used in implementations of these systems.

The lwd system is implemented in the C programming language and the *perl* script

language. It has approximately 5000 lines of C code, including header files and comments, and 350 lines of perl code. Up until August of 1993, the database at the UBC site approximately consists of 185,569 entries. Due to the information exchange mechanism of the lwd system, other sites of the system are expected to maintain a comparable amount of information as well.

From the perspective of system architecture, information exchange mechanism, and user interface, the lwd system possesses the following features which make the system unique in comparison to other systems surveyed in the Chapter 4.

- Simplicity of the system architecture. Although the lwd system is a distributed system, it does not require global cooperation, as most of other directory service system do. Usually such global cooperation is conducted in the form of coordinating a set of network servers, each of which maintains highly structured information (*e.g.* X.500). The lwd system adopts the strategy of storing all of the white-pages information locally. This allows all the users' queries to be fulfilled locally, without cooperation from other sites in the system. This strategy makes the lwd system unique and gives it an obvious advantage in terms of response time to users' queries. Among the systems surveyed in the Chapter 4, the lwd system performs the best in this area. Not requiring global cooperation also allows easy installation on other sites in the system.

- Mechanism of information sharing. To store as much information on each site as possible, the system provides a mechanism to collect and exchange information among different sites in the system. Two methods have been implemented separately to facilitate information exchange between different sites. One method is the hash method. It guarantees that all information maintained on one site can be transferred to another. However, to get better performance, it requires choosing

the size of a bit map table carefully. The other method is the key method, which is more intuitive and easy to implement than the hash method. The disadvantage of this method is that, if duplicate names exist, it may not be able to transfer all information to another site.

- Source of information. Currently, the lwd system obtains network users' information from three different sources, *i.e.* USENET, EAN - an implementation of X.400, and a local directory database operational at the Department of Computer Science at UBC. However, the architecture of the lwd system allows easy access to other sources to obtain as much information as possible.

- Flexibility of query. The system provides a simplified user interface. To submit a query to the system, a user only needs to provide a person's name as an argument, thus making queries easier. A user does not need to submit other information such as organization name. The system also provides some error-tolerance capability to users' queries by conducting "approximate matches" to user submitted partial names. This capability allows the system to find the right answer even though it is given misspelled names.

5.2 Future Work

Future developments of the lwd system can be done in the following areas, The focus of the work is to enhance the usability and practicality of the lwd system.

- Increasing the number of information sources. Currently, the lwd system collects information from three different sources. To make the system more effective and useful, it is necessary to enhance the ability of the system to access information

from other sources, such as those sources accessible to the *KIS* system.

- Experimenting with the lwd system in a practical "multiple-site" environment. The current implementation and experimentation of the lwd system is done at the University of British Columbia. Due to the limit of resource, we are not able to install the lwd system at any other sites outside of the domain of "cs.ubc.ca". It is important to create a real multiple-site environment to evaluate and improve the system.

- Improving the system services to users. For example, the current system provides addition, lookup, and deletion of entries in a database. Modification of entries is embedded in the addition operation: if a user wants to modify an entry, he can use *add* command to add new properties to a person's entry in the system without changing other existing properties (recall that *add* command allows user to modify properties of existing entries). However, providing a separate modification operation may make users' access to the database more convenient.

- Security of the system. For safety reasons, it has been suggested that the system should check a user's privilege before allowing him to delete or modify an entry in the database. It has also been suggested that the system should keep a record of modifications made to the database in order to trace sources of intentional damage to the databases.

- Integrating the lwd system into mail systems. It would be very useful to integrate the lwd system into a mail system, (*e.g.*, EAN, or the unix mail system), because the need for the lwd system usually arises when a user intends to send a mail but does not know the exact email address of the intended receiver. Therefore, providing a directory service in a mail system will give users convenience and improve the efficiency of the mail system.

Bibliography

[Peterson88] Larry L. Peterson, "The Profile Naming Service", *ACM Transactions on Computer Systems*, Vol. 6, No. 4, 341-364, November 1988.

[Schwartz89] Michael F. Schwartz, Panagiotis G. Tsirigotis, "Experience with a Semantically Cognizant Internet White Pages Directory Tool", *Journal of Internetworking: Research and Experience*, Vol. 2, No. 1, pp. 23-50, March 1991.

[Harrenstien85] K. Harrenstien, M. Stahl and E. Feinler, "NICNAME/WHOIS", *Request For Comments 954*, October 1985.

[Rose89] M. T. Rose and M. L. Schoffstall, "An Introduction to a NYSERNet White Pages Pilot Project", Technical Report, NYSERNet Inc., December 1989.

[Neufeld92-1] Gerald Neufeld and Son Vuong, "An Overview of ASN.1", *IEEE Networks and ISDN Systems*, Vol. 23, No. 5, pp. 393-415, February, 1992.

[DS88] CCITT, "Recommendation X.500, OSI: Specification of the Distributed Directory System", Data Communication Networks, Blue Book, Volume VIII, Fascicle VIII.8, Omnicom, 115 Park Str., S.E., Vienna, VA22180 USA, Nov. 1989.

[ASN188] CCITT, "Recommendation X.208, Specification of Abstract Syntax Notation One (ASN.1)", Data Communications Networks Open Systems Interconnection (OSI) Model and Notation, Service Definition, Blue Book, Volume VIII, Fascicle VIII.4, Omnicom, 115 Park St., S.E., Vienna, VA22180 USA, Nov. 1989.

[Kernighan88] Brian W. Kernighan and Dennis M. Ritchie, "The C Programming Language, 2nd Edition", Prentice-Hall, 1988.

[MHS88] CCITT, Recommendations X.400-X.420, Data Communication Networks Message Handling Systems, Blue Book, Volume VIII, Fascicle VIII.7, Omnicom, 115 Park St., S.E., Vienna, VA 22180 USA, Nov. 1989.

[Neufeld92-2] Gerald Neufeld, Barry Brachman, Murray Goldberg and Duncan Stickings, "The EAN X.500 Directory Service", Internetworking: Research and Experience, Vol. 3, pp. 55-81, 1992.

[Sample93] Michael Sample and Gerald Neufeld, "Implementing Efficient Encoders and Decoders for Network Data Representations", IEEE INFOCOM '93 Proceedings, Vol. 3, pp. 1144-1153, , March 1993.

[Coulouris88] George F. Coulouris and Jean Dollimore, "Distributed Systems Concepts and Design", Addison-Wesley Publishing Company, 1988.

[Clark89] David D. Clark, Van Jacobson, John Romkey and Howard Salwen, "An Analysis of TCP Processing Overhead", IEEE Communications Magazine, pp. 23-29, June 1989.

[Brachman92] Barry Brachman and Gerald Neufeld, "TDBM: A DBM Library With Atomic Transactions", Summer '92 USENIX, pp. 63-80, June 1992.

[Wu91] Sun Wu and Udi Manber, "Fast Text Searching With Errors", Technical Report, Department of Computer Science, The University of Arizona, November 1991.

[AT&T79] AT&T, dbm(3X), *Unix Programmer's Manual*, Seventh Edition, Vol. 1, Bell Laboratories, January 1979.

[Berkeley86] Computer Systems Research Group, Computer Science Division, EECS. ndbm(3), *4.3BSD Unix Programmer's Reference Manual (PRM)*, University of California, Berkeley, April 1986.

[Droms90] Ralph E. Droms, "Access to Heterogeneous Directory Services", *Proceedings of the InfoCom Conference*, pp. 1054-1061, June 1990.

[Comer90] D. E. Comer and R. E. Droms, "Uniform access to Internet directory services", *Computer Communication Review*, Vol. 20, No. 4, pp. 50-59, September 1990.

[Schwartz87] M. F. Schwartz, J. Zahorjan and D. Notkin. A Name Service for Evolving, Heterogeneous Systems. *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pp. 52-62, Austin, Texas, November 1987. Published as Operating Systems Review Vol. 21, No. 5.

[Semtaniuk91] Bohdan Semtaniuk, "Distributed operation of the X.500 directory", *Computer Networks and ISDN Systems*, Vol. 21, No. 1, pp. 17-40, March 1991.

[Tanenbaum85] Andrew S. Tanenbaum and Robbert Van Renesse, "Distributed Operating Systems", *Computing Surveys*, Vol. 17, No. 4, pp. 419-470, December 1985.

[Knuth77] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast pattern matching in strings", *SIAM Journal on Computing*, Vol. 6, pp. 323-350, June 1977.

[Boyer77] R. S. Boyer and J. S. Moore, "A fast string searching algorithm", *Communications of the ACM*, Vol. 20, pp. 762-772, October 1977.

[Wall90] Larry Wall and Randal L. Schwartz, "Programming Perl", *O'Reilly & Associates, Inc.*, 1990.

[Seltzer91] M. Seltzer and O. Yigit, "A New Hashing Package for UNIX", *Proceedings of the Winter Usenix Conference*, Usenix Association, pp. 173-184, January 1991.

[Seltzer92] M. Seltzer and M. Olson, 'LIBTP: Portable, Modular Transactions for UNIX", *Proceedings of the Winter Usenix Conference*, Usenix Association, pp. 5-25, January 1992.

[Neufeld85] Gerald Neufeld, John Demco, Brent Hilpert, and Rick Sample, "EAN: an X.400 message system", *Computer Message Systems*, IFIP, pp. 3-15, 1985.

[Yigit90] O. Yigit, "sdbm - Substitute DBM or Berkeley sf ndbm for Every UN*X Made Simple", sf sdbm source distribution, Dec. 1990.

[Tanenbaum88] Andrew S. Tanenbaum, "Computer Networks, 2nd Edition", Prentice-Hall, 1988.

Chapter 7

Appendix A

LWD(1) USER COMMANDS LWD(1)

NAME

lwd - light weight directory

SYNOPSIS

lwd [name]

DESCRIPTION

lwd allows users to lookup a person's email address by specifying the person's name.

Users have multiple choices for submitting different forms of a person's name. A person's full name, first name, last name or a regular expression are all valid forms.

In addition to a person's email address, lwd will supply other properties such as telephone number, organization and address about a given name. Currently, however, most of entries in the database don't have such information.

An lwd command with a name argument provides the person's

email address and possibly other related information.

MESSAGES

lwd will possibly supply other messages, described below, for a given name if it fails to find the requested email address.

Matched names
: If you provide a person's first name, last name, or a regular expression, a group of matched names are returned. You can then use one of the names to get that person's email address.

No name matched
: There is no information related to the person in the database. A group of approximate matching names are returned then.

Enclose regular expression in single quotes
: For a regular expression, one should always enclose the entire pattern argument in single quotes, i.e., 'pattern'. This avoids characters, which cause unexpected results when included in an unquoted pattern, since some characters may also be meaningful to the shell.

  A regular expression is written the same as for agrep. Please see the manual page for agrep for more information related to regular expressions.

Unmatched '
: There is no matching single quote for a regular expression.

```
LWD(1)                    USER COMMANDS                    LWD(1)


EXAMPLES
     example% lwd Hongbing Li
     ----------------------------
     Email=lihong@cs.ubc.ca
     ----------------------------

     example% lwd
     Enter name: Gerald Neufeld
     ----------------------------
     Email=neufeld@cs.ubc.ca
     ----------------------------

     example%lwd
     Enter name: 'neuf[a-z]*'

     Matched names:
     andrew neufeld
     christopher neufeld
     eric neufeld
     gerald neufeld
     kathryn neufeld

DEPENDENCIES
     lwd uses tdbm and the functions it provides to establish,
     maintain and operate on a database. tdbm is a dbm database
     with nested atomic transactions. It was developed by Barry
     Brachman in the Dept. of Computer Science of UBC.

FILES
     /cs/public/bin/lwd
     /cs/public/generic/src/network/lwd

SEE ALSO
     whois(1), agrep(1)

BUGS
     This is the first release of lwd. Any bug reports, comments
     or suggestions are ALWAYS appreciated. Please email them
     to the authors.
```

```
AUTHOR
     Hongbing Li <lihong@cs.ubc.ca>
     Gerald Neufeld <neufeld@cs.ubc.ca>



Sun Release 4.1      Last change: 31 Jan 1993                  2
```

Chapter 8

Appendix B

```
/*
 * From sdbm, an ndbm work-alike hashed database library
 * Author: oz@nexus.yorku.ca
 * Status: public domain.
 *
 * polynomial conversion ignoring overflows
 * [this seems to work remarkably well, in fact better
 * than the ndbm hash function. Replace at your own risk]
 * use: 65599   nice.
 *      65587   even better.
 *
 * [In one experiment, this function hashed 84165 symbols (English words
 * plus symbol table values) with no collisions. --Barry Brachman,
 * Department of Computer Science, UBC]
 */
u_long
sdbm_hash(str, len)
char *str;
int len;
{
        register u_long n;

        n = 0;

#define HASHC   n = *str++ + 65587 * n

#ifdef NODUFF
        while (len--)
```

```
                n = HASHC;
#else

        if (len > 0) {
                int loop;

                loop = (len + 8 - 1) >> 3;
                switch (len & (8 - 1)) {
                case 0:
                        do {
                                HASHC;
                        case 7: HASHC;
                        case 6: HASHC;
                        case 5: HASHC;
                        case 4: HASHC;
                        case 3: HASHC;
                        case 2: HASHC;
                        case 1: HASHC;
                        } while (--loop);
                }
        }
#endif
        return(n);
}
```