Freebrain: A Distributed and Scalable Interest-based Instant Group Communication System

by

Kalpak Dilip Kothari

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

Author .....
Department of Electrical Engineering and Computer Science
May 10, 2002

Certified by .....
Stephen J. Garland
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Freebrain: A Distributed and Scalable Interest-based Instant Group Communication System

by

Kalpak Dilip Kothari

Submitted to the Department of Electrical Engineering and Computer Science
on May 10, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Freebrain is a distributed and scalable interest-based instant group communication system that enables people and on-line databases or services to share their knowledge in a personalized way with those seeking help. The Freebrain architecture resembles a publish and subscribe system. People and databases subscribe to categories of questions they are interested in answering. When a user asks (*i.e.,* publishes) a question, an invitation is routed to the subscribers of the categories that match the content of the question. Human invitees can choose to enter a group conversation to answer the question. Machine invitees (on-line databases) can directly return results. The system achieves load balancing and fault tolerance using Chord for its subscription distribution. Instant-messaging (IM) is achieved using Jabber, an open-source, distributed, IM system. The main contribution of Freebrain is a powerful, new paradigm that enables users to obtain real-time answers from both people and on-line services using a single, unified IM interface without requiring any prior knowledge of the possible sources of information.

Thesis Supervisor: Stephen J. Garland
Title: Principal Research Scientist

Acknowledgments

I would like to thank the following people who helped me (directly and indirectly) with my thesis: my friend, Aakash Kambuj, for the initial discussion of the idea of Freebrain; Ronnie Misra, for helping me with the initial implementation and subsequent discussions about the design; most importantly, my advisor, Steve Garland, for allowing me to pursue my interest in Freebrain, for supporting and helping concretize the Freebrain system, and his suggestions that have tremendously improved the quality and presentation style of this thesis; other folks in the Network and Mobile Systems group, namely, Hari Balakrishnan for some ideas and suggestions and Magdalena Balazinska for the Chord code that Freebrain uses; and last but not least, my family and friends.

Contents

List of Figures

Real Time: Here and now, as opposed to fake time, which only occurs there and then.

On-line: The idea that a human being should always be accessible to a computer.

Chapter 1

Introduction

1.1 Motivation

Consider when you've spent hours on the Internet searching for an answer to an urgent or nagging question. The Internet presents you with a number of sources or systems where you can seek help, as depicted in Figure 1-1. Irrespective of whether you are a novice or an advanced Internet user, you must first ask yourself the question, "Who do I ask for help?" Let's examine your options. Popular search engines like Google [10] or content-based data servers allow you to find answers from machines. Real-time chat systems like Internet Relay Chat (IRC) [13], popular instant-messaging (IM) networks like Yahoo! Messenger or AOL Instant Messenger, private community-based IM systems like MIT's Zephyr [30], and mailing lists or newsgroups [11] allow you to find answers from other people. Thus, the Internet gives you a plethora of options to seek answers. These options fall into two broad groups—seeking help from machines (databases), and seeking help from people.

In spite of all the options we have for seeking help, we are often unsuccessful in finding answers. If we hope to find a quick answer, we may find ourselves spending too much time, or, perhaps, we many even give up the task all-together. If we try search engines, we may be inundated with too much information or, sometimes, with too little information. If we try mailing lists or newsgroups, we don't get instant answers. If we try chat systems, we have difficulty finding the right chat room to enter and ask

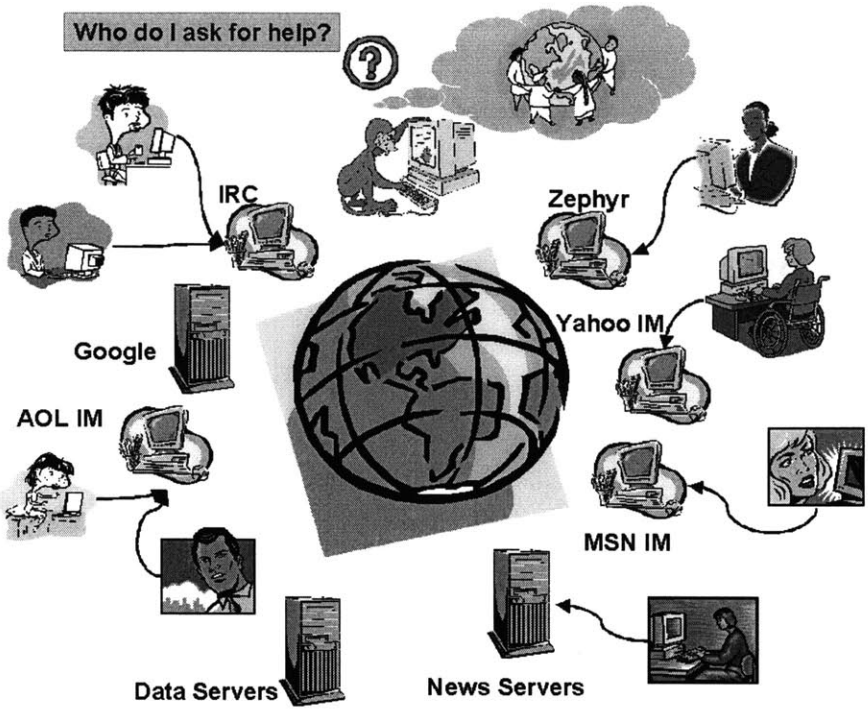

Figure 1-1: Present World - Who do I ask for help?

our question. Above all, there is a more fundamental problem. There are thousands of knowledgeable people as well as on-line resources, but all accessible in different ways. For an end-user seeking help, this presents a big problem, especially if he is looking for a quick answer. He faces a barrier to entry either because he does not know all the available options, or because it requires too much effort to communicate with people on the various systems. Finding the appropriate people or on-line resources in real-time is thus a challenge. As an end-user, he wishes the Internet would present itself as a more unified network where seeking help is as effortless as possible.

Can we build a system that simplifies the end-user's task of seeking help in real-time? Consider the world depicted in Figure 1-2, a world that focuses on people's interests, and a system that is aware of these interests. In this alternate world, the end-user simply asks his question once, and the system takes care of everything else—finding the right people who would be interested in answering or discussing the question. This is the world of Freebrain. It brings a universal access point to the end-user for real-time communication. For instance, if I ask the question, "Could

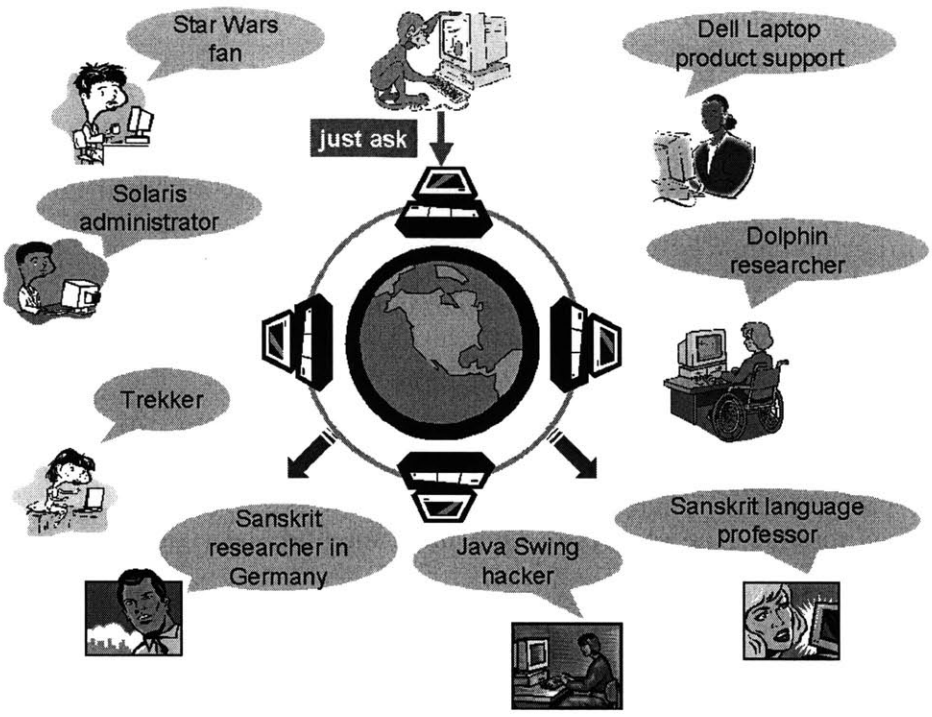

Figure 1-2: Freebrain World - I simply ask Freebrain

someone point me to a good on-line reference discussing the history of Sanskrit[1]?", as shown in Figure 1-2, Freebrain will route my question to the people who have expressed an interest in Sanskrit.

In this thesis, we discuss the architecture of Freebrain, a system that simplifies the end-user's task of seeking help in real-time. The primary idea of Freebrain is a system that allows people to ask questions and contribute answers in real-time based on interests. A powerful derivative of this simple idea is to involve on-line databases for contributing answers via the same real-time communication medium. This allows both machines (databases) and people to be accessible to end-users under one common medium. Freebrain tries to match a person seeking information not only with interested people but also with databases that can offer answers. One can imagine hundreds of ways on-line databases can provide us with useful information. For instance, if I wish to locate the office room of a member of LCS, I could potentially

[1]Sanskrit is an ancient Indian language, considered to be one of the oldest and most systematic languages in the world.

ask Freebrain a question, "What is the MIT LCS office for John Doe?", which in turn could route my query to an LCS directory database. Another example would be a medical dictionary that, when a user asks a question like "What is the dictionary meaning of the medical term patellalgia?", receives this query and returns the meaning of the word "patellalgia" to the user. The key benefit of Freebrain is the ability to receive real-time answers from both people and machines using the same medium. No present system does this.

We thus argue the need for a system like Freebrain because we believe there are many people (and databases) willing to offer help. The problem lies in the fact that, for real-time answers, the seekers of help may not find the right sources (people or databases) or may be forced to use multiple mediums (web searches, chat systems, IM networks, *etc.*). Moreover, existing systems do not provide an easy mechanism for quickly alerting these sources about a seeker's question. Freebrain is an endeavor to fix these problems and simplify a person's experience when he seeks real-time help.

1.2 Goals

If we seek real-time answers, we need a framework that allows us to locate others with similar interests and chat (have a real-time conversation) with them. While this is the primary goal, we also enumerate a set of intermediate and desirable goals that we aim to achieve in the design and implementation of Freebrain.

Real-time messaging — Freebrain should provide people the ability to ask questions and receive answers from other people or databases in real-time. Our focus in this thesis is on real-time text-based interaction, or instant-messaging (IM).

Subscribing to interests — In order to match people with common interests, Freebrain should allow users to subscribe to the categories related to their interests. Subscribing to a category (interest) means that the user is willing to receive messages (questions) that match that interest.

Simple interface — Freebrain should provide a very simple interface to end-users to ask questions and to respond with answers. As a result, Freebrain is integrated within the current IM paradigm, which is familiar to nearly every user on the Internet.

Automated category selection — Freebrain should be able to route a question without requiring the user to manually specify what interests (categories) the question belongs to. Freebrain may also provide the user with an option to manually specify categories to which a given question should be routed, but an automated category selection is essential to simplify the requirements for an end-user.

Integrate benefits of related systems — Freebrain should integrate the benefits of current systems that allow users to ask questions and receive help. In particular, it should retain the benefits of IRC, *viz.* allowing people to enter a multi-user conference room to have a conversation on a particular topic (see Section 2.1). Other important benefits from related systems like newsgroups are a friendly categorization of interests and web-based access to everyone to read, search, and learn from past conversations that Freebrain users have had (see Section 2.2).

Distributed and scalable architecture — In addition to ensuring that Freebrain meets all the aforementioned goals, we aim to produce a distributed and scalable architecture. This is necessary for achieving Internet-wide messaging at low cost and for ensuring that there is no central server or authority that can bring down the entire Freebrain network. Apart from the benefits of higher system availability, a decentralized architecture can also provide greater privacy to users.

The final outcome of our goals is simple — Freebrain must be able to enhance productivity of information retrieval in the form of real-time answers from people and on-line sources. In this thesis, we focus on the design and implementation of

the distributed and scalable interest-based instant group communication architecture that achieves our goal—enabling people and on-line information sources (databases) to share their knowledge in a personalized way with those seeking help using a common medium for real-time communication.

1.3 System overview

The basic Freebrain architecture resembles a publish and subscribe system. People and on-line information sources (such as databases) can subscribe to certain categories (or interests) of questions they are interested in answering. When a user (called an *inviter*) asks (publishes) a question, it gets routed to the subscribers (called *invitees*) of the categories that match the content of the question. A human subscriber that receives a question can choose to join a conversation, *i.e.,* a real-time multi-user group chat with the inviter, and help answer the question. If an invitee is a database it can directly return results from a query to the inviter.

The front-end or the client-side interface for communication (sending and receiving messages) is through instant-messaging (IM). We have also developed a preliminary web-based front-end for secondary functionality such as allowing users to manage their category subscriptions, or viewing and searching past conversations.

The back-end consists of a distributed network of servers that handle the delivery of instant messages, host the category namespace and user subscriptions, and route questions to invitees based on the categories that match the content. It also provides the necessary conferencing capabilities as discussed in the goals. The back-end is composed of several distinct modules. We discuss the design of this modular architecture in Chapter 3. In practice, we use several existing components (for instance, the IM platform) and integrate them to achieve our goal, rather than build everything from scratch.

1.4 Contributions

Our main contribution is the paradigm of interest-based real-time group communication where, apart from the intent (*viz.* the question), the person who seeks an answer requires no prior knowledge of the possible sources of information. We design a framework that automatically routes the person's question to subscribers with matching interests, abstracting away the need to know where to ask or go look for an answer.

The concept of Freebrain for real-time communication is very powerful. We believe it has a practical application given the popularity of instant-messaging and the desire of people to offer help. Our system brings the seekers one step closer to the sources of knowledge. In fact, our system allows non-humans such as on-line services to easily participate and offer their knowledge by simply subscribing to their related interests. In the past, the task of finding the right source (be it human or machine) of information has often been a problem. Freebrain tries to bridge that gap by providing an interesting, new IM-based medium for seeking real-time answers from any source of information.

People who go to conferences are the ones who shouldn't.
– Ozman's Law

Chapter 2

Related Work

In this chapter we discuss several other systems that provide group communication. We restrict our discussion to three systems that are in wide use by the general Internet public or members of the MIT community, namely, Internet Relay Chat, Newsgroups, and MIT's Zephyr.

2.1 Internet Relay Chat (IRC)

IRC is a protocol designed over a number of years (beginning in 1989) for use with text-based conferencing. To facilitate group conferencing, IRC uses channels. A channel is a named group of one or more users who will all receive messages addressed to that channel. The IRC protocol is based on a client-server model. Servers host channels, provide the necessary message multiplexing, and manage channels by keeping track of the channel members. Multiple servers interconnect with each other in a spanning tree architecture. Further details of the IRC architecture are described in RFC 2810 and other RFCs referenced from it [14].

2.1.1 Pros

IRC provides a virtual meeting place where people can meet and talk. Group discussions can occur at varying levels of interest and depth on one of the many thousands

of IRC channels. People can also talk in private to family or friends.

2.1.2 Cons

Technical Disadvantages —

Scalability – IRC requires each server to have a copy of the global state information; all servers must know about all other servers, clients, and channels, and information regarding them must be updated as soon as it changes. This is a serious handicap, which limits the maximum size an IRC network can reach.

Reliability – The only network configuration allowed for IRC servers is that of a spanning tree, so that each link between two servers is a serious point of failure.

Network Congestion – A problem related to the scalability and reliability issues, as well as the spanning tree architecture, is that the protocol and architecture for IRC are extremely vulnerable to network congestion, especially during link failure between two servers: if congestion and high traffic volume cause a link between two servers to fail, not only does this failure generate more network traffic, but the reconnection (eventually elsewhere) of two servers also generates more traffic.

Privacy – Since servers need to know all information about all other entities (both local and non-local users), privacy is also a concern.

Social Disadvantages —

IRC has become a way for people to "hang out" with others on the Internet. Getting help usually takes a back-seat unless one joins a specific channel for help started by some company or organization wishing to provide chat-based support. However, a large number of special channels makes it harder to reach everyone who might have some related information to share.

Another disadvantage is that if a user wishes to seek information, he/she must first find a channel that already has online users. The search process is often non-intuitive, because it is hard to match interests to channel names. Also, it can be hard to grab attention in a busy channel.

There is also very little personalization in common channels. Discussion in public is easy, but it is much harder to form a coterie. Of course, one can set up his/her own new channel and then hope that interested people join it, but the difficulty of finding this new channel creates a barrier to entry. There is no automated mechanism to get invitations for new channels. Thus, it is usually the case that users just join existing channels that they know have lots of people.

Last but not least, the busiest channels always have some people who, over course of time, begin to assume authoritative command over any ongoing discussion. Channels can have moderators, who can decide to kick out or ban specific users. Moderation is often useful to get rid of spammers, but it can turn into a power struggle.

2.1.3 Summary of Issues

In summary, there are various technical and social issues with IRC. Some of the technical issues include ineffective advertisement of a channel's existence, difficulty in finding interesting channels and the number of people each channel has, and difficulty in controlling content (moderation *vs.* spam). The social issues for help-based channels include difficulty in getting an optimal number of participants—too many leads to noise, while too few means very low chance of getting help.

The social disadvantages of IRC are shared by group chat services provided by proprietary IM networks such as MSN or AOL, which are, in essence, functional replicas of the older IRC system.

2.2 Newsgroups

The help page of "Google Groups" [11] has a short history of newsgroups. To get an idea about newsgroups, we quote a short paragraph from there — "Before the Web and web browsers, and before email became ubiquitous, online communication meant posting text messages on electronic bulletin boards where others could read and reply to them. Usenet began as a collection of these bulletin boards (now called discussion forums or newsgroups) started in 1979... Over the years, the number of such newsgroups has grown to the thousands, hosted all over the world and covering every conceivable topic about which humans converse."

Each newsgroup contains threads made up of messages (also referred to as "articles" or "postings") that look like e-mail between one user and another, but can be read by anyone accessing that particular newsgroup.

2.2.1 Pros

A newsgroup is identified by its name, which can be thought of as a category. There are more than 70,000 newsgroups (*i.e.,* categories) to choose from for participating in email-like discussions. The categorization is hierarchical, and there can be an arbitrary number of levels. Most widely used newsgroups are at level 3 or 4 (*e.g.,* `comp.os.linux.redhat`). This rich hierarchy simplifies the selection of a newsgroup based on very specific interests.

Most newsgroups maintain a list of frequently asked (answered) questions called FAQs, which can contain a wealth of information for newcomers. FAQs are periodically updated with the most commonly asked questions and answers seen on those particular newsgroups.

Searchable newsgroup archives are maintained on Google and other search engines for easy access to all the messages. This is a very useful feature, and often many answers can be found by simply searching these archives.

2.2.2 Cons

The most important disadvantage of newsgroups is that they do not provide instant communication. Newsgroup postings have a high latency because they are characterized by long messages that readers read and respond to like e-mail.

Another disadvantage is that it is very easy to spam newsgroups and hard to filter it. The problem of spam is greater in newsgroups than in IRC, since IRC moderators can ban the spammers. Although there are moderated newsgroups, the vast majority are not, making it relatively easy for spammers to fill them up with messages that none of the subscribers care about.

While there is incredibly valuable information available in the discussions taking place on newsgroups, finding that information can be an exercise in frustration and futility despite searchable archives available through Google. Searching takes time, and it is often the case that one might find related information, but not quite a solution to the exact problem one started the search with. In such cases, having a form of instant communication is highly desired.

Creating new newsgroups can be a very lengthy process, and requests can be rejected. Generally, a newsgroup creation request is made via a special message that asks news server administrators everywhere to create the group locally on their servers. Specific rules for requesting the creation of newsgroups are available online [23].

2.2.3 Summary of Issues

The biggest disadvantage of newsgroups is the lack of real-time communication. Newsgroups are good for asking questions that do not require an immediate response or that are characterized by long messages requiring careful thought and analysis. Other issues with newsgroups include excessive spam, and a very lengthy process for creating a new category.

2.3 Zephyr

Zephyr [30] is Athena's [21] text-based communication system for sending instant "zephyrgrams" (short text messages) among users. Zephyr also provides a centralized means for finding users' locations while they are logged in, and it allows fast message service (on the order of a few seconds) from a single sender to multiple recipients.

2.3.1 Pros

Apart from instant messaging, Zephyr's benefit is in its simple publish and subscribe system. Users may elect to subscribe to specific *classes*, which allow them to receive any messages that are sent to those classes. *Classes* are similar to IRC channels. A secondary level of classification called *instances* is also provided. Users may subscribe or send messages (publish) to specific instances within a class. A *class* and an *instance* together provide a two-level hierarchical naming system.

2.3.2 Cons

Zephyr's use of Kerberos [19] for user authentication severely limits its deployment to a few select colleges and organizations that currently use Kerberos authentication. A world-wide deployment is harder, since it involves setting up both Zephyr servers as well as Kerberos servers. In addition, Zephyr lacks a purely distributed architecture; servers must stay in contact with one another and provide a reliable backup system (via duplication) in the event of network failures. Thus every server replicates the entire subscription database.

Zephyr was designed for small communities, so no filtering or anti-spam mechanisms are built into the servers. It is relatively easy to flood or spam a given *class* or *instance.*

While Zephyr facilitates group communication, it makes it hard for the participants to know who else is listening to messages sent to a particular class or instance. This gives rise to "lurkers" who listen to all messages but do not participate in the

conversation. Thus, Zephyr lacks the notion of a group conference where the participants are visible to each other.

Another drawback is that it does not provide off-line messaging. Messages sent to offline users are discarded.

Also, the level of classification is limited by two keywords, one word indicating a *class* name, and the other indicating a particular *instance* within that class. While sufficient for small communities, this granularity is very coarse (compared to newsgroups), which makes it hard to cater to the varying topics of interest on the wide Internet.

Last but not least, Zephyr requires that the person sending a message know the exact name of the *class* and *instance*. This works well as long as the number of classes is small. For a richer categorization, the size is simply too large to expect an end-user to scan and select categories. An automated mechanism for category selection is much desired, as envisioned in the Freebrain system.

2.3.3 Summary of Issues

In spite of the similarities that Zephyr has with Freebrain, it has several drawbacks. Zephyr lacks important functionality such as group conferencing with visibility of participants. More importantly, Zephyr does not provide for an automated category (class/instance) selection. While this does not pose a problem within small communities, it will have the same issues as IRC if it is deployed on the wide Internet. Other issues include lack of off-line messaging and secure one-to-one messaging.

2.4 Other related work

Various search websites exist on the Internet like AskJeeves.com [2], that provide a web-based interface for asking a question and return results by searching a large collection of resources. AskJeeves also has a web-based forum where someone can can post a question and others can post answers, similar to newsgroups. The main disadvantage with such websites is the lack of real-time communication as in newsgroups.

Adde parvum parvo magnus acervus erit.
(Add little to little and there will be a big pile.)
– OVID

Chapter 3

Design

3.1 Introduction

In this section we discuss in detail the design of Freebrain, the accompanying research issues and challenges, the rationale that went into certain design choices, and advantages and disadvantages of the overall architecture.

3.2 Design Challenges

Designing a system for instant group communication on the wide area Internet presents several challenges:

Category Namespace — Since the goal of the system is to allow any user to ask any question, and to receive an answer, routing questions effectively requires a reasonably-sized namespace for the categories. With too large a namespace, the probability of any two users subscribing to the same category is close to zero, thereby defeating the primary goal of Freebrain, *viz.* bringing together people with common interests. With too small a namespace, users may be inundated with many questions that may not be of high interest to them. Thus, creating an appropriate namespace for message routing is very important to the success of Freebrain. We have chosen a public and open namespace provided by the

Open Directory Project (ODP) [24] used by some of the most popular web search engines including Google [10], Netscape Search, AOL Search, Lycos, and HotBot. The benefit of using the categorization of ODP as opposed to creating a new one is that it is updated periodically and is widely used in search engines. Easy access to the data from ODP was also an important factor that won over alternative choices such as using the newsgroups categorization.

Category selection — Given an appropriate namespace, the next challenge is matching a user's question with a set of matching categories. We do not directly tackle this challenge of natural language processing in the research for this thesis. Instead, we focus on a modular design so that our naive category selection module can be easily replaced with a smarter one. The algorithm currently used for category selection (using the full-text indexing capabilities of MySQL [22]) is sufficient and fairly powerful. We describe the details in Section 4.2.1.

Load balancing and fault tolerance – Centralized *vs.* Decentralized — In principle, load balancing and fault tolerance can be achieved by either a centralized architecture or a decentralized architecture. In practice, a centralized architecture only makes sense if the system is supported by a well-funded organization that can maintain a large set of machines in order to achieve the necessary load balancing and fault tolerance. A decentralized approach shifts control from a central authority to a wider audience. To facilitate open exchange of information, it is generally considered best to avoid control by a central authority that can dictate the flow of information [4]. Achieving load balancing and fault tolerance in a decentralized system is considerably more difficult and challenging than in a centralized approach that uses a server farm or a large cluster of machines [12]. For that reason, a decentralized system poses more interesting research challenges. The two approaches present an interesting trade-off between cost of deployment and performance. While a centralized architecture (*e.g.,* all the popular web search engines) requires high set-up cost, it also assures very high performance. A decentralized approach, on the other

hand, requires practically no set-up cost, but may come with a hit on performance. We chose the decentralized approach for the following reasons:

- Freebrain must not be controlled by a central authority.
- We wish to avoid a central point of failure that can potentially bring down the entire Freebrain network.
- It should be easy to grow the Freebrain network by allowing users to participate as in a peer-to-peer network like Gnutella [9]. A larger network also means the system can effectively cater to more users, since each new participant brings in more processor power and storage space.

Network bandwidth – Achieving efficient message delivery — With potentially thousands of users, it is essential to have efficient message delivery for invitations. The simplest option of sending unicast invitations to each invitee has clear limitations on performance and bandwidth. A multicast algorithm that takes into account geographic distribution and network latencies of invitees would be the best in terms of performance and bandwidth [20], but it would be overly complex for the purpose of an initial demonstration of Freebrain. The principle we follow is to use a simpler and relatively sufficient multicast algorithm that can easily be replaced by a more complex one, if the need arises. We discuss our approach in Section 3.5.5.

Scale – Achieving Internet-wide group communication — As we discussed in Section 1.2, one of the goals of Freebrain is to allow Internet-wide instant messaging to the seekers and sources of information. We expect Freebrain to serve anyone in the world. The challenge is to maintain a single global Freebrain network that allows anyone to join it as a client (user) or a server (node) and contribute "knowledge" or "resources" to the system. Choosing a decentralized and distributed approach allows our system to be highly scalable. Our system achieves scalability by using Chord [29] for distributing subscriptions, and by using Jabber [15], a distributed and scalable, instant-messaging platform. We discuss these components in Sections 3.3.3 and 3.5.3.

3.3 Design Choices

In this section, we elaborate on some of the choices we had among existing instant messaging or group communication systems, the distinguishing characteristics of Freebrain over those systems, and the criteria we used to select Jabber [15] as the instant-messaging module in our design.

3.3.1 Improvement over IRC

The idea of instant group communication was first made popular by IRC. Hence, it may appear that we can use IRC as a substrate to add the new Freebrain functionality. However, as discussed earlier in Section 2.1, IRC has various technical pitfalls, and thus we chose not to modify IRC to incorporate Freebrain concepts. Our goal was to provide a scalable network, which IRC fails to achieve.

In terms of functionality, Freebrain improves on IRC by providing personalized group communication. In IRC, the seeker has to join an existing chat room whose participants may be already engaged in conversation, and hence, often has to find ways to draw attention to his query. In contrast, a Freebrain conference (similar to an IRC channel) is created dynamically based on the inviter's question, and invitations are sent out to join that conference. The inviter can easily be given control (moderator status) since the conference is created dynamically at his request and others join it.

3.3.2 IM Substrate

Since we ruled out IRC, we explored the next alternative, instant messaging (IM) networks. We favored an IM platform as a substrate since it is a more popular and prevalent environment for messaging today (*e.g.,* Yahoo! Messenger, AOL Instant Messenger, and ICQ). However, we could not use any proprietary IM network, since we needed to add new functionality.

Jabber [15] is an open-source IM platform. It became our system of choice since it did not have the shortcomings of an IRC network, but still provided instant group

communication. Apart from being open-source, Jabber provides features (described in Section 3.3.3) that make it easier to implement our ideas.

Although Zephyr (which is also an IM system) is a potential substrate for Freebrain, it has several deficiencies. In particular, it lacks several features that are already provided (or easily implementable) in Jabber. Some of these features include off-line messaging, filtering, spam control, secure SSL-based messaging, PGP-based encryption, and non-textual data transfer.

In terms of functionality, Freebrain can be considered an extension of Zephyr that pays attention to automated messaging without requiring manual specification of class and instance names. Freebrain is also an extension of newsgroups that offers real-time messaging while still providing the benefits of interest-based discussions.

3.3.3 Jabber Features

Jabber is an open, XML-based protocol for instant messaging, which is managed by the Jabber Software Foundation [15]. An open-source server implementation of the protocol (called jabberd [16]) is available as are a number of clients for practically all OS platforms. Jabber servers run independently in a decentralized and distributed manner and provide robust, XML-based message routing between each other similar to the email system. Some of the important features of Jabber are:

- Users are associated with the particular Jabber server they register with and log in to. This provides a mechanism for a globally unique Jabber ID, as in the email system (*viz.* username@hostname).

- In order to send an instant message to another Jabber user, the sender simply constructs an XML message that contains the Jabber ID of the receiver and the sender, the message type[1], a subject tag, and a body tag. A sample message may look like:

[1]The "type" attribute allows clients to determine how to display a message. For instance, a "chat" type indicates that multiple messages of this type should be displayed in the same window, while a "normal" type indicates a new window should be created for each message.

```
<message type='normal' to='kalpak@localhost' from='me@localhost'>
  <subject>test</subject>
  <body>hi there</body>
</message>
```

- Users can start a group conference to chat with other users by inviting them manually. Note that this requires the inviter to know the Jabber ID for each invitee and to send each invitee a message.

- Jabber provides the ability to add special server-side components or "proxy" modules called *transports* to a server. Messages specially addressed to a transport get routed by the Jabber server to the specified transport. The transport can interpret the message and respond accordingly. The main motivation for transports was for adding interoperability between Jabber and the various proprietary IM systems. For instance, many on-line Jabber servers have transports for AIM, Yahoo! Messenger, and MSN. This allows Jabber users to communicate with people on proprietary networks using the same Jabber client. The "transport" feature of Jabber is used in Freebrain to proxy messages from Jabber clients (end-users) to the Freebrain network. Further details of the Freebrain transport are described in Section 3.5.5.

- Jabber provides the ability to log messages on the server in a given group conference. The log can be made available on the web similar to newsgroup postings, or IRC chat logs.

- Unlike proprietary IM systems, Jabber's protocol is well documented and provides for extensibility through its use of XML for all messaging. Moreover, its modular architecture allows developers to easily add voice interfaces, automated agents, filtering mechanisms, *etc.*

3.4 Freebrain Features

This Section describes the set of features of Freebrain. The focus here is on desired functionality. The design details of how we provide these features are in Section 3.5. Specific implementation details are deferred to Chapter 4.

Dynamic conferencing — with dynamic conferencing, a user is not required to have *a priori* knowledge of the intended recipients (or classes and channels as in Zephyr and IRC) in order to ask a question.

Category subscription — we allow users offering help to subscribe to their interests (in Freebrain terminology: *category* list). This forms the basis for dynamically routing a question to a matching target audience.

Category namespace — we allow a rich and open categorization to which users can subscribe. Rather than having all categories created by end-users (a system that can easily be abused), we use the ODP categories to achieve a common namespace. There are currently over 426,000 categories, all organized in a hierarchical structure. We call the categorization open, since the ODP is an open project and interested people can volunteer and participate in the maintenance of the category list. Members collaborate to add new categories. Since creating new categories in the ODP may be cumbersome, end-users are allowed to temporarily create new categories within the Freebrain system. Two or more users can mutually agree on a unique category name and subscribe to that category, thereby modifying the category namespace[2]. This feature is similar to that provided in Zephyr.

Sending invitations — the Freebrain network allows a user to invite people or send messages to databases based on interests (*categories*) in real-time. Invitations are sent out based on categories that match the content of the inviter's question,

[2]While adding and subscribing to new, non-ODP categories is possible, the current implementation does not support routing questions based on these categories (but it can easily be added). This will allow Freebrain to provide functionality similar to Zephyr—allowing communities to directly exchange messages, by-passing the automatic category selection.

thereby providing a high probability of bringing together those with common interests. Details of the automated category selection are in Section 4.2.1.

Instant group communication — people that receive invitations can elect to have a group discussion with the inviter by entering a conference (chat) room, which is dynamically created for that particular question. People can also choose to simply ignore the invitations. Invitees that are on-line databases may either elect to join the conference or directly respond to the inviter. We also provide a feature that allows non-invitees to join a particular discussion of interest. This is achieved by allowing users to browse the list of on-going discussions via a web-based interface.

Further desired features that would add more value to Freebrain include a searchable history of conversations similar to newsgroup postings or mailing lists, integrated web searches, and location-based routing to restrict messaging to smaller domains. Another useful feature would be a user rating system that allows an inviter to gauge the quality of an invitee. These features are detailed further in Section 5.4 discussing future work.

3.5 Freebrain Architecture

So far, we have discussed some of the choices we made in designing Freebrain. We now discuss the architecture and rationale, as well as the functionality of each module.

3.5.1 Overview

The Freebrain architecture consists of the following modules (see Figure 3-1):

- **Lookup primitive (Chord Key Router)** — provides the basic framework to create a self-configuring, peer-to-peer network of Freebrain nodes.
- **Freebrain Server** — maintains category listings and user subscriptions. Multiple servers collaborate to maintain subscriptions in a distributed and fault

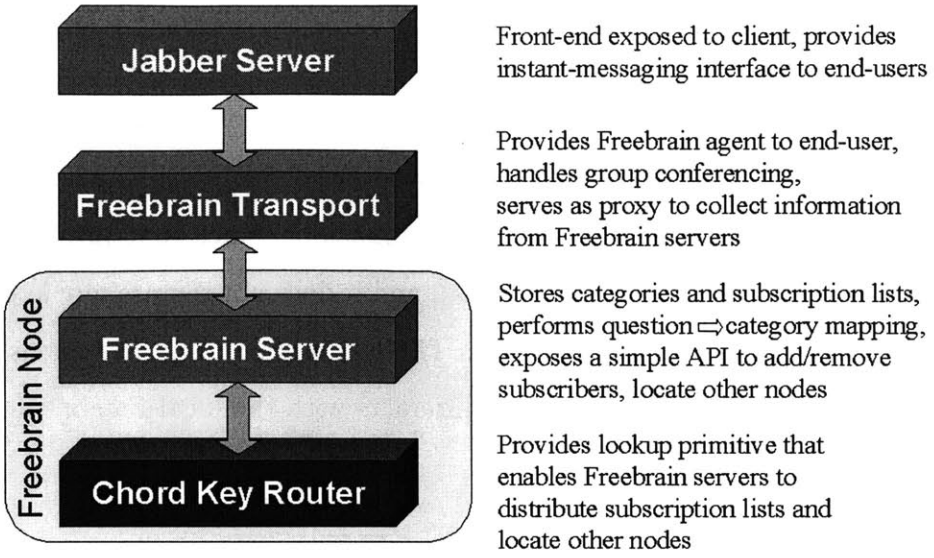

Figure 3-1: Freebrain Architecture - Module Functions

tolerant manner using the lookup primitive, Chord.

- **Jabber Server** — provides the instant-messaging service accessible to end-users. Multiple servers can exchange messages to provide a distributed and decentralized instant-messaging platform. Jabber is an existing IM platform; its features are described in Section 3.3.3.

- **Freebrain Transport** — module within each Jabber server that acts as a proxy to access the functions of a Freebrain server from within Jabber. The transport receives a user's question, uses the Freebrain network to determine the invitees, and sends out invitations. It also provides the necessary group conferencing capabilities.

In addition to the aforementioned server-side modules, a **Freebrain/Jabber Client** provides a GUI interface for an end-user to ask questions, receive invitations, and initiate conversations to exchange answers. The client can be any regular Jabber IM client with some added functionality for Freebrain's Q&A mode of messaging.

Thus, Freebrain has a layered and modular architecture. Every machine that wishes to participate as a server in the Freebrain network must run at least the lower two server-side modules, namely, the Freebrain server and the Chord key router. The Freebrain server exposes its functionality via TCP sockets so that a Freebrain transport can access it from another machine. However, typically we expect participants to run all four modules together on the same machine, since that provides the best performance (in terms of network latencies), and it does not require any knowledge of the remote Freebrain server's IP address. The Freebrain transport runs as a plug-in module within the Jabber server, and it interacts with the local Freebrain server via local TCP (or UNIX) sockets.

3.5.2 Layering Rationale

The 4-layer modular architecture achieves maximum decoupling among the modules. The order is shown top-down from a client's point of view. The Jabber server for IM is the top-most module in our design. Clients need only connect to a Jabber server in order to use the Freebrain services. All other components are abstracted away from the user. Jabber's own modular architecture provides us with several benefits. A Jabber server is easily extensible via its use of server-side components such as *transports*. As a result, adding a Freebrain transport module was easy given our choice of Jabber. An alternative design would have the Freebrain transport and (Freebrain) server merged into one module. However we decided against it, since the chosen design provides maximum decoupling between the front-end messaging service and the Freebrain features for categories and subscriptions. First and most importantly, if a Freebrain server crashes, it does not affect the Jabber server, which can continue to provide regular instant-messaging service to its users. Secondly, the design allows us to be Jabber agnostic. While Jabber fits the bill with its features and its extensible protocol and architecture, if a better IM platform emerges, we can easily switch to that because of the decoupling.

Thus far, we have reasoned about the top three modules, *i.e.,* the Jabber server, the Freebrain transport, and the Freebrain server. One could argue that the lookup

primitive module should really just be part of the Freebrain server. However, it is intentionally kept separate for the same reasons, *i.e.,* to maintain maximum decoupling from the various off-the-shelf components. Chord is used for the lookup primitive since it exposes a very simple API and makes theoretically sound guarantees about fair distribution [29]. In addition since it is being developed at MIT within LCS's NMS and PDOS groups, it was easy to incorporate. Since various other systems use Chord as a lookup primitive, it runs as an independent service daemon. A Freebrain server accesses the Chord service via a small network-based gateway program. Another benefit of abstracting away the Chord lookup functionality from the Freebrain server via a network interface is the ability to replace Chord with an alternative lookup algorithm without requiring any changes in the server.

Having discussed the layering strategy, we now delve into details of the individual functions of each module. We begin with the bottom-most module and move up the module stack since each higher module depends on functionality of its lower one.

3.5.3 Chord Key Router

This module provides the basis for achieving a distributed network of Freebrain servers. Chord is a peer-to-peer lookup algorithm developed at MIT LCS. It solves the problem of locating a data item in a collection of distributed nodes. In particular, it provides a primitive such that given the "key" of a piece of data (a 160-bit hash of the data), it returns the IP address of the node responsible for that "key". We use a prototype version of the Chord algorithm's implementation developed in the PDOS group. A Chord client (or equivalently, server, since this is a flat peer-to-peer network) runs as a daemon on a given machine and can join an existing network of Chord peers using a simple bootstrapping mechanism. A Chord client provides a simple yet powerful API that allows us to build peer-to-peer applications. Freebrain specifically uses two methods from the API, `getNode` and `getBackupNodes`.

The specifications for the two methods are as follows:

- **getNode(key)** – Given a valid `key`, queries the Chord network and returns the

IP address and port of the node that is responsible for that `key`.

- **getBackupNodes(key, address:port, maxnum)** – Given a valid `key` and the IP `address` and `port` of its responsible Chord node, returns the IP addresses and ports of `maxnum` number of backup nodes.

Using these two methods, Freebrain servers can achieve the goal for decentralized distribution of subscription lists. Chord makes several desirable guarantees that we can leverage upon:

- It is completely decentralized and symmetric, and can find data using only $\log N$ messages, where N is the number of nodes in the system.
- Chord's lookup mechanism is robust in the face of frequent node failures and re-joins.

In our design, the keys are based on the names of categories. The way a Freebrain server uses the Chord Key Router API is detailed in the next section.

3.5.4 Freebrain Server

This module is the heart of the Freebrain concept, **interest-based** group communication. An individual Freebrain server maintains subscription lists of several categories, which can be dynamically updated through a simple network-based API. The server uses a local database to permanently store valid lists of categories and maintains a soft-state list of subscribers for a subset of the categories. It exposes a simple API via TCP sockets with the following functionality:

Functions accessible to other Freebrain servers (or transports) —

- `SUBSCRIBE(catname, username)` – adds `username` to user list of category `catname`.
- `UNSUBSCRIBE(catname, username)` – removes `username` from user list of category `catname`.

- `GETUSERS(catname)` – returns all `usernames` from user list of category `catname`.

Functions accessible only to local Freebrain transport —

- `LOCALSUBSCRIBE(catname, username)` – calls `getNode(catname)`[3] to find responsible server and sends SUBSCRIBE(catname, username) to it[4].
- `LOCALUNSUBSCRIBE(catname, username)` – calls `getNode(catname)` to find responsible server and sends UNSUBSCRIBE(catname, username) to it.
- `LOOKUP(catname)` – calls `getNode(catname)` in local Chord key router and returns `<host, port>` of the responsible server as the result.
- `QUERY(question)` – first performs lookup of words in `question` in local category database to find matching categories, and then for each category, determines responsible server using `getNode`. This function returns a list of `<catname, host, port>`.

Using this API, Freebrain servers can collaborate to maintain subscribers of categories in a distributed manner. For instance, in order to subscribe a user to a particular category, a Freebrain transport need only send a `LOCALSUBSCRIBE` message to its local Freebrain server. Where the subscription is actually stored is abstracted away from the transport. If necessary, the transport can verify that the subscription was stored by using `LOOKUP(catname)` to find the responsible server and calling `GETUSERS(catname)` on it. Specific details on how we achieve fault tolerance are discussed in Section 5.3.1. The way a Freebrain transport uses the Freebrain server API is detailed further in the next section.

We note that in subsequent sections, we use the term *Freebrain node* to refer to a Freebrain server along with its Chord key router. We use the term *node* since the Freebrain servers essentially form a peer-to-peer network using a Chord substrate.

[3]This is actually the chord ID of `catname`. The function call for converting `catname` to a Chord ID is not shown here for brevity.

[4]Note that this does not take into account replication, which can be achieved using getBackupNodes. We discuss how to achieve replication in Section 5.3.1

3.5.5 Freebrain Transport

This module serves as the main integrator between the Freebrain nodes and Jabber servers. It combines vanilla instant-messaging with interest-based group communication.

The Freebrain transport provides the following functionality:

Group Chat/Conferencing — this provides the necessary functionality to host multi-user chat, aka a group chat or conference. We use a modified version of the "conference" server-side component that comes with the standard Jabber distribution. The original component allows users to create chat rooms and manually invite other users to it. Our modifications incorporate Freebrain functionality that allows automated invitation based on categories.

Automated invitations — when the Freebrain transport receives a question from an end-user, it performs the following steps:

- creates a new conference room for this question.
- sends an invitation to the inviter to join this new conference.
- calls `QUERY(question)` on its local Freebrain server, which returns the names of matching categories and their responsible Freebrain servers (see Figure 3-2).
- for each category, connects to the responsible Freebrain server, issues a `GETUSERS(catname)`, and gets back a list of subscribers (see Figure 3-3).
- groups subscribers that use the same Jabber server and, for each such group, forwards an invitation to the Freebrain transport running on that server containing the question, the list of categories, the list of subscribers and the name of the conference room. The categories are included to allow invitees to view them along with the question. The Freebrain transport that receives such a forwarding invitation sends invitations to each subscriber to join the newly created conference (see Figure 3-4). This mechanism is detailed further in Section 4.3.

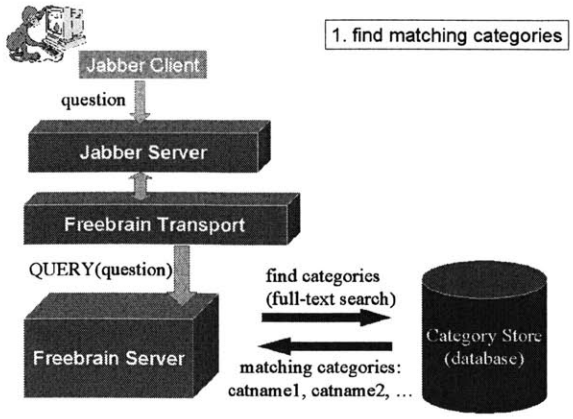

Figure 3-2: Automated invitation – Find matching categories

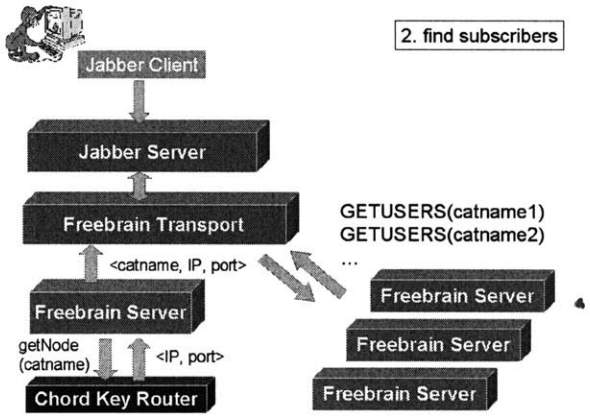

Figure 3-3: Automated invitation – Find subscribers

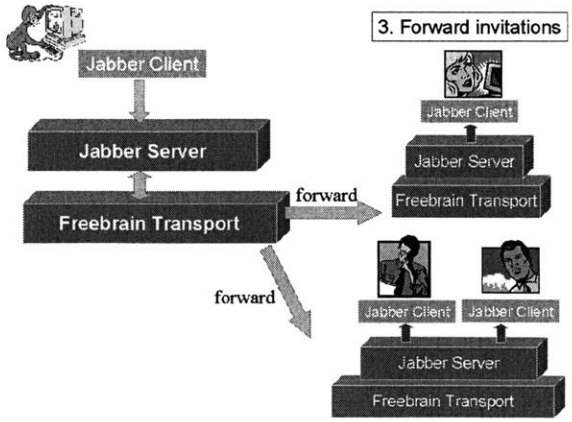

Figure 3-4: Automated invitation – Forward invitations

Access via a Freebrain user (agent) — one feature of Jabber is that any server-side component such as a transport can be made accessible to the end-user as a special Jabber ID where the hostname is a subdomain of the Jabber server. For instance, if we run a Jabber server on `jabber.net`, we can serve the Freebrain transport off the subdomain `freebrain.jabber.net` (the subdomain must be resolvable by DNS to an IP address.) Users can simply add to their contact list a Jabber ID of the form `freebrain@freebrain.jabber.net` and have access to the Freebrain network. We refer to such a Freebrain user as a *Freebrain agent.*

Access to add/remove user subscriptions — this allows users to subscribe to or unsubscribe from categories. The interface is quite simple and is used by a web-browser based client that the end-user can use to maintain subscriptions (see Section 4.4.2). The transport exposes a "subscribe" Jabber ID (say, `subscribe@freebrain.jabber.net`.) In order to subscribe to a category, we simply send an instant message with the subject "add" and the body containing the name of the category. Similarly, to unsubscribe we use the subject "remove". To retrieve current subscriptions, we send a message with the subject "get". The transport will return a message containing the names of subscribed categories.

Log invitations and conference messages for web-based access — this allows both Jabber and non-Jabber users to access message archives of the Freebrain network via a web-based interface. This includes invitations, conference messages, and the current conference status (open or closed). This both allows a Jabber user to join an active conference and also allows any user to search these message archives as they can newsgroup archives on Google Groups. In the current design, these messages are archived on a central server mainly for performance reasons. A distributed archival is certainly possible using the Chord key router; however, the access time to the archives for an end-user would be substantially slower since it would involve aggregating archives from multiple servers. We discuss the method we used for logging in Section 4.3.

3.5.6 Jabber Server

This module is the standard Jabber server distribution. It provides the XML-based IM platform. The important features of the Jabber server were already discussed in Section 3.3.3. The Jabber server is the only module that is directly visible to an end-user. The user requires a Jabber client to connect to the Jabber server. If the Jabber server has a Freebrain transport, she can use the Freebrain network for interest-based group communication by simply adding the *Freebrain agent* in her contact list. To ask a question, she simply sends an instant message to the *agent*.

3.6 System Interaction

So far we have described the functions of each module in our system, and the reader may already have a fair idea of how all the modules in the system interact with each other. This section elucidates the interaction of an end-user with Freebrain.

Create new Jabber account — Using a Jabber/Freebrain client, an end user (say, John) must first create an account on a Jabber server that has a Freebrain transport, say `jabber.net`.

Add Freebrain agent — After account creation, John can add the *Freebrain agent* to his contact list in his client. Jabber clients provide a feature to browse the entire list of transports available on the connected Jabber server. This allows John to find the Freebrain transport and add `freebrain@freebrain.jabber.net`.

Ask a question — When John sends an instant message (a question) to the *agent*, the corresponding Freebrain transport handles it as described in Section 3.5.5. John receives an invitation to join a newly created conference room. All online users who are connected to the Freebrain network via their respective agents and have matching interests (categories) receive an invitation containing the question. These invitees can passively ignore the invitations or take action such as joining the conference to discuss the question.

Join group conversation — Invitees that choose to join the conference room can have a group discussion with the inviter (John) and help answer his question. This is similar to the chat room environment in IRC, except the participants are only those who are interested in answering the question, unlike IRC.

3.7 Integration with databases

So far we have discussed the architecture of Freebrain and discussed the scenario in which a person asks a question and interested people join the conference to share their knowledge. Our architecture also enables machines to share their knowledge. We focus on databases that serve specific information, such as weather, email addresses, or calendar information. Given the Freebrain framework, a database can share its knowledge by adding a front-end proxy that can speak the Jabber protocol to send and receive instant messages. Specifically, the front-end will subscribe to particular categories that match the content of the database. For instance, a weather front-end can subscribe to any category that contains the word "weather" (*e.g., Regional/North America/United States/Weather* and all its sub-categories such as *.../By State/Massachusetts/Boston*). Whenever a person asks a question such as "What is the weather in Boston, Massachusetts?", the top-ranked matching category would be the one that contains the words "weather", "Boston" and "Massachusetts", *i.e., Regional/North America/United States/Weather/By State/ Massachusetts/Boston.* Since the weather database is subscribed to this category, its front-end will receive the question. Now the front-end can query the database and return the result directly to the inviter.

As an additional feature, which isn't part of the current design, if the message that arrives at a database cannot be mapped to an appropriate query, the front-end can return to the inviter an XML template that defines the fields it can understand. The inviter's client program can now dynamically generate a form based on this template, and the user can fill in the appropriate information. Since instant-messaging is done using XML, exchanging structured information is fairly straightforward.

In retrospect, another advantage of the Freebrain architecture is that it allows scalable dissemination of automated information from databases to people. In this case, everyone interested in getting a periodic update about the weather in Boston can simply subscribe to the appropriate category. The weather database, in turn, periodically publishes the weather information (by sending an instant message to its *freebrain agent*), and the information gets appropriately routed to all the subscribers by the Freebrain network. This provides a very powerful, yet simplified push mechanism for routing information to subscribers using instant messaging.

3.8 Summary

In this chapter we have seen the modular, layered architecture of Freebrain, the functionality of each module, and how the modules interact with one another to provide a distributed and scalable framework for interest-based instant-messaging. We also discussed and dealt with the following research issues:

Naming — We selected the public domain and open ODP categorization as the naming system for categories.

Load balancing and scalability — We designed a distributed and decentralized architecture built upon Chord and Jabber.

Network bandwidth efficiency — We achieved efficiency by grouping invitations based on common Jabber servers and multiplexing at the edge of the network.

In the next chapter, we discuss the details of our specific implementation and focus on the end-user interface, *i.e.,* the Jabber/Freebrain IM client and the web-based interface for viewing categories and searching message archives.

The reward of a thing well done is to have done it.
– Emerson

Chapter 4

Implementation

4.1 Introduction

In this chapter we describe specific implementation details of the Freebrain server and transport, the two modules that constitute the bulk of our implementation. We further describe the clients that end-users can use to take advantage of Freebrain.

4.2 Freebrain Server

As described in Section 3.5.4, an individual Freebrain server maintains subscription lists of categories which can be updated via a network-based API. In our current implementation, a Freebrain server listens on TCP ports 12346 and 12347. Remote Freebrain servers or transports connect to port 12346 while the local transport connects to port 12347. Data exchange occurs via the simple text-based API described in the previous chapter. Our implementation uses a simple regular expression parser to detect the commands from the API. This implementation is sufficient for a working demonstration of Freebrain.

4.2.1 Category Selection

As described in Section 3.5.4, the Freebrain server uses a local database to permanently store valid lists of categories. Our implementation uses a MySQL database.

The current version of MySQL, 3.23.36, allows full-text search capabilities on specified columns in a given table. We use this feature to perform our category selection. Our database has a `categories` table with two main columns, `category` and `description`.

In order to populate this table, we use an automated script that downloads and parses the current RDF dump [25] of the ODP categorization from the ODP website[1]. We chose to populate our own database, as opposed to directly querying (using HTTP) the ODP website or the Google Directory website, so we can implement novel algorithms to find matching categories rather than depend on the results provided by these websites. For example, Google and ODP only return the top one or two matching results even if there may be several more related categories. In addition, directly querying the remote website would involve parsing its HTML response, which may have a different format if the web page layout is changed. Thus, we chose to populate our own local database and use the original RDF dump, which is in a consistent, structured XML format.

When the Freebrain server receives a `QUERY(question)` message, it performs a full-text search of `question` in the `categories` table and picks the n results with the highest matching scores. In the current implementation, we limit n to 5. Although a larger value of n may increase the target audience (since more categories will likely have more subscribers), it results in a noisier set of categories with lower matching scores and hence may cause the question to be forwarded to the wrong people. Some careful analysis of how many matching categories to select may be necessary to optimize quality over quantity, but we leave this as a task for future work. Moreover, the next version of MySQL will be equipped with more flexible full-text search capabilities, which can improve the quality of our selection.

[1]The RDF dump is updated on the ODP website once every two weeks.

4.2.2 Integration with Chord

As described in Section 3.5, the Freebrain server is layered on top of the Chord service. However, one issue arose in the implementation. The current distribution of Chord only exposes an RPC-based API, and it uses its own version of asynchronous I/O, RPC, and various other libraries that cause conflicts when compiling other programs that use standard C++ STL templates with Chord. As a result, a simple layering would require that the Freebrain server be built with the entire Chord distribution (due to the various library dependencies), and it would also prevent use of standard C++ libraries. This led us to add another module between the Chord service and the Freebrain server. From a software engineering standpoint, not only did this make implementation of the Freebrain server easier, it also gave us a cleaner design abstraction. The new module, a "Chord gateway", connects to the local Chord service using RPC and exposes the Chord API via TCP network sockets. It simply receives messages on the TCP socket, issues the corresponding Chord RPC call, and relays back the results.

4.3 Freebrain Transport

The implementation of the Freebrain transport was governed primarily by the Jabber API and the existing group conference support. Some particular implementation choices we made were:

Storing subscriptions on Jabber server — in Chapter 3, we discussed how a user's subscriptions are distributed and maintained by the Freebrain servers, with each server responsible for specific categories. In practice, we keep the user's subscriptions not only on the Freebrain servers but also his Jabber server. We do this for two reasons. First, if an off-line user remains permanently subscribed to his interests, the next time he logs in, he will be inundated with all invitations that accumulated while he was off-line (since Jabber supports off-line messaging). Since the primary role of Freebrain is to assist in real-time answers,

it is not desirable to send invitations to off-line users. Second, if we store subscriptions only on Freebrain servers, there is no way for a Freebrain transport to determine on which Freebrain servers a user's subscriptions are stored (short of querying all Freebrain servers) and, hence, to notify them about the status of the user (whether on-line or off-line).

For these two reasons, we implemented a two-stage process. Freebrain servers maintain subscriptions for on-line users as soft-state, and the Jabber server that a user connects to permanently stores his list of subscribed categories. The Freebrain transport automatically subscribes (by calling `LOCALSUBSCRIBE` on the local Freebrain server) the user to all his categories when he connects to Jabber and unsubscribes him when he disconnects. An alternative to permanently storing the list of categories on the Jabber server is to store it on the end-user's client. However, in the Jabber framework, all information (such as the user's contact list or his message filters) is stored on the server-side since clients are meant to be very light-weight and also to allow users to have access to the same information irrespective of which client they use. As a result, we use the same strategy and store category subscriptions on the Jabber server.

We note that since we have an invitation-logging mechanism, a user may always use the web-based interface to scan any invitations he may have missed while he was off-line. The two-stage process we just described simply prevents the user from receiving too many stale invitations everytime he logs in.

XML-based invitations — since instant messages in Jabber are exchanged using XML between clients and servers, we took advantage of this and designed our invitations so that the content is well-specified using XML tags. In particular, we specify the question in a `<question>` tag, the Jabber ID of the inviter in an `<inviter>` tag, and the categories that the question was matched against within a `<categories>` tag. Each category is specified as a nested `<category>` tag. A `<conference>` tag contains the identifier of the conference room that was created for this question. Invitees can join the conference using this identifier.

After an inviter sends a question to his Freebrain agent and the question arrives at the Freebrain transport, there are two further steps of messaging – first from the Freebrain transport to remote transports that host invitees, and next from those remote transports to the actual invitees. The message structure at each step is as follows:

- Between two Freebrain transports –

```
<message ...>
  <body>
    <question>question text</question>
    <inviter>inviter id</inviter>
    <categories>
      <category>catname1</category>
      <category>catname2</category>
      ...
    </categories>
    <users total=num_invitees>
      <user>user1</user>
      <user>user2</user>
      ...
    </users>
    <conference>confname</conference>
  </body>
</message>
```

  The additional <users> tag contains the Jabber IDs of the invitees that the remote transport must forward this invitation to. This tag has an extra attribute "total", which indicates the total number of invitees to which the invitation is being sent. This is the sum total of all invitees, not just the ones listed in the <user> tags. This information allows each invitee to learn how many others received the invitation, and can help them determine whether they should consider joining the conversation with the inviter.

- From a Freebrain transport to an invitee –

```
<message type='freebrain' ...>
  <body>
    <question>question text</question>
```

```
    <inviter>inviter id</inviter>
    <categories>
      <subscribed>
        <category>catname1</category>
        ...
      </subscribed>
      <unsubscribed>
        <category>catname2</category>
        ...
      </unsubscribed>
    </categories>
    <invitees>num_invitees</invitees>
    <conference>confname</conference>
  </body>
</message>
```

One difference in this message is that we do not include the `<users>` tag, and we substitute the value `num_invitees` in an `<invitees>` tag instead. More importantly, we separate the categories associated with the question in two subsets, those to which the invitee is subscribed and those to which he is not. We can do this because this message is forwarded to the invitee by the Freebrain transport that has access to his permanently stored subscriptions. The two subsets can help the invitee learn of categories that he could possibly subscribe to since they will likely be of similar interest. Also, we use a special value, "freebrain", for the `type` attribute of `message`. This allows a Jabber/Freebrain client to detect that this is a special Freebrain message (an invitation) and in turn specialize the GUI for such messages. We give details of the GUI in Section 4.4.1.

Logging questions and conversations — while instant messages are useful to get across to people in real-time, the system we described so far is ineffective without the most useful feature of newsgroups, message archival. We have implemented a simple message logging mechanism that allows anyone to browse the list of questions and conference (conversations) logs via a web-browser. The feature is implemented using Jogger [18], a Jabber-enabled web-log program. Jogger runs as a server-side component and logs any messages that are sent

to it (*e.g.*, if Jogger has the subdomain `jogger.jabber.net`, messages sent to `<anyuser>@jogger.jabber.net` get logged). Currently, all Freebrain servers are configured to send a copy of the questions and conference messages to one central Jogger service. Jogger maintains messages in a MySQL database, and the data is accessible via a PHP [26] script. While the current single Jogger service may pose a bottleneck problem, it is less of an issue since the purpose of it is archival and not real-time messaging. Moreover, Jabber has a built-in queuing mechanism, so bursts of messages can be easily handled. We discuss possible solutions for decentralizing this logging service in Section 5.4.

4.4 Freebrain Client

The Jabber/Freebrain client is the most important part of the system from an end-user's point of view. A user with a standard Jabber client can easily use the Freebrain network to ask or receive questions and join a conference. However, the number of questions received can quickly escalate as the number of Freebrain users increases, thereby making current forms of displaying messages (as individual pop-up windows or as raw text messages in a single chat-style window) fairly inadequate. One advantage of Freebrain's design is that interested developers can easily add specialized Freebrain functionality (*e.g.*, for display) to existing Jabber clients. We developed a small prototype extension to a commonly-used Linux-based GNOME Jabber client called Gabber [6].

We believe a Jabber/Freebrain client should have the following important functionality:

- **Conferencing** – it must have support for conferencing (group chat) similar to IRC clients. Good Jabber clients already have this functionality built in.

- **Invitations Viewer** – a simple window interface that can list all incoming invitations in a tabulated manner and allow the user to select an invitation, view the details, and join the conference room corresponding to that invitation.

Advanced features would include the ability to sort and group invitations based on categories, search through them, and delete them from the viewer.

4.4.1 Gabber

The wide availability of clients on multiple platforms is one of the key benefits of having selected Jabber. We selected Gabber as our Jabber client since it is a popular and open-source, Linux client. However, it is not cross-platform, and it will be necessary to modify other clients[2] so that Freebrain features can be widely available to users of all OS platforms, especially Windows and MacOS.

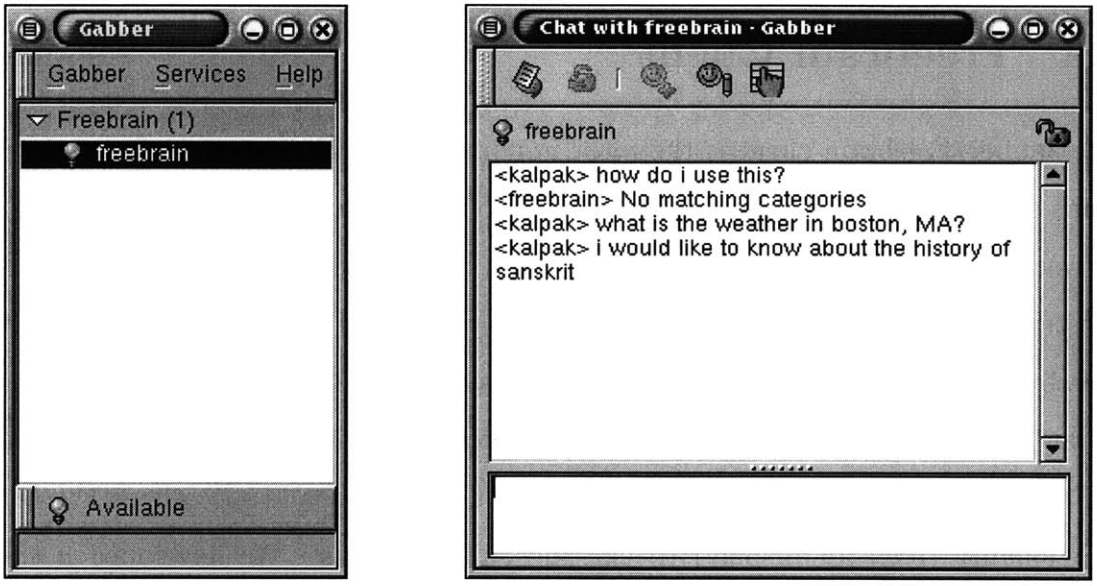

Figure 4-1: Gabber: (a) Main Window, (b) Chat with Freebrain

The basic look and feel of Gabber is shown in Figure 4-1. The first image shows that an end-user has added a Freebrain agent to his contact list. The second image shows that the interface for asking a question is simply the one provided in Gabber for sending an instant message. An enhanced interface for asking a question could allow a user to select from various options such as manually specifying a list of categories.

[2]We started to work with Psi [27], an open-source QT-based cross-platform Jabber client. However, Psi currently lacks group conferencing support. The author of Psi plans to add group conferencing support in the next release, and we plan to incorporate the Freebrain Invitations Viewer at that time.

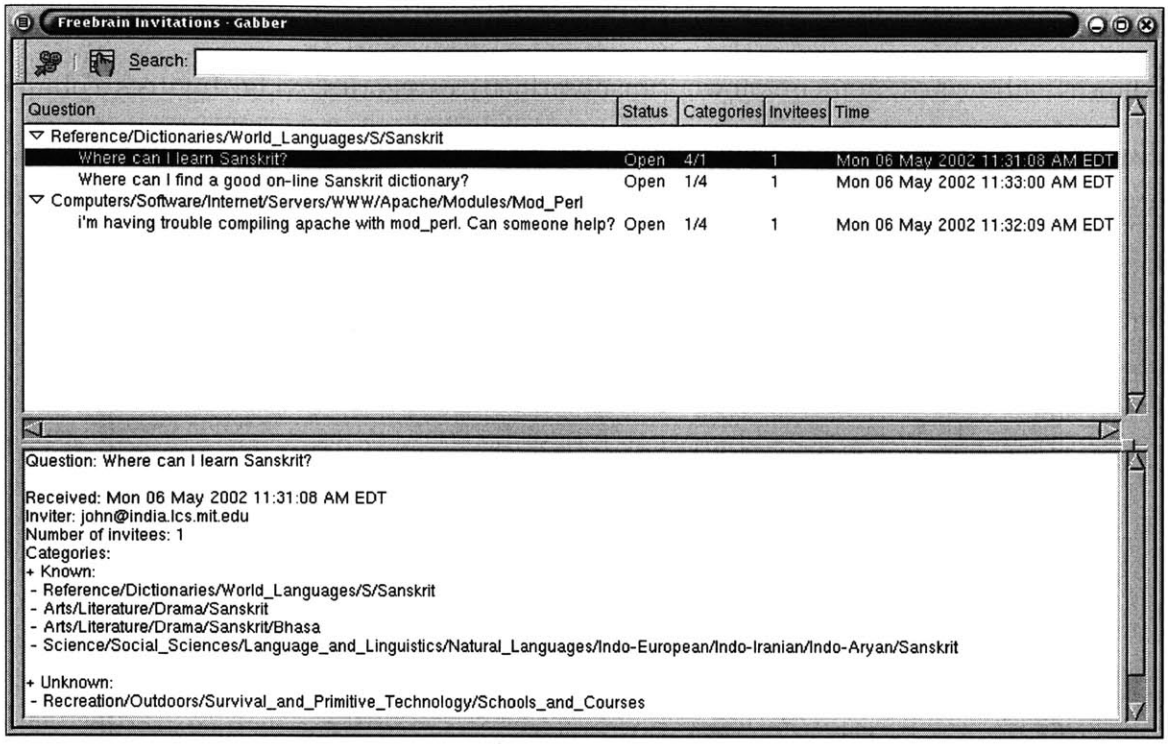

Figure 4-2: Gabber: Freebrain Invitations

However, for the purposes of a demonstration of the Freebrain architecture, the simple interface currently provided in Gabber is sufficient.

The only addition we made to Gabber was a new Freebrain Invitations viewer for displaying incoming messages of the type "freebrain". The viewer we have created is merely one possible interface. Other GUI developers are free to write their own specialized viewers. Our viewer is characterized by a table with 5 columns, as shown in the screenshot in Figure 4-2. The columns are Question, Status, Categories, Invitees, and Time. The contents of an incoming invitation are mapped as per the correspondence between the column name and the XML tag name. The questions are arranged in a two-level tree. We specify the first category from the `<subscribed>` tag as a parent in this tree. All invitations with the same first category are grouped under that parent. The purpose of the Status column is to indicate whether the conference room associated with an invitation is open or closed. This feature is easy to implement on the server-side since a conference room can simply send out a "closed" status message to all the invitees similar to the manner in which the initial invita-

tions were sent out. The Categories column indicates two numbers—the number of known (subscribed) and unknown (unsubscribed) categories. The Invitees column indicates the total number of invitees, and the Time column indicates the time at which the invitation was received. A text pane at the bottom provides further details of a selected invitation, such as the names of all the categories and the name of the inviter.

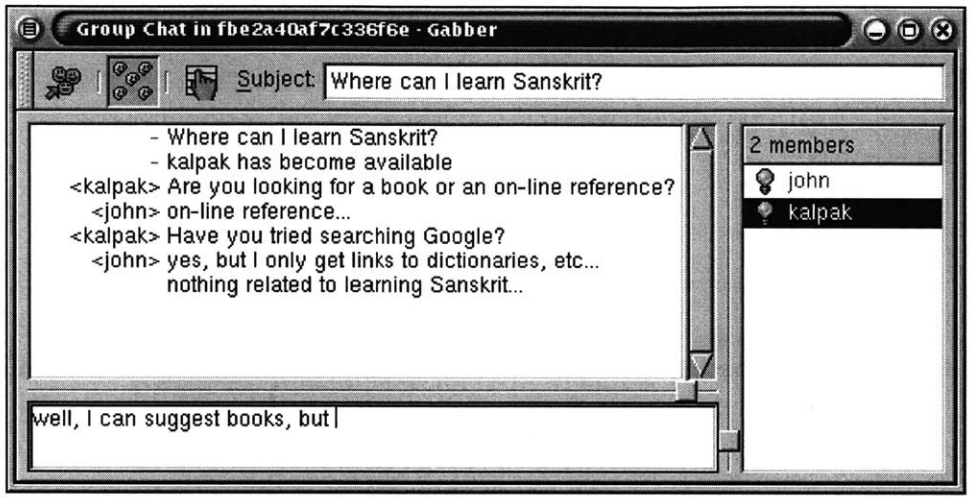

Figure 4-3: Gabber: Group Conversation

To join the conversation, *i.e.,* the conference room, associated with a selected invitation the user simply double-clicks on the selection or clicks on the "Join Conversation" button (upper-left icon), which initiates the standard Jabber conference join protocol and spawns off a new conference window. A sample subsequent group conversation is shown in Figure 4-3.

4.4.2 Web-based interface

While Gabber provides the real-time instant messaging environment, certain other important features are currently provided via a web-based interface, namely, the ability to navigate the current category structure and add or remove subscriptions to these categories. We have written a PHP script that allows a Jabber/Freebrain user to log in using his Jabber ID and password. It then presents the user with a list of the categories he is currently subscribed to, allowing him to unsubscribe from them. The

interface also allows him to search through all the available categories and subscribe to the ones he wants.

On the back-end, the PHP script connects to the user's Jabber server and speaks the Jabber XML-based protocol. Thus, adding or removing categories from the user's subscription is simply a matter of sending a message to the `subscribe` Jabber ID of the Freebrain transport (*e.g.,* `subscribe@freebrain.jabber.net`), as described in Section 3.5.5.

As discussed in Section 4.3, a modified version of Jogger that allows users to view or join conversations is also part of the web-based interface.

4.5 Summary

This chapter discussed the implementation details of the two main modules of the Freebrain architecture, the Freebrain server and the Freebrain transport. It introduced the various clients that end-users can use to access the features of Freebrain. In particular, it discussed modifications we made to a Jabber IM client to aid the display of invitations, and also described details of the web-based interface that allows easy access to Freebrain conversations, similar to web-based interfaces to newsgroups.

In the pursuit of learning, every day something is acquired.
In the pursuit of Tao, every day something is dropped.
—The Tao Te Ching

Chapter 5

Results and Discussion

5.1 Introduction

This chapter discusses issues related to the design and implementation of Freebrain, and it presents some preliminary results of our implementation. Finally, it makes several suggestions for future work.

5.2 Results

The implementation of Freebrain is dependent on several existing components, especially the Jabber IM platform and the Chord lookup service. While it is important to evaluate the performance of the entire system, we are aware that our initial implementation is not geared towards high performance. Rather than quantify specific details such as the number of messages each node can handle per second, or the number of subscriptions we can maintain on a given server, we evaluate our system by focusing on the main modules and analyzing their basic performance.

First, we note that the initial implementation achieved the key goal of Freebrain—creating a simple, intuitive interface for end-users to ask questions and have interest-based conversations. Freebrain abstracts away the process of finding an appropriate category (or channel, as in IRC), and not only brings together the benefits of instant responses from peers, but also allows existing on-line services to provide results, via

the same IM interface. Thus, Freebrain acts as a universal access point for asking questions and receiving answers in real-time.

5.2.1 Basic performance

Jabber Server — the relevant metric for evaluating the Jabber server is the number of concurrent users it can handle. This translates to the number of concurrent users that can send/receive messages using that server. The server is limited by the number of socket connections possible on the particular system and the GNU pth (Portable Threads) [8] (which the server uses for multi-threading). For instance, on a standard Linux machine the limit is 1024 simultaneous socket connections. There are currently efforts underway, in concert with the Open Source Development Lab, to improve the scalability of the Jabber server [17].

Freebrain Server — the initial implementation of the Freebrain server is a simple, single-threaded event loop that blocks on all socket I/O calls. The most resource-intensive API call of the server is `QUERY`, which involves a full-text lookup against a MySQL database of approximately 426,000 categories. This call blocks from anywhere between 1-4 seconds on an Intel Pentium-II 450MHz machine. Since this is primarily a disk-intensive task, the lookup time should not be too different on a faster CPU. However, since MySQL caches recent queries, subsequent similar queries are significantly faster. All other delays are accountable to network lag associated with locating and connecting to another Freebrain server for `(UN)SUBSCRIBE` calls. However, we are guaranteed by Chord that locating another node will take no more than $O(\log n)$ delay, where n is the number of nodes in the Freebrain network [29]. Since the server is single-threaded and it blocks on every socket call, other users must wait till it finishes servicing one entire request (be it `QUERY` or `(UN)SUBSCRIBE`).

Naturally, to increase performance perceived by users, we should move to a multi-threaded or asynchronous I/O architecture for the Freebrain server so we can handle simultaneous connections like the Jabber server.

Freebrain Transport — since the Freebrain transport runs as a server-side module within the Jabber server, it benefits from the multi-threaded architecture and can handle simultaneous connections. Thus the only limiting factor in its performance is the fact that it relies on the single-threaded Freebrain server.

Forwarding invitations — our system is quite efficient in forwarding invitations as it distributes the load towards the edge of the network. Thus, the number of invitations that a source Freebrain transport has to send out only grows as the size of the network (number of Jabber servers) and not as the size of the user base. The edge servers must deal with multiplexing the invitation to their local users, which they can handle since they already support those users.

Overall scalability and load-balancing — owing to the distributed nature of both the Jabber and Freebrain servers, we can achieve a highly scalable network as we add more servers. Jabber can presently handle up to 1024 simultaneous connections, and the multi-threaded release of the Freebrain server will be able to provide faster service to each user. Thus, with every new Jabber and Freebrain node we add, we can allow about 1000 new users to use the system (*i.e.,* ask questions and have conversations). We also achieve good load-balancing characteristics, since all Freebrain servers are equal. Since we use Chord, the distribution is fair among all participating servers. Each server is responsible for an equal number of categories. We do not think the actual number of subscribers in any given category is a concern, since memory is cheap and even 10Mb can accommodate approximately 350,000 entries (using an average Jabber ID size of 30 bytes), a number sufficiently large for most practical purposes.

5.2.2 Quality of service

Ideally, we should evaluate the quality of Freebrain service as perceived by end-users by collecting comments and evaluations from them after they have used Freebrain for some time. However since the system is yet to be polished for public release, we have not yet conducted such a study. Moreover, the success of Freebrain is directly

dependent on the size of the user base—too few users will not provide the true benefit of Freebrain. Developing a sizable user base is a matter of time as word spreads and more users start using Freebrain. Naturally, we want the first release of Freebrain to be good enough to attract users. The framework we have currently built will be ready for public use this summer, after we have integrated some of the suggestions for robustness and performance made in this chapter. The Freebrain/Jabber client will need special work since that is the only element in our system that end-users will interact with and will determine the popularity of Freebrain.

For now, we restrict our present discussion to components that we can evaluate independent of the user base:

Automatic category selection — the results from our simple full-text search of the flat ODP category database in MySQL are roughly equivalent to results obtained by running the same query on the Google Directory or the original ODP website [24]. In particular, most queries produce the same top-ranked matching category. Our simple search however is, at present, limited by the features of the current version of MySQL. For instance, MySQL can only index words that have 4 or more letters, rendering it incapable of a full-text search of important 3-letter words (for instance, LCS, SSL, *etc.*)[1]

In addition, the lower-ranked results often include categories that only match insignificant keywords from the query, producing some undesirable category selections. For instance, if the query is "Where can I learn Sanskrit?", the top 5 ranked matching categories are:

1. Reference/Dictionaries/World_Languages/S/Sanskrit
2. Arts/Literature/Drama/Sanskrit
3. Arts/Literature/Drama/Sanskrit/Bhasa

[1]The 4-letter limit can be changed, but since it requires re-compiling MySQL, it will be harder to deploy Freebrain servers on standard machines. Besides, since the next version of MySQL fixes this problem via a user configurable file, it is prudent to simply wait until standard Linux (or other OS) vendors ship with the newer version, which will happen within a month or two.

4. Science/Social_Sciences/Language_and_Linguistics/Natural_Languages/Indo-European/Indo-Iranian/Indo-Aryan/Sanskrit
5. Recreation/Outdoors/Survival_and_Primitive_Technology/Schools_and_Courses

Although the top 4 matching categories are related to Sanskrit, the last one is a noisy category, which shows up because its description (not shown here) has the word "learn". Based on these results, we realize that adding extra logic to the automatic category selection process is much desired.

As we noted in Section 4.2.1, the next version of MySQL has more powerful full-text search features, and so our system will produce better results. It is important to realize that a simple full-text search is still surprisingly useful even though the category database has 426,000 categories; we originally thought the amount of noisy categories would be very high, but it is evident that if questions have sufficient context, the top matching categories are quite accurate. Also, restricting the search to a narrower interest domain will certainly produce far more accurate results as it will reduce further the space for noisy categories.

The benefit of using the ODP data is that many users are familiar with this categorization, having used Google or other web search engines, and hence will be able to easily determine whether the matching category is appropriate or whether they should try rewording their question to add more context.

Unified IM interface — to demonstrate the true power of our architecture, *viz.* the ability to find not only people but also on-line databases, we created a sample automated weather report application (called *"weatherdb"*). We used a small Perl library [7] that can fetch weather information from weather.com and added a Jabber front-end client [3] that can connect to a Jabber server like a regular user. We subscribed *weatherdb* to all the categories concerned with weather, which accounts for about 4,000 categories in the ODP (every known city in the US as well as cities from several other major countries). We added some logic to the front-end so it can parse certain questions (messages) and return results

to the inviter. If it cannot understand the format of an incoming message, it returns a message to the inviter indicating the format that it does understand.

Using Gabber, we asked the following questions to Freebrain:

- What is the weather in Boston, MA?
- weather boston, ma
- weather 02141

When we ask any of the above questions, Freebrain picks the matching weather category, and in this case, since *weatherdb* is subscribed to all weather categories, it receives our question. We include a screenshot of a sample reply from *weatherdb* in Figure 5-1.

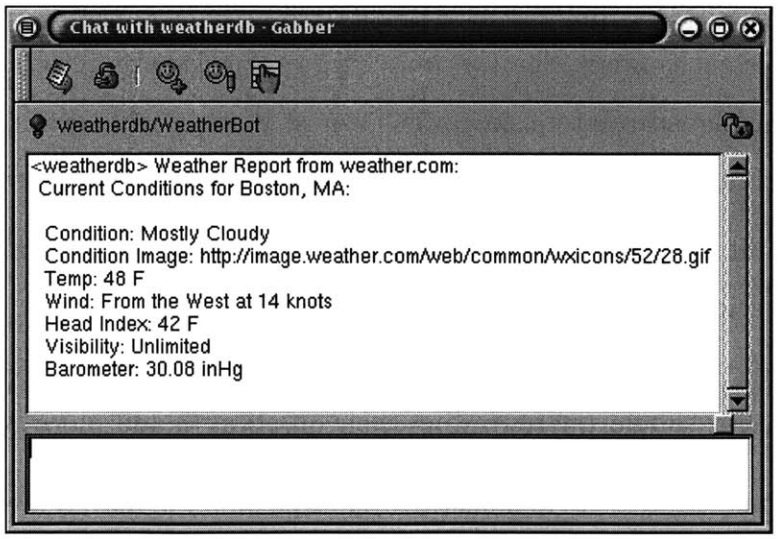

Figure 5-1: Automated Weather Response using Freebrain

The weather application reflects the simplicity that Freebrain provides to end-users and content providers to find one another with minimal effort. An attractive point about Freebrain is that an end-user may get an automated response from a knowledge database and also be able to have a quick real-time conversation with other interested people.

5.3 Design and Implementation Issues

5.3.1 Achieving fault tolerance

So far our discussion of the Freebrain servers has considered only how to achieve load balancing and scalability using a decentralized and distributed architecture. But what happens to the subscription lists when a particular Freebrain server fails? We need to add fault tolerance to build a robust Freebrain network.

We achieve this goal by replicating the subscription lists. Essentially, whenever we wish to add a new subscriber to a particular category, rather than find just one responsible server, we find k backup nodes (using `getBackupNodes` from the Chord API) and store the subscription on all k servers. If the primary node fails, we can get the subscription list from among the remaining k backup nodes.

To increase robustness further, the network ought to recover even when all k responsible servers for a particular category fail. We can achieve this by periodically refreshing the subscriptions, thereby always maintaining k backup nodes for any given category. In particular, we can use two methods:

- the Freebrain transport initiates periodic renewal of subscriptions for its local users (those that connect to its parent Jabber server).
- one or more of $k + 1$ nodes responsible for a category initiate periodic renewal of subscriptions. This assumes that not all $k + 1$ servers will fail within the refresh time interval.

We note that the periodicity of the renewal should be dependent on the number of servers participating in the network, their arrival and departure rate, and average rate of failure of each server. However, the basic assumption is that Freebrain servers will be permanent nodes, so dealing with frequent node arrival or departure is not the primary concern. The main element to be concerned with is network link failure, whereby a particular node may get temporarily disconnected from the remaining Freebrain network. This failure can easily be taken care of by the k-replication of subscriptions and a relatively low subscription renewal rate.

5.3.2 Graceful degradation

When a Jabber or Freebrain server fails, our system provides graceful degradation of service. We discuss these two cases individually:

Jabber Server failure — when a Jabber server fails, it disconnects all users that login to it and prevents them from using instant-messaging, let alone using Freebrain. However, it affects only these local users. All others that are connected via other Jabber servers experience no effect on their service, including Freebrain. If necessary, the users that get disconnected can resume service by creating an account on another Jabber server.

Freebrain Server failure — when a Freebrain server fails, it affects the local Freebrain transport that uses it to perform `QUERY` and `(UN)SUBSCRIBE` operations. When a Freebrain server fails, it prevents the local Freebrain transport from allowing its users (those that are connected to the parent Jabber server) to ask questions or modify subscriptions. However, these are the only services that are affected. Users of the Jabber server continue to receive invitations from people connected to other Freebrain servers, and they can also join conversations.

The distributed nature of Jabber and Freebrain servers, as well as the system modularity, allows this graceful degradation of service, in comparison to complete failure in a centralized approach.

5.3.3 Size and type of invitees

While technically our system can handle forwarding an invitation to thousands of users, that does not make practical sense. Sending someone's question to thousands of people could result in too many responses. Also, if knowledgeable people get too many invitations they may begin discarding most of them, in which case the inviter may not get a good response. A better approach is that if there are too many invitees, Freebrain should select only a subset (possibly random) to which to forward the invitation. Another mechanism would, if the inviter does not get any

response from the initial set of invitees, forward the invitation to a different subset. Based on observations in current chat systems like IRC, the largest (and hence, least constructive) chat rooms have no more than a thousand participants. Most chat rooms have fewer than 50 participants. In Freebrain, the equivalent of participants are invitees. A more detailed study is needed to determine the right number of invitees. However, we note that Freebrain is far more passive (and hence better) in nature than current chat systems. Invitees who receive a question may simply ignore it. They do not have to be present for the actual conversation, unlike IRC or other IM networks where presence in a chat room translates to receiving all messages in that room.

A second related issue has to do with the type of invitees, particularly, machines (on-line services or databases). What should Freebrain do if there are multiple machine subscribers in a given category? In some cases (such as weather), forwarding an invitation to all machine subscribers is unnecessary; it is better to just pick one. But in other cases (such as a knowledge-based question or a search), an end-user may wish to receive responses from multiple sources.

The ability to handle issues related to the size and type of invitees can help improve the quality of service. There is opportunity to study current systems and also gather empirical evidence from the initial Freebrain release to tackle these issues.

5.4 Future Work

Implementing fault tolerance — we should add fault tolerance to the Freebrain servers by implementing the suggestions made in Section 5.3.1. We should also consider how a Freebrain transport can contact another Freebrain server if the local one fails. This is relatively simple to add, since we only require the Freebrain transport to be able to query its local Chord node to locate another Freebrain server.

Searchable conversations history — although we have already implemented the ability to log conversations, it would be desirable to allow users to search

through the archives to find answers. Besides web-based search, we can reuse Freebrain's message routing capabilities to provide domain-specific results as instant-messages to inviters. For instance, we can add front-ends that subscribe to specific (or all) categories; when these front-ends receive questions, they can search the archives and return appropriate results.

Decentralized message archival — the current logging mechanism for archiving invitations and conversations uses a central server. However, a decentralized approach would allow the message archival process to be highly scalable and robust to failure of any particular log server. One idea is to spread the logs to different servers based on the categories (using Chord's `getNode`). All invitations and conversations concerned with one category will be logged on one server, and those with a different category on another.

Tight integration with web searches — when a user asks a question to Freebrain, the query can (optionally) start with a search through popular search engines (say, Google) and return the results. This provides a tight integration of IM with web search engines.

Location-based routing — we could allow the scope of invitations to be restricted based on location. This would be useful if the inviter wishes to first ask his question to a small community such as a research lab like LCS, a student dormitory, or a corporate office floor, and then gradually increase the scope to the wider Internet if he does not receive any response. Location-awareness can be obtained using existing systems such as Cricket [5] for indoor geographic location. Other notions of location can be obtained using domain names (say, to reach users connected to a particular Jabber server), IP addresses (say, to reach users only at LCS), *etc.*.

User rating — we should allow an inviter to rate an answer (hence, indirectly rating the user who answered) to improve the quality of content over time, especially in conversation histories, and to assist other inviters in deciding the reliability

of answers from users. Freebrain could benefit from existing research done in peer-based user ratings in systems such as Advogato [1] and Slashdot [28].

Improve category selection — we should improve the category selection by using novel algorithms. One idea is to weight a category based on the number of conversations that have occurred in it and the current number of subscribers. Forwarding the invitations to subscribers of a category that is more popular can improve an end-user's chance of receiving an answer.

Achieve interoperability — we initially proposed the idea of Freebrain with the desire to reach people across different IM systems. Although Jabber already provides an infrastructure for interoperability, we have yet to add the hooks so users from other IM networks (such as Yahoo! or AOL) can add a *freebrain agent* to ask questions and contribute answers.

5.5 Conclusions

Freebrain augments the current IM paradigm by providing a framework that allows people to have dynamic, interest-based group conversations in real-time. Unlike any present system, Freebrain offers both personalized conversations and a universal access point for seeking help from both machines and people. Freebrain has a distributed and scalable architecture that enables real-time response systems to plug into Freebrain and immediately offer their knowledge to seekers.

In accomplishing our initial goal, we realized the breadth and scope of work required to enable Freebrain to reach its full potential. The technical challenges in implementing the full vision of Freebrain include not only some of the hardest problems in the computer science (and artificial intelligence), such as natural language processing, but also encompass a broad set of areas such as distributed network systems, human-computer interaction, and information retrieval. Still, Freebrain is an important step in the right direction—of making the Internet friendlier and more useful, not only to the novice but also to the advanced user.

Bibliography

[1] Advogato's trust metric. http://www.advogato.org/trust-metric.html, Apr 2002.

[2] Ask Jeeves. http://www.askjeeves.com, Apr 2002.

[3] Chatbot. http://www.jabberstudio.org/cgi-bin/viewcvs.cgi/chatbot, May 2002.

[4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2000.

[5] The Cricket Indoor Location System. http://nms.lcs.mit.edu/projects/cricket/, Apr 2002.

[6] Gabber: The GNOME Jabber Client. http://gabber.sourceforge.net, Apr 2002.

[7] Geo::Weather. http://freshmeat.net/projects/geoweather/, May 2002.

[8] GNU Pth - The GNU Portable Threads. http://www.gnu.org/software/pth/, Feb 2002.

[9] Gnutella. http://gnutella.wego.com, Apr 2000.

[10] Google. http://www.google.com, Apr 2002.

[11] Google Groups. http://www.google.com/googlegroups/basics.html, Apr 2002.

[12] Google Technology. http://www.google.com/technology/, Apr 2002.

[13] Internet Relay Chat. http://www.irc.org, 2002.

[14] Internet Relay Chat: Architecture. http://www.ietf.org/rfc/rfc2810.txt, 2000.

[15] Jabber Software Foundation. http://www.jabber.org, Sep 2001.

[16] jabberd - the original open server implementation of Jabber. http://jabberd.jabberstudio.org, Feb 2002.

[17] Jabberd FAQ. http://jabberd.jabberstudio.org/faq.html#AEN93, Mar 2002.

[18] Jogger - A Jabber Powered Weblog. http://jogger.jabber.org, Apr 2002.

[19] Kerberos: The Network Authentication Protocol. http://web.mit.edu/kerberos/www/, Dec 2000.

[20] Kalpak Kothari and Nikolaos Michalakis. Design and Implementation of an Efficient Late-binding Mechanism in Twine. Final Project Report for Computer Networks course, Dec 2001.

[21] MIT Information Systems: Athena Computing Facility. http://web.mit.edu/is/athena/, Mar 2002.

[22] MySQL Full-text Search. http://mysql.org/documentation/mysql/bychapter/manual_Reference.html#Fulltext_Search, Apr 2002.

[23] Newsgroup Creation Procedure. http://www.geocities.com/ResearchTriangle/Lab/6882/ncreate.html, Sep 2001.

[24] ODP - Open Directory Project. http://dmoz.org, Apr 2002.

[25] Open Directory RDF Dump. http://dmoz.org/rdf.html, Apr 2002.

[26] PHP: Hypertext Preprocessor. http://www.php.net, Apr 2002.

[27] Psi Jabber Client. http://psi.affinix.com, Apr 2002.

[28] Slashdot's moderation system. http://slashdot.org/faq/com-mod.shtml, Apr 2002.

[29] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *ACM SIGCOMM 2001*, Aug 2001.

[30] Zephyr Answers. http://web.mit.edu/answers/zephyr/, Apr 2001.