Keio University Master's Thesis Academic Year 2003

GGCAST - A Location Based Multicast -

Keio University Graduate School of Media and Governance

Koshiro Mitsuya

Abstract of Master's Thesis Academic Year 2003

GGCAST - A Location Based Multicast -

Summary

Geographic Grouping Multicast (GGCAST) is a new one-to-many communication method in which receivers are determined by their geographic positions. Applications of GGCAST include dissemination of traffic information within relevant locations, such as passing the cause of a traffic jam to cars running behind. The objective of this research is to propose the architecture of GGCAST for a mobile ubiquitous computing environment.

Since, in GGCAST, a multicast group is defined by geographic domain, its membership changes dynamically, and it is difficult to keep a logical multicast tree for data delivery. The fundamental propositions of this research are dynamic group membership and efficient data transmission in such a multicast group.

In this paper, GGCAST is proposed as an application layer multicast (ALM) using a peer-to-peer (P2P) lookup algorithm with a location based service directory. In the targeted environment, nodes are freely moving over several ISPs. This makes ALM to be more suitable for the situation because it doesn't need a network layer protocol support. In GGCAST, members of a multicast group construct a overlay network using routing information of a P2P lookup algorithm, and a logical multicast tree is built on the overlay network. A member is discovered with the location based service directory as an initial node of the overlay network. The overlay network is using Chord [1], a P2P lookup algorithm. The routing information of Chord has little influence on the overlay network even when a node fails. Because membership of GGCAST changes frequently, the robustness is important.

GGCAST has been implemented as an ITS-related application for evaluation. It has been confirmed that our directory server can process about 2,000 queries per second. It is estimated that about 4,000 servers would be sufficient to provide a globally available version of this directory service. Other results include that even in a case in which membership changes every 0.9 seconds, over 99% of the members can receive the multicast delivery of information. We conclude that GGCAST realizes a practical location-based multicast.

Keywords:

location based multicast, dynamic group membership, peer-to-peer network, ubiquitous computing, ITS

Keio University Graduate School of Media and Governance

Koshiro Mitsuya

修士論文要旨 2003 年度 (平成 15 年度)

GGCAST - A Location Based Multicast -

論文要旨

Geographic Grouping Multicast (GGCAST) は、受信者が地理位置情報によって選択される 1 対多型のコミュニケーションである。サービス例として、後続の車に渋滞の原因を伝えるアプリケーションが挙げられる。本研究の目的は、モバイルユビキタス環境下の地理位置情報に基づいたマルチキャストである GGCAST を提案することである。

地理位置情報によってマルチキャストグループが定義されるので、受信者が動的に変化する。そのため、マルチキャストのための論理経路パスを維持することが難しい。したがって、本研究における主題は、動的なグループメンバ管理とそのグループ内での効率的なデータ配送である。

GGCAST は、地理位置情報に基づいたサービスを発見するためのディレクトリサービスと P2P コンテンツ発見アルゴリズムを用いたアプリケーション層マルチキャスト（ALM）から構成される。想定環境では、ノードは複数の ISP 間を自由に移動する。ALM はネットワーク層プロトコルの対応が不要であるので、想定環境で利用するのに適している。GGCAST では、マルチキャストグループ内のメンバ同士が P2P コンテンツ発見アルゴリズムによる経路情報を用いてオーバレイネットワークを形成する。そして、そのオーバレイネットワーク上にマルチキャストのための論理経路パスが構築される。この際、オーバレイネットワークに参加するための初期ノードは、提案するディレクトサービスを利用して発見される。なお、メンバの動的な変化に対して強い Chord アルゴリズム [1] を P2P コンテンツ発見アルゴリズムとしてを利用した。

GGCAST は ITS 関連アプリケーションとして評価を行った。評価実装を用いて測定した結果、ディレクトリサーバは毎秒 2000 クエリ程度を処理できることが確認された。また、4000 台程度のサーバを用意すれば、世界中でこのディレクトリサービスが利用できることが算出された。また、0.9 秒毎にメンバが変化するような状態でも、メンバの 99 %以上がマルチキャスト配送を受信できることが確認された。したがって、GGCAST を用いて、地理位置情報に基づいたマルチキャストが実現できることが分かった。

キーワード:

地理位置情報に基づいたマルチキャストグルーピング、動的メンバ管理、ピアツーピアネットワーク、ユビキタスコンピューティング、ITS

慶應義塾大学大学院 政策・メディア研究科
三屋光史朗

Acknowledgments

I would like to thank my thesis supervisors, Professor Jun Murai, Professor Hideki Sunahara (Nara Institute of Science and Technology) and Associate Professor Hiroyuki Kusumoto for their guidance and advice throughout the process of writing this thesis. I am also grateful to Associate Professor Osamu Nakamura. He always gave me insights to this research and guided me to write the thesis in a more appropriate manner. This thesis benefits greatly from his insights, suggestions, and patient revisions.

I would like to thank Dr. Kenjiro Cho (Sony CSL, Inc.), Dr. Keisuke Uehara, Dr. Kazunori Sugiura, Dr. Noriyuki Shigechika, Dr. Yasuhito Watanabe, Masaki Minami, Yuji Sekiya (The University of Tokyo), Ryuji Wakikawa, Shinta Sugimoto (Ericsson Japan, Inc.), Yasuhiro Ohara and Masaaki Sato for their enduring support and guidance. They have always took great care of me in learning from the time I first enrolled into this laboratory. Their efforts have routed me to today's activity.

A huge thanks to all the members of the Internet Research Laboratory for making work a lot of fun. I am especially grateful to Dr. Thierry Ernst, Kenji Saito, Susumu Koshiba, Masafumi Watari, Koji Okada and Hiroki Matsutani for providing support and insight while I was writing my thesis.

Thanks also to my colleagues (past and present) for their daily assistance and encouragement. I greatly enjoyed working with them, and I will, especially, miss Tetsuji Hino, Gen Kobatake, and Keita Miyajima. I also would like to thank Kotaro Kataoka, Takashi Shimizu, Shun Hirose, Hiroaki Takahashi, Daisuke Naruse, Kazuki Hashimoto and Yohei Tanioka.

I would also like to thank Bakery & Restaurant Kinoko, New Orleans and Bar Kaito. They have enhanced the pleasure of not only the table but also my living.

Lastly, and most importantly, I would like to thank my family for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the background of this research. First of all, the proposed model called Geographic Grouping Multicast (GGCAST), a location based multicast, is defined. Next, its importance in real-world situations is explained. Then, our objective in this research is described. At last, the organization of this thesis is described.

1.1 Background

A number of applications such as large-scale file distribution, Internet TV, video conferencing and shared white boards [2] [3] [4] [5] require one-to-many message transmission to enable efficient many-to-many communication. Multicast is a communication between a single sender and multiple receivers on a network. Together with anycast and unicast, multicast is one of the communication types in the Internet.

Multicast is often used in a situation where an application needs to send the same information to more than one destination. This is because multicasting consumes much less network resources compared to unicasting to each destination. Network resource consumption should be considered with higher importance in the mobile ubiquitous networking environment where mobile hosts communicate with each other over wireless links.

Meanwhile, the needs for mobility support in the Internet has grown significantly in recent days. For example, it is useful to connect mobile hosts such as cars and trains to the Internet and exchange their information freely. Probe Information Service [6] and InternetITS [7] in which our Internet CAR Project [8], have been participating are good examples of such information services.

The Probe Information Service utilizes automobiles as rovering sensors

scattered over many locations. Automobiles are connected to the Internet, providing Internet connectivity to each sensor which acquires information such as location, velocity, and movement of wipers. By collecting information from such sensors in the automobiles via the Internet, new kind of information, such as knowledge of a traffic congestion or the surrounding weather constructed from collected information, can be obtained.

The derived information is always up-to-date, and is much more detailed than what existing systems can provide. The InternetITS is a concept which uses the Internet as the common ITS infrastructure. ITS (Intelligent Transport System) [9] is a system designed to promote the advancement in the car navigation technology to give users more effective driving supports. The Electronic Toll Collection system (ETC) [10] is a good example of driving supports using computer technologies which has reduced the traffic congestion on toll-ways by automatically collecting the toll fees. It is expected that several ITS services are achieved efficiently with the common infrastructure based on Internet technologies

In this world, a lot of nodes are connected to the Internet with mobility supports [11] [12] at all time and all places, those mobility supports allow to keep connections alive while hosts or networks are moving around. And they use network resources distributed ubiquitously [13]. We call this *mobile ubiquitous computing environment.*

1.2 Fundamental Propositions

In order to achieve multicasting, it is required to find some way to define a multicast group. In conventional multicasting algorithms [14], a multicast group is considered as a collection of hosts which register to the group. In order to receive multicast messages, a node must first join a particular group. A distribution tree rooted at a traffic source is constructed within the group members. When a host sends a message to the multicast group, it just sends the message to the IP address representing the group. The message is delivered according to the tree. All the group members then receive the message.

Geographic Grouping Multicast (GGCAST), a location based multicast, is one of one-to-many communication in which receivers, members of a multicast group, are determined by their geographical positions. Its applications include advertising regional information to travelers or telling the cause of a traffic jam to cars running behind. One of the interesting feature of GGCAST is that membership of the multicast group changes dynamically because nodes move physically and the group is determined by their locations.

Since a member changes dynamically, this makes it difficult to maintain the distribution tree for data delivery.

The fundamental propositions of this research are the architecture to achieve the following:

- Dynamic group membership
- Efficient data transmission in the multicast group

1.3 Objective

Geographic Grouping Multicast (GGCAST) is one of one-to-many communication which members of a multicast group are determined with geographical position information. The objective of this research is to propose GGCAST, a location based multicast architecture in mobile ubiquitous networking. To our best knowledge, there is no preceding researches in computer networking area. Thus we believe that this research deserves serious attention.

To achieve this objective, we will analyze characteristics of the targeted environment, and discuss appropriate approaches corresponding to the characteristics based on the analysis and the opinions of existing works. We propose a new location based multicast model based on one of the discussed approaches, and we simulate the model in different scales for evaluation. The simulation models an actual application in the targeted environment.

1.4 Organization

The targeted environment of this research is described in Chapter 2. In Chapter 3, the appropriate approaches and the GGCAST architecture are discussed. In addition, usage scenarios are introduced to clarify requirements for the system. In Chapter 4, related works of this research are introduced, and the differences from this research are discussed. Chapter 5 describes the detail of GGCAST. Chapter 6 explains our implementation of the model. The model is evaluated by simulations whose conditions and results are discussed in Chapter 7. Chapter 8 addresses the area for future study and concludes the paper.

Chapter 2

Targeted Environment

In this chapter, we describe the targeted computing environment. We define our targeted environment as a mobile ubiquitous computing environment. In this world, network resources are ubiquitously distributed, and Internet nodes, e.g. Cars, laptops and PDAs, are always connected, regardless of the changes of their locations and access points to the Internet. Meanwhile, a lot of applications using physical information such as geographical location information will appear.

2.1 Mobile Ubiquitous Computing

Ubiquitous Computing is Coming Up

The word "Ubiquitous" gained a lot of attention recently. The origin of the word means that it's omnipresent in Latin, which means that it can be used anywhere and anytime. Ubiquitous Computing have applied this concept in the world of computing. Mark Weiser of Xerox PARC research institute advocated this notion in 1989. His paper [13] describes a world where wirelessly networked computers are distributed throughout the environment.

The technology of small information terminals, such as a cellular phone, PHS and PDA, is progressing, and the attention to the ubiquitous computing has greatly gathered in recent years.

Constant Internet Access for Mobile Hosts

As a network for realizing Ubiquitous Computing, wireless technology has a big possibility. A cellular phone is a good example of the wireless communication tool. The mobility and coverage superior to other technology are

the reason why the cellular phone became popular in our life. Here, mobility means migration support and coverage means the area where service is offered.

In recent comparison, the ratio of the data communication is increasing by the cellular phone. As Figure 2.1 shows, number of cellular phones which has Internet access function is increasing in Japan [15].

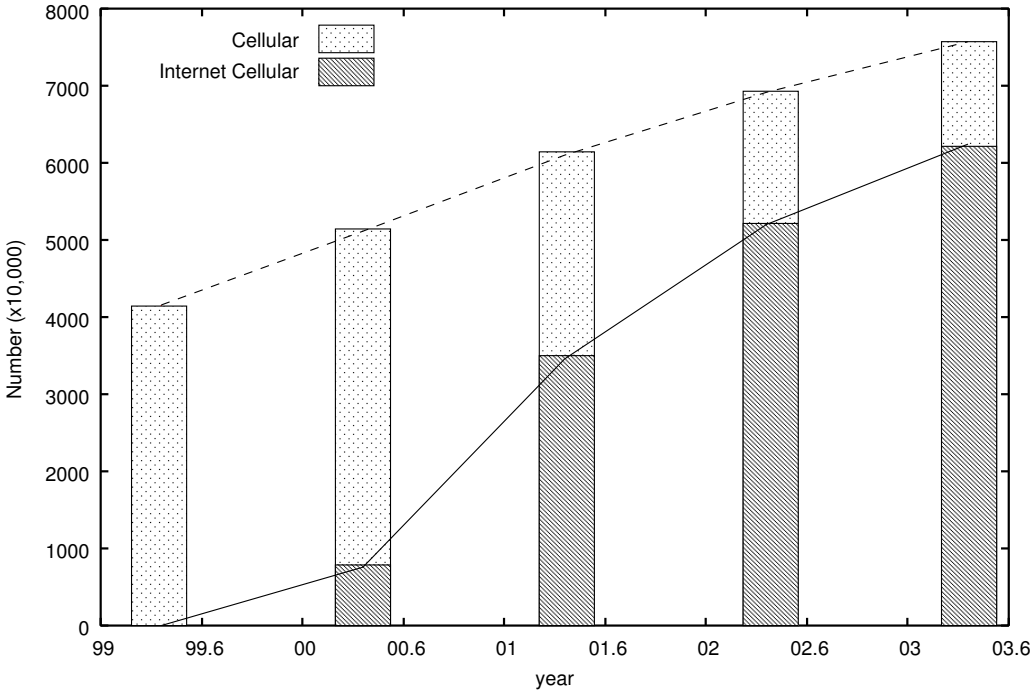


Figure 2.1: Changes in number of cellular phone with and without Internet access function in Japan

However, data traffic in the current cellular phone system requires high cost. When transmitting even a small data packet, the system must reserve a circuit to the destination. Thus, the method to build a network infrastructure suitable for data communication becomes a key from now on.

Multimedia Research Institute Corporation propose to fulfill providing the network infrastructure which combines IP technology with TD-CDMA [16]. TD-CDMA [17] is one of the communication systems standardized as IMT-2000 of the third generation cellular phone (3G). TD is an abbreviation for Time Division; uplink/downlink circuit of a communication is segmented by time. Compared to the existing Frequency Division, it is said that affinity with data communication is higher. It is possible to build a very efficient

and simple data communication network by combining IP technology. The bandwidth is expected to be 100Kbps for upload and 1Mbps for download.

In the near future, there will be many wireless hotspots on the streets, offices, trains, resorts and everywhere. These hotspots are assumed to support IP network. Therefore, the mobile hosts ubiquitously distributed in the world can obtain the Internet connectivity anytime and anywhere.

Mobility Supports in IP Networks

Several mobility support protocols are developed to support IP nodes under various situations. Host mobility is for a mobile host to hide movements from the Internet. Mobile IPv6 [11] is designed to support such host mobility and is being standardized at the Internet Engineering Task Force (IETF) [18]. Network Mobility protocol, known as NEMO, conceals movements for a network that moves entirely. NEMO is being discussed at the NEMO working group at IETF [12]. A "mobile ad hoc network" (MANET) is an autonomous system of mobile nodes connected by wireless links without a particular infrastructure. Several MANET routing protocols, such as DSR[19], AODV[20], OLSR[21], TDBRPF[22] have been proposed for a MANET with a dynamically changing topology.

With these mobility support protocols, mobile nodes can obtain Internet connectivity just as any fixed nodes. Thus, the changes in communication environment for the mobile nodes are not considered in this paper.

2.2 Practical Use of Position Information

In the future, position information sensing systems will be deployed in every mobile node. Position information will become an information that is as common as time; receiving input from Global Positioning System [23], when outdoors, and from other position information providing devices, when indoors.

The car navigation system is a well known mobile node using GPS to obtain position information. Figure 2.2 shows the shipping volume of car navigation system in Japan [24]. The graph shows the total number of the car navigation systems shipped in label y and its corresponding year in label x. The estimated volume of shipments of the car navigation systems is 12 billion at 2004 and its volume is increasing year by year.

In the near future, all vehicles will be equipped with car navigation systems with Internet connectivity. In fact, a part of present car navigation system is already equipped with such function, the Internet is used to up-

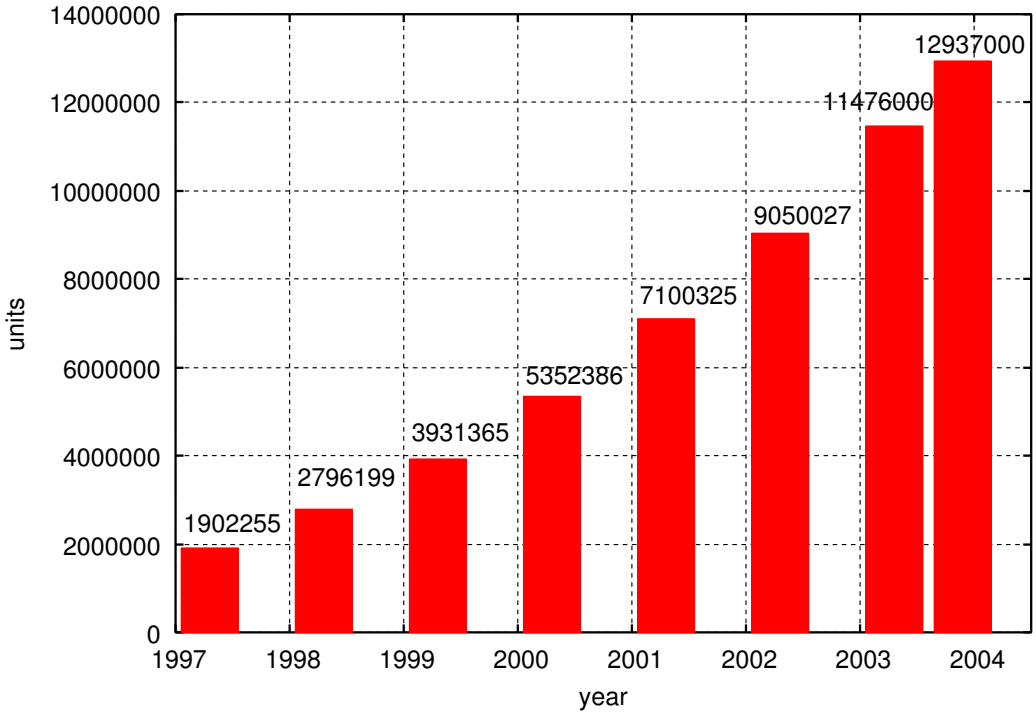


Figure 2.2: Total number of the car navigation systems shipped per year in Japan (c)JEITA

grade maps or to provide route search using the corrected road information from the server. Thus it is expected that many car navigation systems have Internet connectivity soon. This is a example of practical use of position information.

GPS is introduced as a position sensing technology here, however, there are many other position sensing technologies [25]. Details are listed in Appendix A.

As described above, position information can be used wherever even if it is used indoor or outdoor. All devices connected to the Internet - high spec workstations, tiny nodes, and any nodes - use a technology that best matches their use. Thus, it is expected that every mobile computer are connected to the Internet and equip a position information sensing system, they use position information on a routing application.

2.3 Summary

Ubiquitous Computing is coming up. In ubiquitous computing, nodes are connected to the Mobility supported Internet anywhere and anytime. User can access to various Internet services from terminal computers. This world is defined as "Mobile Ubiquitous Computing". Additionally, position information can be used whether even if it is used indoor or outdoor. This research targets on this environment (Figure 2.3).

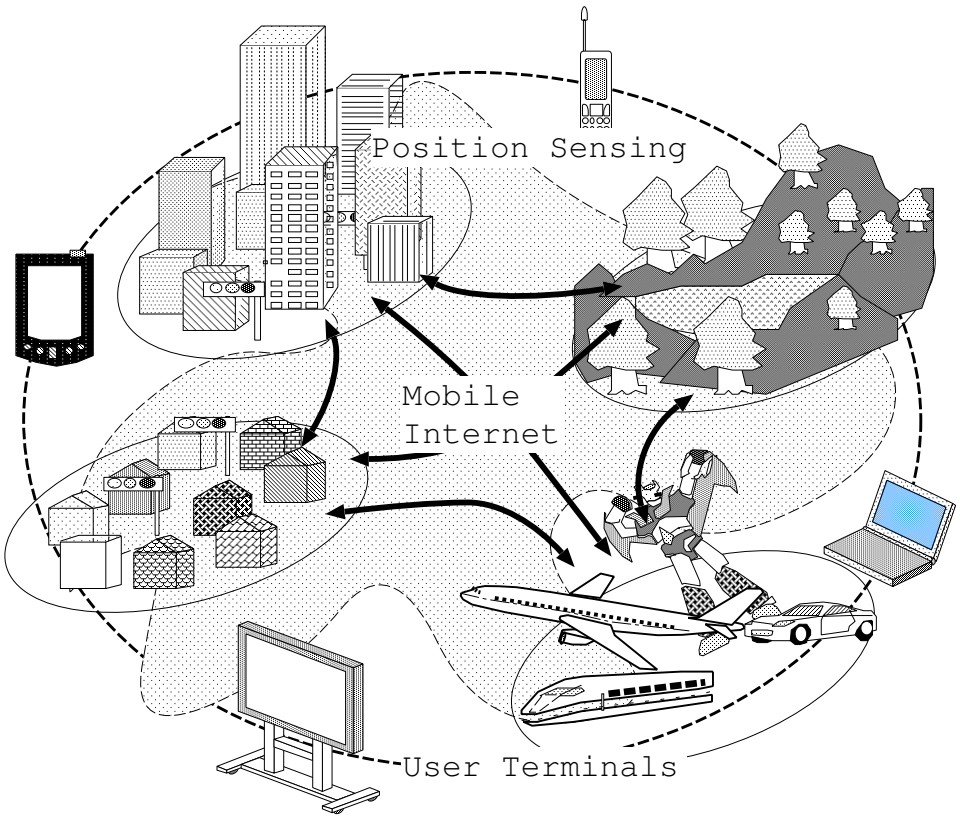


Figure 2.3: Targeted Environment

Chapter 3

GGCAST Architecture

In this chapter, we describe Geographic Grouping Multicast (GGCAST), a location based multicast architecture. GGCAST is one to many communication model where receivers are selected using geographical position information. This chapter also describes the scenarios of using GGCAST and describes examples to clarify the requirements for the architecture.

3.1 Approach

Multicasting typically consists of multicast group management scheme and multicast routing. In conventional multicasting algorithms, a multicast group is considered as a collection of nodes which register to that group. For a node to receive any multicast message, it must first join a particular group. A distribution tree rooted at a traffic source is constructed within the group members. When a node send a message to a multicast group, sending the message to the IP address of that multicast group is all it takes. The message is delivered according to the tree, all the group members then receive the message.

The interesting feature of GGCAST, as shown in Figure 3.1, is that the members in a group change dynamically according to the physical movement of the nodes, since the group is formed based on geographic position information.

A doted square in Figure 3.1 represent a multicast region. N1, N2, N3,,,N9 are mobile nodes. First, members are N1, N2, N3 and N4. Because mobile nodes are moving around, the members are changing dynamically; members change to N4, N5, N6, N7 and N8 in the figure.

In order to achieve dynamic group membership, we propose following approaches. Details on each approaches are described in this section.

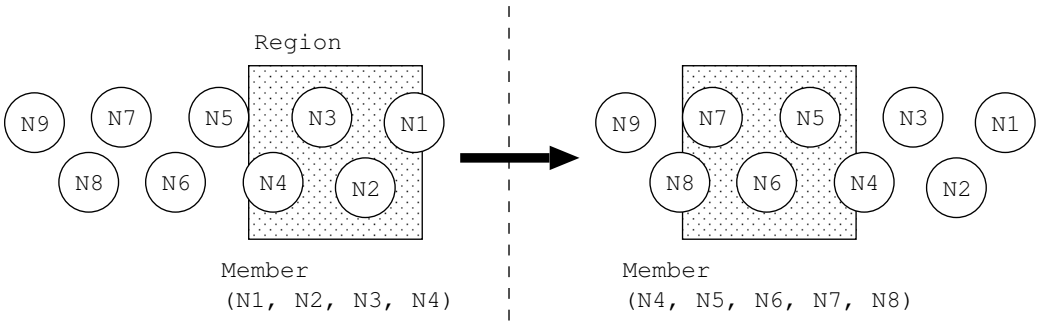


Figure 3.1: Multicast grouping with position information

- Multicast grouping with geographic information
- Join/Leave model
- End node organization

In order to achieve efficient data transmission in the multicast group, we propose following approaches.

- End node Organization
- Overlay Network which has robustness when a node fails
- Less Impact of Members Failures
- Less Packet Duplication

Multicast Grouping with Geographic Information

With IP Multicast, the group is abstracted as multicast IP address. In order to join the group, a node is required to send requests. A distribution tree rooted at the traffic's source is constructed based on the request. A single packet transmitted at the source is delivered to an arbitrary number of receivers by replicating the packet within the network at fan-out points, routers, along a distribution tree rooted at the traffic's source.

A multicast group of GGCAST is determined with geographic information. Although an address is one method to specify a region of geographic information, the region can simply be specified with longitude and latitude.

Join/Leave Model

It is preferable that only the desired node receives multicast traffic, and not all nodes in the region. Although a model where all nodes receive the traffic is far easier for multicast group management scheme, it is not desirable because the resources on mobile node are limited. Therefore, each node has to decide, actively, which multicasting packet it receives. GGCAST divides the world into a number of regions. Therefore, a kind of directory service is necessary to discover the multicast group currently being offered at a certain point by some methods.

End Node Organization

The nodes in a multicast group are connected to the Internet by various media. For example, cellular, PHSs and Wireless LAN are used; nodes change these medias by situation. Even if we are in the same place, we use cellular phones of different carriers. Thus, nodes may connect with separate ISP, even if each node is in near geographic position.

Due to the fact that communication environment changes accordingly with the ISPs connected, inter-domain operation is difficult in most cases. The system that requires alterations in relay nodes is hard to use. ISP provides only the Internet connectivity. Thus only the end node should realize the GGCAST system.

In addition, members in multicast group should perform themselves, because huge number of groups exists all over the world. Since end node organization dose not concentrate on a center system, it has advantageous in robustness, and load distribution is also realizable.

Less Impact of Member Failures

Since members in a group change dynamically, the system should have a robustness to member failures. When a node leaves or fails from the multicast group, the influence on the system should have less impact.

Less Packet Duplication

Since the bandwidth of mobile nodes is limited, multi-unicast by single node is difficult. Members should relay packets efficiently.

This discussion is going to be about make efficient logical distribution tree for multicast delivery. In order to make “no packet duplicate” tree, high costs, e.g. control packets and route calculation, may be required. However, simple flooding, each node reply to their neighbor, takes long times to finish

the deliver and occurs a lot of packet duplication. This is a tradeoff between the costs and its efficiency.

3.2 Architecture Overview

The proposed GGCAST architecture consists of following two functionalities:

- Group management scheme
- Multicasting scheme

Group management scheme includes a directory service to find pointers to multicast groups with location information as search key, and membership management functions to support dynamic alteration of members. Multicasting scheme includes multicast routing to enable efficient data delivery/copy to reduce network load. The scheme is done by the end nodes, and not concentrated on a centralized system.

Figure 3.4 shows an overview of location based multicast architecture. The circle represents a node. A multicast group is described as a rectangle in the figure. There is a directory service to discover the multicast group currently offered at a certain place.

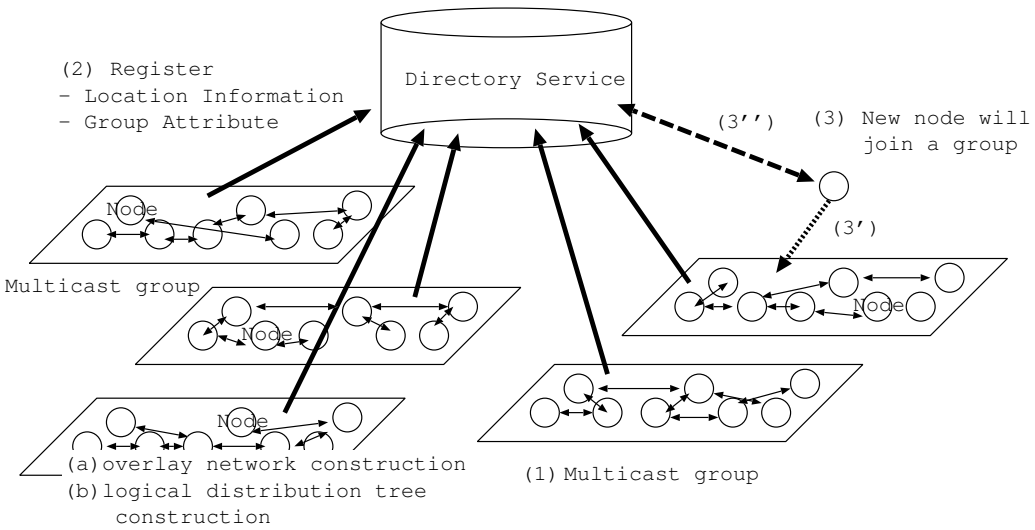


Figure 3.2: GGCAST Architecture

3.2.1 Group Management Scheme

(1) Nodes in a rectangle organize a multicast group. A member of the group offers a multicast service by themselves. (2) Each multicast group registers its geographic positions and an attribute of the multicast group to the directory. A sender node performs the registration. (3) A node discovers a multicast group by using the directory service, and joins the multicast group.

3.2.2 Multicasting Scheme

The multicasting scheme consists of overlay network construction and multicast routing on the overlay network. The members of a multicast group self-organize into an essentially random application-level mesh topology (a). A multicast routing algorithm is used over the topology to construct distribution trees rooted at each possible traffic source (b).

Figure 3.3 shows multicast routing in a multicast group. The circle represent a node. The arrow shows data flow.

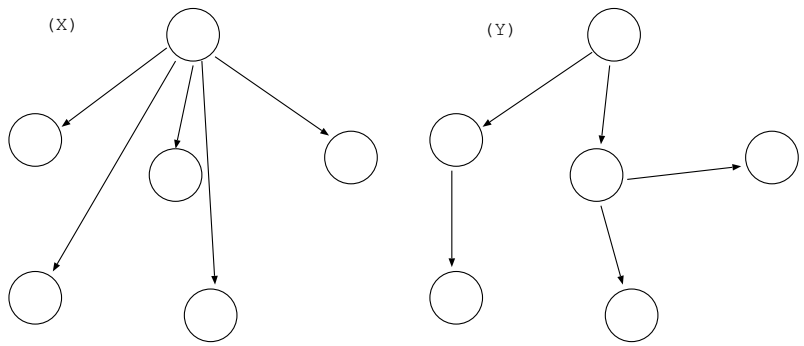


Figure 3.3: Multicast Routing in a multicast group:

Since the bandwidth of mobile nodes is limited, (X) multi-unicast by single node is difficult. It is desirable that the transmitting may be suboptimal (Y).

The propositions in the operation are how to organize overlay network which can be scalable in mobile ubiquitous environment and efficient datagram delivery with saving network resources.

3.2.3 Time Line of GGCAST

In order to figure out the relation between group management scheme and multicasting scheme, we describe the time line of location based multicast.

Figure 3.4 shows the fundamental time line of the proposed model. The figure shows a multicast group which consists of sender and receivers. Directory in the figure shows a directory service to find a multicast group, and N1, N2 and N3 indicate nodes.

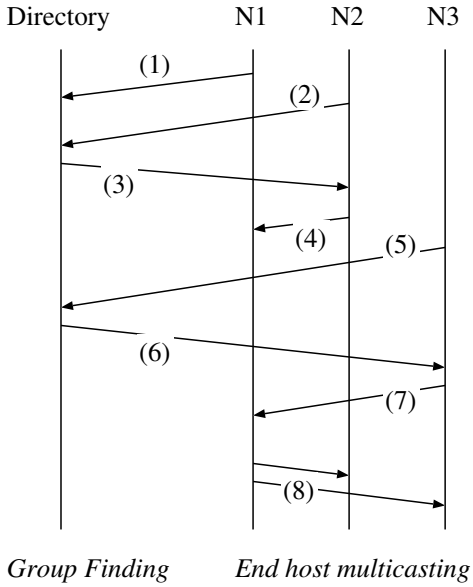


Figure 3.4: Time Line of GGCAST

N1 creates a new multicast group. (1) N1 registers the group to the directory with IP address, port number, the region and description. (2) N2 determines multicast group which serves a point included the region. (3) The directory answers the group created by N1 before. (4) N2 joins the multicast group according to a protocol of the group, members of the group are organizing an overlay network, with IP address and port number which answered by the directory as initial pointer. (5) N3 also determines multicast group which serves a point included the region. (6) The directory answers the group created by N1. (7) N3 joins the multicast group according to a manner of the group. (8) N1 sends data gram to the group, it is delivered by a flooding scheme.

3.3 Scenario : Providing Traffic Information to Cars Running Behind

A possible application of GGCAST is providing traffic information to cars running behind. Traditionally, hazard lights are used to inform the traffic jam to following vehicles. Although the drivers behind don't know, exactly, what has happened, but they could know that something has happened in front. Thus a method of obtaining more detailed information is very convenient.

Figure 3.5 shows a situation of this application. The circle represents a car, the two lines express a road. There is an accident in the direction of cars movement.

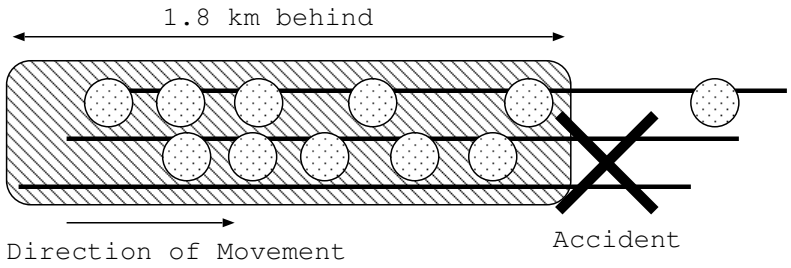


Figure 3.5: Application example: Providing traffic information to cars behind

A node creates a multicast group to tell the cause of this accident to cars 1.8km behind. Then, the multicast region is 1.8 km from the accident point, and the description of this multicast group might be "TRAFFIC JAM". Other nodes check multicast groups serving near themselves. The nodes can know there is the multicast group in their direction of movement by using the directory service. The directory also returns a pointer to the multicast group, nodes can join the group via the pointer. When nodes join to the group, nodes receive the information about this traffic jam. Nodes will leave the group when they pass the accident point.

When 1.8km squares region created from a accident point, the explanatory variables is estimated as Table 3.1. The multicast region is 1.8 km square from the accident point. Speed is means of node's speed (km/h), and Time is a time a node goes through the region. Depending on the Distance (meter) between the cars, number of nodes in the region is estimated as # of nodes.

"Speeds" as described in the table consider following situations. 100km/h is for highway, 60km/h for driving smoothly on streets in outskirts, 20km/h

Table 3.1: Explanatory variables in 1.8km-square region

Speed (km/h)	Time (second)	Distance(meter)	# of nodes
100	65	100	18
60	108	60	31
20	324	20	90
5	1296	5	360

for the streets in downtown, and 5km/h for traffic jam situations. A space between two cars is generally defined as 1/1000 of its speed.

From this estimation, following requirements about its performance are figured. When a traffic jam occurs;

- Number of node in the group is about 360.
- A node stays at the group about 1300 second.
- A node newly joins and leaves to the group per $5m/(5000m/h) = 3.6$ seconds

3.4 Summary

GGCAST architecture consists of group management scheme and multicasting scheme. Group management scheme includes a directory service to find pointers for multicast groups with a location information as keys, and support dynamic membership. Multicasting scheme consists of overlay network construction and multicast routing.

The following items are functional requirements for the GGCAST system.

1. Dynamic group membership
2. Directory service to discover a multicast group
3. Multicast routing realized by only end node.
4. No concentrate on a center system
5. Overlay Network which has robustness when a node fails
6. Less Impact of Members Failures
7. Less Packet Duplication

Chapter 4

Related Works

In this chapter, we explain related works of this research. IP Multicast is a widely accepted concept of multicast on the Internet. However, IP multicast has many fundamental issues, inherent in its original architecture, which need to be changed in order to implement GGCAST. Application Layer Multicast has argued as a more practical alternative to IP multicast, citing the end-to-end argument. To ensure better scalability, dynamic grouping and network fail-safe, application layer multicast has been implemented using peer-to-peer routing algorithms. We focus mainly on this application layer multicast in this thesis. We also describe differences between our GGCAST, a location based multicast, and other model called "location based multicast".

4.1 IP Multicast

The IP Multicast service [14] was proposed as an extension to the Internet architecture to support efficient multi-point delivery at the network level. With IP Multicast, a single packet transmitted at the source is delivered to an arbitrary number of receivers by replicating the packet within the network at fanout points, routers, along a distribution tree rooted at the traffic's source.

In his seminal work in 1989 [14], Deering argues that this second consideration should prevail and multicast should be implemented at IP layer. This view so far has been widely accepted. IP Multicast is the first significant feature that has been added to the IP layer since its original design and most routers today implement IP Multicast.

IP Multicast has been studied for many years now. Yet, IP multicast deployment has been slowed by difficult issues related to scalable inter-domain routing protocols, charging models, robust congestion control schemes and

so forth [26] [27] [28].

IP Multicast has several drawbacks that have so far prevented the service from being widely deployed. First, IP Multicast requires routers to maintain per group state, which not only violates the "stateless" architectural principle of the original design, but also introduces high complexity and serious scaling constraints at the IP layer. Second, the current IP Multicast model allows for an arbitrary source to send data to an arbitrary group. This makes the network vulnerable to flooding attacks by malicious sources, and complicates network management and provisioning. Third, IP Multicast requires every group to dynamically obtain a globally unique address from the multicast address space and it is difficult to ensure this in a scalable, distributed and consistent fashion. Fourth, IP Multicast is a best effort service. Providing higher level features such as reliability, congestion control, flow control, and security has been shown to be more difficult than in the unicast case. Finally, IP Multicast calls for changes at the infra-structural level, and this slows down the pace of deployment. While there have been attempts to partially address some of the issues at the IP layer [29] [30], fundamental concerns "stateful" architecture of IP Multicast and support for higher layer functionality have remained unresolved. Following it the above summary, several drawbacks of IP multicast.

- IP Multicast requires routers to maintain per group state.
- IP multicast model allows for an arbitrary source to send data to an arbitrary group.
- IP Multicast requires every group to dynamically obtain a globally unique address.
- IP multicast is a best effort service.
- IP Multicast calls for changes at the infra-structural level.

When implementing GGCAST on IP multicast, we face on other problems additionally. First, IP multicast itself has no functionality to assign a multicast address to the location based group. In IP multicast, every group obtain a multicast address, a globally unique address. The address typically assigned manually by administrators. Since there are large number of multicast groups in the world, the management of multicast addresses is difficult. Second, IP multicast is not necessarily effective. A comparatively few number of nodes form a multicast group and the nodes are distributed in two or more ISP. The case which IP multicast is most effective is that there is a lot of receivers in a ISP.

- No functionality to assign a multicast address.
- Not necessarily effective in mobile ubiquitous networking.

For the above reasons, IP Multicast is not suitable for GGCAST.

4.2 Application Level Multicast

Because of the problems facing the deployment of a network level multicast service as described at previous section, many recent research proposals have argued for an application level multicast service [26] [27] [31] and have described designs for such a service and its applications.

The majority of these proposed solutions (for example [27], [31] and RelayCast [32]) typically involve having the members of a multicast group self-organize into an essentially random application level mesh topology. In order to construct distribution trees rooted at each possible traffic source, a traditional multicast routing algorithm, such as DVMRP [14], is used over the topology.

Such routing algorithms require every node to periodically announce its estimated distance from every possible destination to its local neighbors and hence every node maintains state for every other node in the topology. Further, in the case of a change in the topology, every node must learn about this change and update its routing table if required. Hence, although these proposed solutions are well suited to their targeted applications, their use of global routing algorithm limits their ability to scale to large (more then a thousand nodes) group sizes and to operate under conditions of dynamic group membership.

To ensure better scalability and handling of dynamic groups and network failures, application level multicast has been implemented using peer-to-peer routing algorithms. For example, Bayeux [33] is implemented on top of Tapestry, and organizes multicast receivers into a distribution tree routed at the source. In Bayeux, nodes explicitly join and leave a multicast session by notifying the source node. The service model is limited to a single source.

SelectCast [34] is a peer-to-peer publish/subscribe routing service, built on *Astrolabe*, a peer-to-peer domain aggregation service. Both SelectCast and Astrolabe are heterogeneity-aware, but at the cost of some increased management overhead when compared to other peer-to-peer protocols. Although these services do not rely on any special-purpose servers, they can leverage asymmetries between hosts and between network connections. SelectCast is also restricting the service model to a single traffic source.

[35] has proposed an application level multicast scheme capable of scaling to large group sizes without restricting the service model to a single source. Their scheme leverages recent work on Content-Addressable Networks (CANs) [36]. Briefly, a Content-Addressable Network is an application-level network whose constituent nodes can be thought of as forming a virtual d-dimensional Cartesian coordinate space. Every node is a CAN "owns" a portion of the total space. For example, Figure 4.1 shows a 2-dimensional CAN occupied by 5 nodes. A CAN is scalable, fault-tolerant and completely distributed. Such CANs are useful for a range of distributed applications and services. For example, in [35] they focus on the use of a CAN to provide hash table-like functionality on Internet-like scales - a function useful for indexing in peer-to-peer applications, large-scale storage management system, the construction of wide-area name resolution services and so forth.

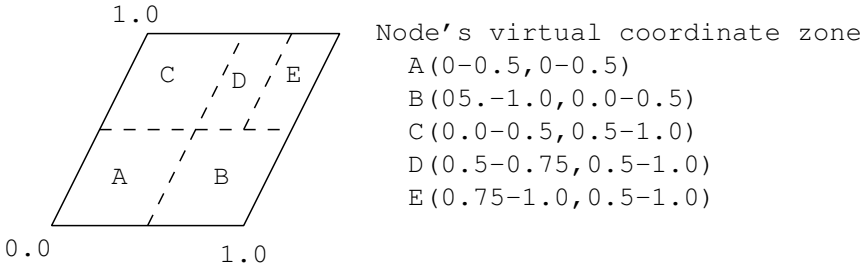


Figure 4.1: Example 2-d coordinate overlay with 5 nodes

Table 4.1 shows comparisons with another researches on application layer multicast. This paper looks into the question of how the deployment of such P2P distributed infrastructures might be utilized to support multicast services and application. We outline the design of an application level multicast scheme built using a Chord routing algorithm. Our design shows that extending the Chord framework to support multicast comes at trivial additional cost in terms of complexity and added protocol mechanism. A key feature of our scheme is that because we exploit the well-defined structured nature of Chord topologies we can eliminate the need for a multicast routing algorithm to construct distribution trees. This allows our Chord-based multicast scheme to scale to large group size.

4.3 Location Based Multicast in Geocasting

The term "location based multicast" is also used in the area of mobile ad hoc networking. A geocast [37] is delivered data grams to nodes by using geo-

graphical information. The location based multicast algorithms, geocasting, is delivered to the set of nodes within a specified geographical area. That indicates another meaning as we focus on location based multicast. Unlike the traditional multicast schemes, the multicast group (or geocast group) is implicitly defined as the set of nodes within a specified area. The set of nodes in the multicast region is the location based multicast group.

[38] discussed the problem of geocasting - broadcasting to every node in a specified geographical area- in mobile ad hoc environments. In [38], the specified geographical area is called the multicast region, and the set of nodes that reside within the specified multicast region is called a location-based multicast group. They propose two location-based multicast algorithms in geocasting. The proposed algorithms limit the forwarding space for a multicast packet to the so-called forwarding zone. Simulation results indicate that proposed algorithms result in lower message delivery overhead, as compared to multicast flooding. As simulation results show, while reducing the message overhead significantly, it is possible to achieve accuracy of multicast delivery comparable with multicast flooding. They also discuss how the basic location-based multicast schemes may be optimized to improve performance. As evaluation of these traditional multicast algorithms shows, it is possible to implement a location-based multicast by maintaining a multicast tree. A comparison between the algorithms presented in this paper and the alternative approach of maintaining a multicast tree is also a topic for further work.

Consider a node S that needs to multicast a message to all nodes that are currently located within a certain geographical region. This specific area is the "Multicast Region". The multicast region would be represented by some closed polygon such as a circle or a rectangle(see Figure 4.2). Assume that node S multicasts a data packet at time t_0, and three nodes (X, Y and Z) are located within the multicast region at that time. Then, the multicast group, from the viewpoint of node S at time t_0, would be have three members that are expected to receive the multicast data packet sent by node S. Accuracy of multicast delivery can be defined as ratio of the number of group members that actually receive the multicast packet, and the number of group members which were in the multicast region at the time when the multicast is initiated. For example, if only node X among three members of the multicast group actually gets a multicast packet, accuracy of delivery for the multicast packet will be 33.33%.

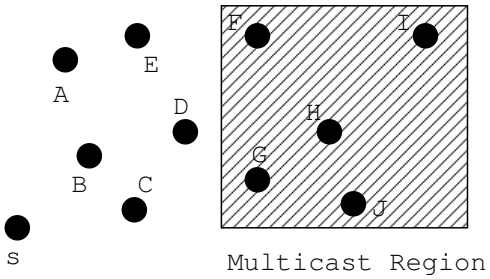


Figure 4.2: Location Based Multicast in Geocasting

4.4 Summary

In mobile ubiquitous networking, nodes are freely moving over several domains, thus IP multicast is not necessarily effective. In contrast, application layer multicast is recognized for the fact that it does not require network layer multicast protocol.

Otherwise, application layer multicasts using a peer-to-peer routing algorithm have been proposed to ensure better scalability and handling of dynamic groups and network failures. In this paper, we propose a application layer multicast using the Chord routing algorithm with a location based service directory.

Table 4.1: Comparison with other research on application layer multicast

	End host discovery	Overlay network construction	multicast routing
Relay Cast	N/A. They assume that nodes know the rendezvous points.	Everyone maintains state for every others, the topology is optimized with Metric.	DVMRP or PIM-SM/CBT or Shortest widest path algorithm.
Select Cast	Everyone knows the root domain's router.	Hosts are organized in a domain hierarchy, and each domain has a set of attribute. every nodes maintain state of every others.	distribution tree rooted at the root domain's router. Senders specify the set of intended destination hosts through the use of SQL condition.
CAN based application layer multicast	A CAN has an associated DNS domain name, and that this resolves to the IP address of one or more CAN bootstrap nodes.	A node maintains the IP addresses of its neighbor that hold coordinate zones adjoining its own zone.	Flooding based on CAN routing table.
Bayeux	N/A. They assume that nodes know at least one node participating the network.	Each node maintains a number of neighbors with a common matching prefix.	They organize multicast receivers into a distribution tree routed at the source

Chapter 5

Design

In this chapter, we discuss the details of GGCAST. We describe the overview of our design first, an application layer multicast using a peer-to-peer lookup algorithm and combined with a location based service directory. Next, the details of each function is discussed.

5.1 Overview

Geographic Group Multicast (GGCAST) consists of two-level hierarchical peer-to-peer (P2P) overlay networks. The upper overlay network is a directory to find a multicast group. The lower overlay network constructs an application layer multicast (ALM) service.

The overview of GGCAST is shown in Figure 5.1. The circle represent a node in the multicast group. The double circle shows a server which organize the directory.

A GGCAST multicast group determined by geographic positions organizes an overlay network to construct a logical multicast tree. Nodes use this directory to find the pointer to the group in which they decide to join.

ALM typically consists of end host discovery, overlay network construction and multicast routing. In GGCAST, member of a multicast group construct a overlay network by using the Chord lookup algorithm [1], and a logical multicast tree for data delivery is built on the overlay network. A node finds a member of the overlay network as the initial pointer to the group from a location based service directory. For the detail of Chord lookup algorithm, refer to Appendix B.

Details of the functionalities are described in the following subsections.

- Location based service directory with peer-to-peer lookup protocol using consistent hashing

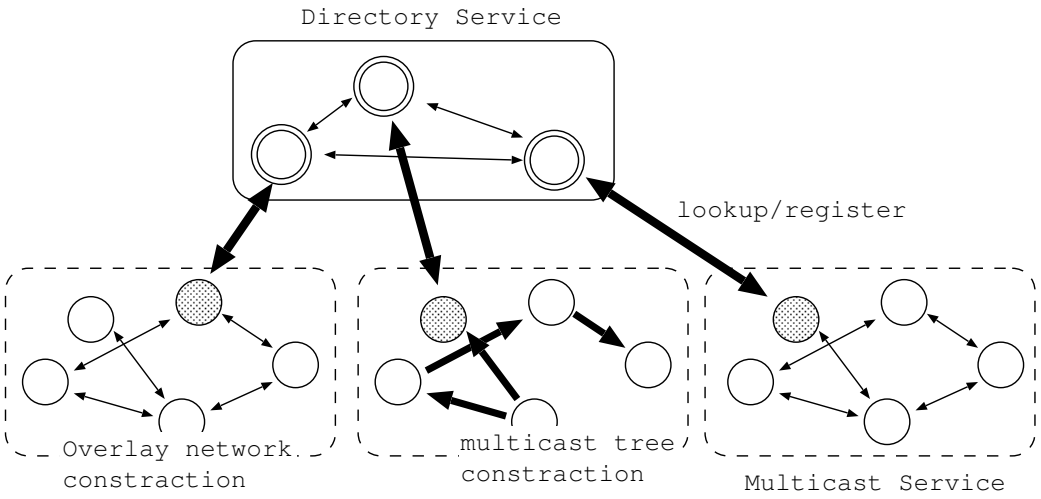


Figure 5.1: GGCAST using Hierarchical P2P System

- Overlay network construction using Chord lookup information
- Flooding on the overlay network

5.1.1 Location Based Service Directory

The location based service directory provides a lookup service, using geographic position (location) as search keys. The service returns pointers to a multicast groups (services) as values.

There may be a huge number of multicast groups in the world. Therefore, the system has to process a vast number of queries to registrate or find groups. The centralize system is typically weak at its root. In contrast, P2P systems has an advantage to balance load because P2P systems are distributed systems without any centralized control. This is the reason that the location based service directory consists of P2P systems.

Especially, GGCAST uses a P2P lookup algorithm which assigns keys to nodes with consistent hashing [39], [40], which has several desirable properties. The use of hash function will balance the load with high probability, therefore, all nodes receive roughly the same number of keys.

5.1.2 Overlay Network Construction

A member of a multicast group constructs an overlay network, and constructs a logical multicast tree on the overlay network. There is a possibility that

the overlay network breaks into fragments, when a member fails. Thus a fundamental proposition is how to build the overlay network which is robust in the face of partially incorrect routing information.

We focus on overlay networks designed for P2P lookup algorism. In particular we use Chord [1] algorithm. It is our conjecture that similar results would be obtained through other P2P lookup algorism such as CAN [36], Pastry [41] and Tapestry [42]. However Chord has the advantage that its correctness is robust in the face of partially incorrect routing information. Chord lookup algorithm has little influence even if node's failure occurs.

5.1.3 Multicast Routing

A logical multicast tree is being constructed on the overlay network. Because members of a multicast group change frequently, the topology of the overlay network may be subject to frequent changes. Thus, the fundamental proposition is how to build the logical multicast tree which is robust in the face of partially incorrect routing information. Since all members of the overlay network should receive multicast datagrams, a flooding scheme could be used. This is a similar situation in MANET.

Many protocols for MANET propose construction of routes reactively using flooding. The advantage hereof is that no prior assumption of the network topology is required in order to provide routing between any pair of nodes in the network. In mobile ad-hoc networks, where the topology may be subject to frequent changes, this is a particularly attractive property.

The total overhead incurred by a routing protocol consists of two elements: overhead in form of control traffic generated by the protocol, as well as overhead from data traffic forwarded through non-optimal routes. Such non-optimal routes brings a non-negligible overhead that is proportional to the data load of the network.

Any flooding method could be used on the overlay network of GGCAST. It should be chosen in consideration of the characteristic of a overlay network; e.g. a number of members, a rate of change of topology and density of a topology.

The characteristic of the overlay network is not studied yet in this research. We cannot evaluate the flooding schemes. Thus, the architectonics of the flooding scheme is out of scope in this thesis.

5.2 The Location Based Service Discovery Algorithm

5.2.1 Overview

The overview of the location based service directory is shows in Figure 5.2. There are server and client. Server application constructs a directory service using the Chord lookup algorithm. Client application is a resolver of the directory, ALM application on other hand, basically performs *lookup*() and *register*() on the mobile hosts.

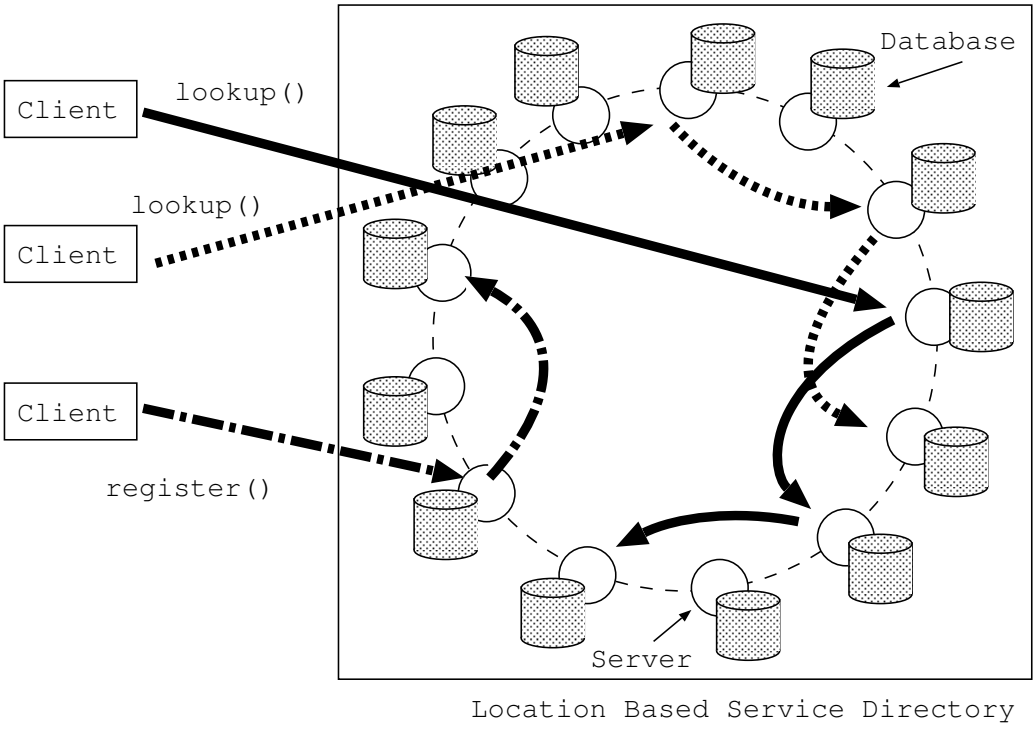


Figure 5.2: Overview of The Location Based Service Directory

The directory distribute the database across numerous servers. Clients search multicast groups with geographical position by using *lookup*(). When *lookup*() is called, the client sends a query to a server of the directory service. The client is assumed to know at least one of the servers in the directory. Server which receives the query searches the data location of the key, and sends a internal-registration query to the location. The lookup operation is performed according to an algorithm discussed in Section 5.2.5.

When a client first creates a new multicast group, *register*() will be called. Then, the client sends a registration query to the server within its region, lifetime and description.

5.2.2 Keys and Values

This directory provides a lookup service, with location information as search keys and it returns a pointer to a multicast group as values. The location information is simply specified by a certain specific geographic position.

The pointer is assumed to be a set of IP address and port number. If we use an application by a fixed port, it becomes impossible to use two or more location based multicast applications. When a geographic position is given as search key into the directory, the directory will return a list of pointers to multicast groups which included the geographic position.

The directory also reply with the region of the multicast group and the description of the group. For example, the region specified will be used to tell when to leave the multicast group as mobile node is moving out of the region. It is not necessary that a case that a node join all multicast group. The node makes judgments whether to join the group or not, by using the descriptions.

The summary is shown in Table 5.1.

Table 5.1: Key and Values of the location based service directory

Key	Values
a geographic position	a pointer to the multicast group geographical region description lifetime

5.2.3 Load Balancing

In order to lookup or register multicast groups, there are vast number of queries. Thus load balancing between servers organizing the directory service is important.

The consistent hash function assigns each node and key an m-bit identifier using SHA-1 . Since a server to store a key is determined according to the obtained identifier, a key is being distributed with high probability.

Likewise, the *lookup*() and *register*() queries are being distributed with high probability.

5.2.4 The method to specify a multicast region

In order to specify a multicast region, we need a common measuring method. GGCAST uses latitude and longitude for specifying a region. As explained in Chapter 2, mobile nodes are assumed into be capable of obtaining their geographic position. Any position sensing device maybe used since the position can be convertible into latitude and longitude.

Since the earth is a flat rotative ellipsoid in a direction, the length of latitude 1 minute becomes long as it goes north. Moreover, it becomes short as it goes to a pole, since the length of longitude 1 minute is the length of the width which cut the earth into wedges. Near Japan, 1 second in longitude is from 21 to 28m, and 1 second in latitude is about 31m. The surface in Japan is split by the square of about 30m around. The receiving region is expressed by this squares.

Figure 5.3 shows the example. A map is a matrix which records the multicast region of the multicast group. The multicast region are represented as black boxes in the figure. The x-axis represents longitude, the y-axis represents latitude. Each map has a identifier, and the length of the identifier is defined as m bits.

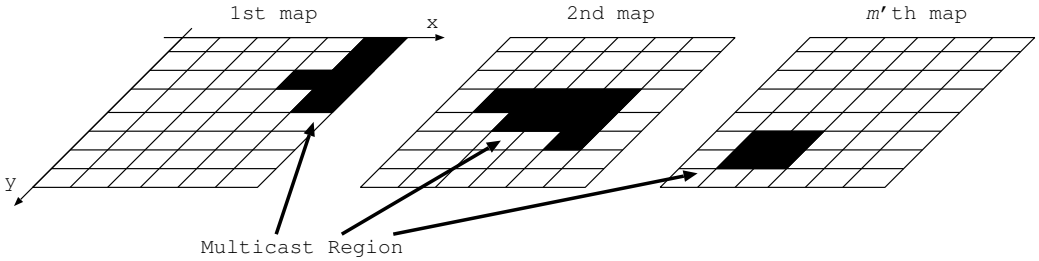


Figure 5.3: Maps which stores the location information of a multicast group

All receiving regions are shown in this way, and the directory stores this information. Since there are huge number of multicast groups all over the world, distributed processing is required. To be solved here is how can we distribute and search data efficiently.

5.2.5 Data Location and Lookup Algorithm

The lookup algorithm is an extension of the Chord algorithm. In this section, the method to store a key and the lookup algorithm is discussed.

Data Location

A map is divided into regions based on a geographic position information, and each piece is stored into an array as described in Figure 5.4. Here the arrays of (x1, y1), (x2, y2) and (x', y) are shown in the figure. When a group is registered and the identifier of the group is obtained by hashing a pointer to the multicast group, a bit corresponding to the identifier is set.

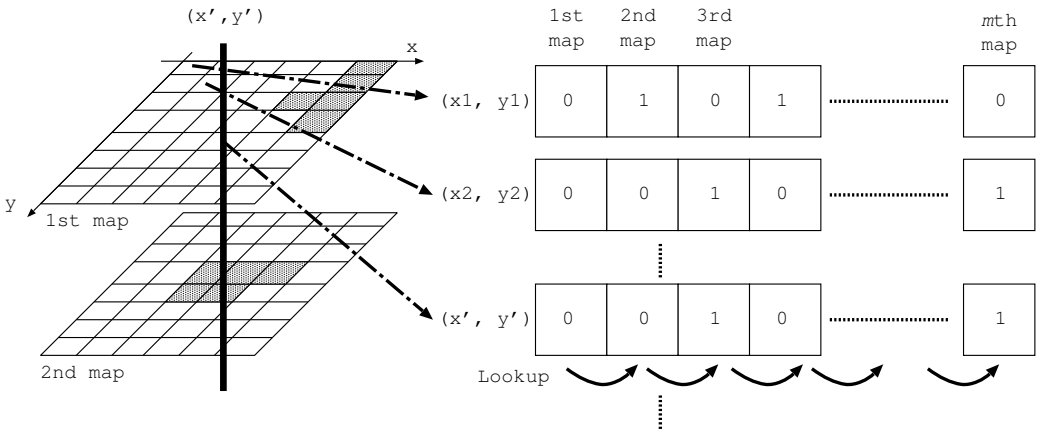


Figure 5.4: array for each point

Lookup: *lookup*()

When (x', y') is given as a search key, the directory first searches the array corresponding to the location given by the search key. Next, the identifier of map which is set to 1 is searched from the array.

The *lookup*() is then performed according to the algorithm described in Figure 5.5. When a *lookup*() is called, the corresponding array is searched using Chord lookup algorithm, which is shown as get_array_from_directory() in the figure. The system searches the registered identifiers of map from the array. Finally, pointers to the map will be returned.

```
// check the (x', y') bit of array
lookup (key) {
    array = get_array(key);

    for i = 1 upto m
        if ( array[i] )
            return a pointer to [i]'th map;
}

get_array(key) {
    array = get_array_from_directory(key);
    return array;
}
```

Figure 5.5: scalable search for the array

Registration: *register*()

The registration query includes the information discussed in Section 5.2.2. A server which received the query searches the data location from the multicast regions. When the location is determined, internal-registration queries are generated and transmitted. An identifier of a map is obtained by hashing the IP address and port number.

The case when a map includes (x1, y1), (x2, y2), (x3, y3), (x4, y4) and (x5, y5) is considered as an example. Since it is inefficient if the client sends 5 queries for the registration, a client sends a registration query to the directory. When the server receives the map information, the directory try to divide the map to five queries, (x1, y1), (x2, y2), (x3, y3), (x4, y4) and (x5, 5). Then the server sends 5 internal-registration queries.

5.3 Overlay Network Construction

5.3.1 Overview

To construct an overlay network, each nodes has to maintain information about other nodes, that is "routing" information. As described in Section 4, there is an approach which requires the information all other nodes to be maintained, and other approach which requires a number of other node information. Although maintaining information of all other nodes improve correctness, it also occurs less availability to reflect newly joined nodes as

well as node failures.

GGCAST uses the Chord lookup algorithm, which is a peer-to-peer DHT lookup algorithm to construct overlay network. Chord has the advantage, because its correctness is robust in the face of partially incorrect routing information.

5.3.2 Dynamic Operations

Here, we describe the pseudocode for joins and stabilization. When node n' first starts, it calls $n.join(n')$, where n' is any known Chord node, or $n.create()$ to create a new Chord network. The $join()$ function asks n' to find the immediate successor of n. By itself, join() does not make the rest of the network aware of n.

Every node runs *stabilize*() periodically to learn about newly joined nodes. Each time node n runs *stabilize*(), it asks its successor for the successor's predecessor p, and decides whether p should be n's successor instead. This would be the case if node p recently joined the system. In addition, *stabilize*() notifies node n's successor of n's existence, giving the successor the chance to change its predecessor to n. The successor does this only if it know of no closer predecessor than n.

Each node periodically calls fix fingers to make sure its finger table entries are correct; this is how new nodes initialize their finger tables, and it is how existing nodes incorporate new nodes into their finger tables. Each node also runs *check_predecessor* periodically, to clear the node's predecessor pointer if n.predecessor has failed; this allows it to accept a new predecessor in notify.

Details of the Chord lookup algorithm is mentioned in Appendix B.

5.3.3 Robustness

Figure 5.6 shows affection when a node has failed in Chord network. A circle is a node, an arrow indicate a virtual path between nodes.

When a node has failed, only $O(logN)$ fractions would happen, because each node maintains only $O(logN)$ routing information. Only one piece of information per node need be correct in order for Chord to guarantee correct routing of queries. That's why Chord has good performance despite continuous failure and joining of nodes.

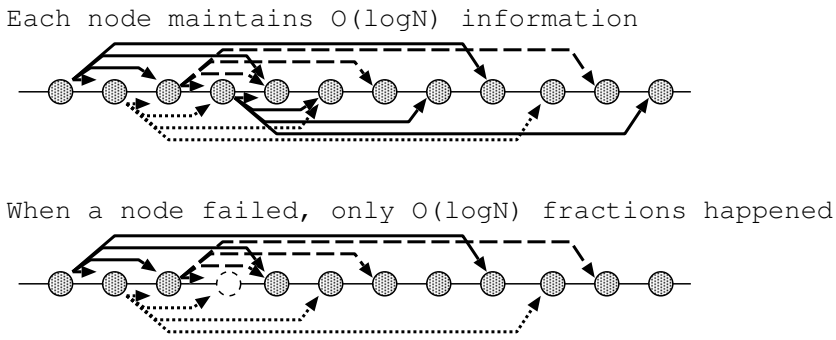


Figure 5.6: Affection when a node has failed

5.3.4 Multiple registration of pointers to the multicast group

In order to keep the connectivities between multicast groups and directory service, the multicast group keeps an ordered list of the candidates pointer to this group. This list is sent periodically to the directory service. When a pointer node fails, the first regular node in the list becomes the pointer node and registers to the directory service.

5.4 Summary

Our location based multicast model consists of hierarchical P2P overlay network, upper overlay network is a directory service to find a multicast group, lower one is overlay network for application layer multicast. Both overlay networks are constructed using Chord lookup algorithm.

In our model, the world is divided into a grid based on longitude and latitude. A receiving region of multicast is shown as the grid. These information is divided for every grip and registered to the directory. Meanwhile, a logical trees for multicasting is built using Chord routing tables. The forwarding algorithm send the message to each node in the finger table, but adjusts the region of multicast before sending to limit the number of duplicate messages.

Chapter 6

Implementation

In this chapter, we describe the implementation of GGCAST. We describe the overview of our implementation first. Next, we explain the data structures and the application libraries used in GGCAST. Finally, we introduce the user interfaces.

6.1 Overview

Our implementation consists of following five targets.

- The Chord library: This provides the basic functionality of Chord. It maintains routing tables and is able to evaluate find_successor.
- The Directory library: This provides the DHash object which provides the directory service in a Chord network.
- The Directory server daemon (dsd): dsd is the daemon which implements the Chord and directory protocols.
- The Mcast library: This library provides the application layer multicast object.
- The Mcast server daemon (msd): msd is the daemon which provides multicast service.

Figure 6.1 shows the overview of our implementation. The rectangle in the figure represents a modules, there are three modules, (1) overlay network module, (2) directory module and (3) mcast modules. The dsd consists of (1) and (2), and the msd consists of (1), (2) and (3).

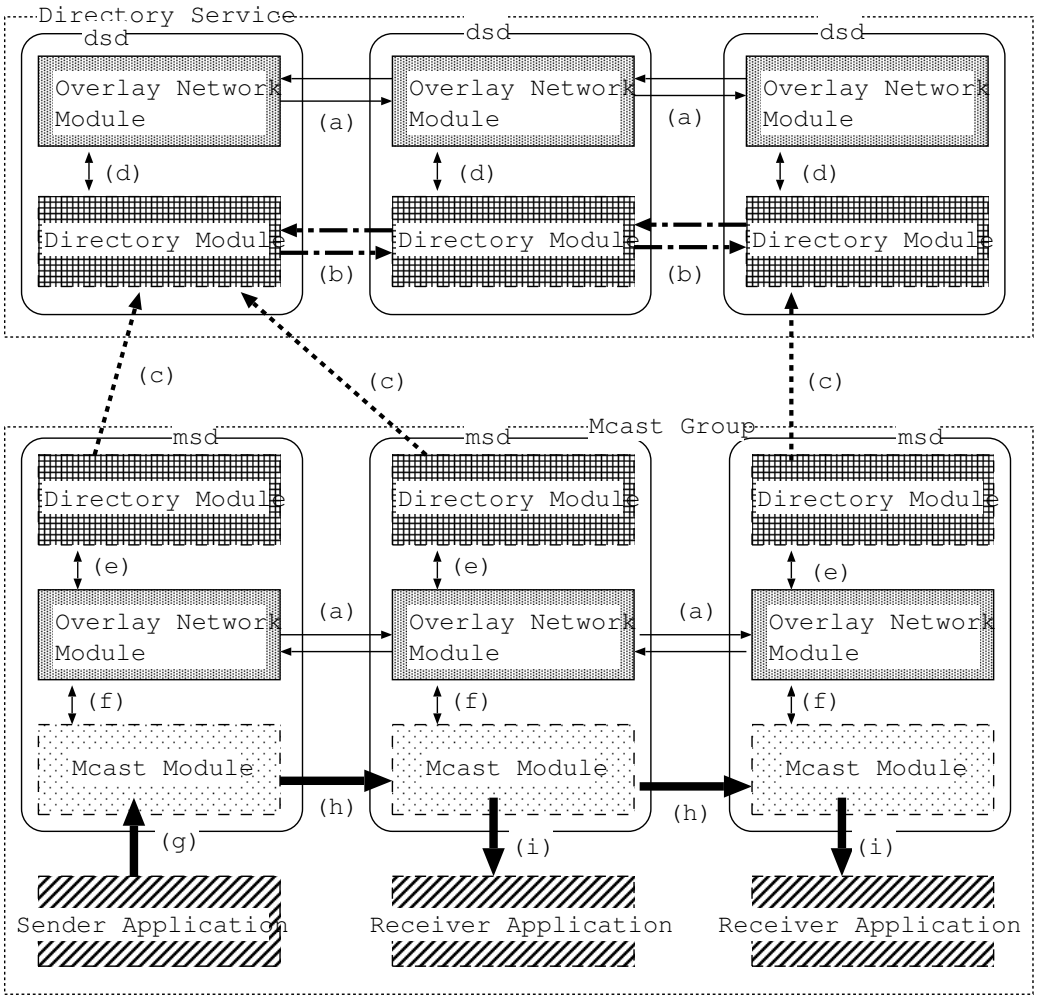


Figure 6.1: GGCAST Implementation Modules

(a) The overlay network module exchanges the routing information between each other. (b) The directory modules communicate each other and construct the directory service. (c) A resolver included at the directory module sends a lookup or registration query, the directory modules process the request. At the those operations, routing information maintaining by overlay network is used via the Chord library (d).

(e) The lookup and registration operations are performed via the directory library. The mcast module uses the routing information via the Chord library (f). When a sender application sends data grams (g), the data grams are transmitted according to the routing information (h) and Receiver application receives the data gram (i).

6.2 Library

Table 6.1 shows the Chord library, an interface to use overlay network modules.

Table 6.1: The Chord Library

Name	Description
create()	Create a new Chord ring
join(struct sockaddr *n)	Join a Chord ring containing n
stabilize()	Called periodically. verifies n's immediate successor, and tells the successor about n, notify(n).
notify(struct sockaddr *n)	n thinks it might be our predecessor.
fix_fingers()	Called periodically. refreshes finger table entries. next stores the index of the next finger to fix.
check_predecessor()	Called periodically. check whether predecessor has failed.
find_successor(struct id *id)	Ask node n to find the successor of id.

Table 6.2 shows the directory library, an interface to use the directory service.

Table 6.3 shows the multicast library, an interface to use application layer multicast.

Table 6.2: The Directory Library

Name	Description
lookup(struct position *xy)	Ask node n to find multicast groups.
add(struct range *xyxy, struct sockaddr *pointer, int lifetime, void *description, size_t length)	Register a new multicast group.

Table 6.3: The Mcast Library

Name	Description
msend(int s, const void *msg, size_t len, int flags)	transmit a message to another nodes in a group via mcast server.
mrecv(int s, void *buf, size_t len, int flags)	receive message from mcast server.

6.3 Variables

NUM_BITS is identifier length, and AVG_PKT_DELAY is average latency of a packet between any two nodes in the system, the latency is exponentially distributed. RROC_REQ_PERIOD is average interval between two invocation of the *process_request*() procedure. The duration of this interval is uniformally distributed in [0.5*PROC_REC_PERIOD, 1.5*PROC_REC_PERIOD). STABILIZE_RERIOD is average interval between two invocations of the *stabilization*() procedure. The duration of this interval is uniformly distributed in [.5*STABILIZDE_REC_PERIOD, 1.5*STABILIZE_PERIOD). TIME_OUT is timeout to detect a node failure. A node eliminates a finger from its finger table if it doesn't hear an answer within TIME_OUT ms after it has sent a query. HASH_SIZE is size of hash table used to maintain all nodes. DEFAULT_NUM_FINGERS is finger table size. DEFAULT_NUM_SUCCS is number of successors maintained by each node; these are the first NUM_SUCC nodes in the finger table.
DEFAULT_MAX_LOCATION_CACHE is size of cache for NUM_SUCCS and NUM_FINGER.

6.4 User interface

In this section, we describe the user interface of GGCAST. This application consists of two parts, Directory Service (dsd) and Multicast Service (msd).

6.4.1 Directory Service

The directory service takes a group of confusing options on the command line. The directory service daemon is named *dsd*. Run it with no options just to make sure the build went right (Figure 6.2) .

```
% ./dsd
Usage: dsd -d -j hostname:port -p port ...
```

Figure 6.2: Running Directory Service

When a node joins a Directory Service Network, based on Chord routing protocol, it must contact a well-known, or bootstrap node. Any node, once running, can serve as a bootstrap node. However, the first node run as part of a Chord network must bootstrap itself. All this really means is that when starting the node one must specify a port (rather than letting Chord choose one) and set the bootstrap node to itself.

Figure 6.3 illustrates this process as it might be run on a host named wanwan.sfc.wide.ad.jp (the command line options will be described in detail in a later section). The important fact to note is that this node has specified a port number and specified itself (wanwan.sfc.wide.ad.jp:10000) as the bootstrap node.

When starting additional (non-bootstrap nodes) the port argument is optional, and the address of a running node should be specified after the -j parameter. If no port is specified, dsd will choose an unused port (Figure 6.4).

6.4.2 Multicast Service

The multicast service takes a group of confusing options on the command line. The directory service daemon is named *msd*. Run it with no options just to make sure the build went right 6.5.

When a node joins a multicast group network, based on Chord routing protocol, it must contact a well-known, or bootstrap node. Any node, once running, can serve as a bootstrap node. However, the first node run as part

```
% ./dsd -j wanwan.sfc.wide.ad.jp:10000 -p 10000
Chord: running on 18.26.4.29:10000
init_chordID: my address: 18.26.4.29.10000.0
 1004828619:637146 myID is
                   caa42d5de473ac83e5be5cd96cdcaa6f7b85da56
dsd: insert: caa42d5de473ac83e5be5cd96cdcaa6f7b85da56
 1004828622:099189 stabilize:
                   caa42d5de473ac83e5be5cd96cdcaa6f7b85da56
                   stable! with estimate # nodes 1
```

Figure 6.3: Starting a bootstrap node

```
% ./dsd -j wanwan.sfc.wide.ad.jp:10000
init_chordID: my address: 18.26.4.29.2496.0
 1004829020:385471 myID is
                   b678329a53d1a0a3e54fcd7a46d0d09d097fee34
dsd: insert: b678329a53d1a0a3e54fcd7a46d0d09d097fee34
 1004829027:570355 stabilize:
                   b678329a53d1a0a3e54fcd7a46d0d09d097fee34
                   stable! with estimate # nodes 12
```

Figure 6.4: Starting additional nodes

```
% ./msd
Usage: msd -j hostname:port -p port -s hostname:port -r range
           -m description -l lifetime ...
```

Figure 6.5: Running multicast server

of a Chord network must bootstrap itself. All this really means is that when starting the node, one must specify a port (rather than letting Chord choose one) and set the bootstrap node to itself. The first node must register the new multicast group to the directory service, it must contact a well-known node which serves Directory Service.

Figure 6.6 illustrates this process as it might be run on a host named wanwan.sfc.wide.ad.jp (the command line options will be described in detail in a later section). The important fact to note is that this node has specified a port number and specified itself (wanwan.sfc.wide.ad.jp:10000) as the bootstrap node.

```
% ./msd -j modena.sfc.wide.ad.jp -p 55555
        -s wanwan.sfc.wide.ad.jp:10000
        -r 123124,451342,123131,451542
        -m ‘‘traffic jam’’ -l 3600
chord: running on 87.65.43.21:55555
init_chordID: my address: 87.65.43.21.55555.0
 10048286120:756387 myID is
                    0d7608a18e27aa936b5302cd2ac8723f2fb14683
msd: create multicast socket as /tmp/msd-socket.0
msd: insert: 0d7608a18e27aa936b5302cd2ac8723f2fb14683
 10048286125:127934 stabilize:
                    0d7608a18e27aa936b5302cd2ac8723f2fb14683
                    stable! with estimete # nodes 1
```

Figure 6.6: Running multicast server

When a node search a multicast service at a certain point, it must contact a well-known directory service node. The directory service will return a list of multicast group. Figure 6.7 shows this process.

When starting additional (non-bootstrap nodes) the port argument is optional, and the address of a running node should be specified after the -j parameter. If no port is specified, msd will choose an unused port (Figure 6.8). msd creates socket, an endpoint for multicasting.

Figure 6.9 shows the user interface to send datagrams. The socket is connected to the socket of receiver. In the figure, the characters is inputted.

Figure 6.10 shows the user interface to receive datagrams. When a receiver application opens the socket, the receiver will receive the characters inputted at the sender.

```
% ./msd -s wanwan.sfc.wide.ad.jp:10000 -r 123130,451382
Contacting wanwan.sfc.wide.ad.jp:10000 at 123130,451382
..........!
Got the answer:
  hostname:port                 description    lifetime(sec)
                   range
  ------------------------------------------------------------
  modena.sfc.wide.ad.jp:10000   traffic jam    3600
                   (123124,451342)-(123131,451542)
```

Figure 6.7: Finding multicast group

```
% ./msd -j modena.sfc.wide.ad.jp:10000
init_chordID: my address: 12.34.56.78.2498.0
 10048286139:432409 myID is
                    ccbe9a171b14a12d9c53d91d588903c588366990

msd: create multicast socket as /tmp/msd-socket.0

msd: insert: ccbe9a171b14a12d9c53d91d588903c588366990
 10048286149:412349 stabilize:
                    ccbe9a171b14a12d9c53d91d588903c588366990
                    stable! with estimete # nodes 12
```

Figure 6.8: Joining a multicast group

```
% echo ‘‘this is test message.’’ > test
% cat test > /tmp/msd-socket.0
```

Figure 6.9: Sending datagram to the multicast group

```
% cat /tmp/msd-socket.0
this is test message.
```

Figure 6.10: Receiving datagram

Chapter 7

Evaluation

GGCAST is evaluated as an application related to Intelligent Transportation System (ITS), and it has been confirmed that our implementation can provide enough capability for use with in the ITS. From our evaluation, the directory service was found to be realizable with about 4000 servers, each server processing about 2000 query per seconds. It was also found that over 99% of the members are able to receive multicast packets even under conditions were a node joins or leaves a multicast group every 0.9 seconds.

7.1 Group Search Performance

In this section, we evaluate the performance of searches for multicast groups. In doing so, we compare the Chord algorithm, a peer-to peer DHT lookup algorithm, with a distribution tree model, both as a method for the directory service. Figure 1 shows the search flows for both the Chord ring model and the distribution tree model.

Both finding performances are described as Formula 7.1. T_{search} is the search time. H is expected number of hop to complete the lookup. RTT is round trip time of between servers. LP is time of internal processing at a server for the lookup.

$$T_{search} = \sum_{k=1}^{H} (RTT_k + LP_k) \tag{7.1}$$

The LP is explained as Formula 7.2. T_{lookup} is the time required to lookup the entry from the database. $T_{QueryProc}$ is the time required to process the lookup query.

$$LP = T_{lookup} + T_{QueryProc} \tag{7.2}$$

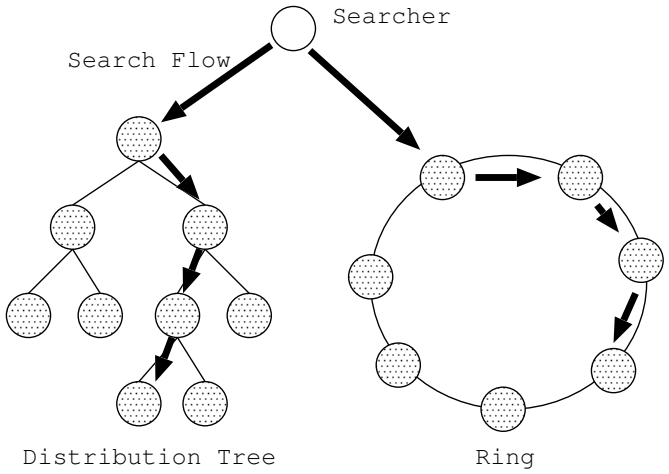


Figure 7.1: Search flow by using distribution tree model and ring model

7.1.1 Experiment 1: Database Lookup

In order to evaluate T_{lookup}, the following experiment was conducted using our implementation. Figure 7.2 shows the experiment.

(1) In the first of the experiment, Query Generator generates random queries to register groups. The number of queries is set to 1000 to 10000 by 1000s and from 10000 by multiples of two up to 1280000. IP address, port number, and the multicast region are generated by using the *rand*() function of Standard C Library. The seed for a sequence of pseudo-random numbers is reset at regular intervals. When a multicast group is meshed in more than one cell, registration are processed for each cell. The lifetime of the multicast groups is set to 300 seconds and the description is set as "TRAFFIC JAM". This queries are directly inputted to the dsd program without a network.

(2) Second, the queries to find a group which selected at random. The number of query is 1000. The queries were also generated by the *rand*() function. For this experiment, the time required to lookup an entry were measured by using the CPU clock conter.

Table 7.1 shows the specification of the PC used for this experiment.

The experiment was performed 1000 times. The result is shown in Figure 7.3. The graph shows the processing time per a lookup in label y and the number of the entries held at the lookup in label x. The mark X represents the average.

The average lookup times was 150 usec. This has no influence in the performance e even if the number of entries increases.

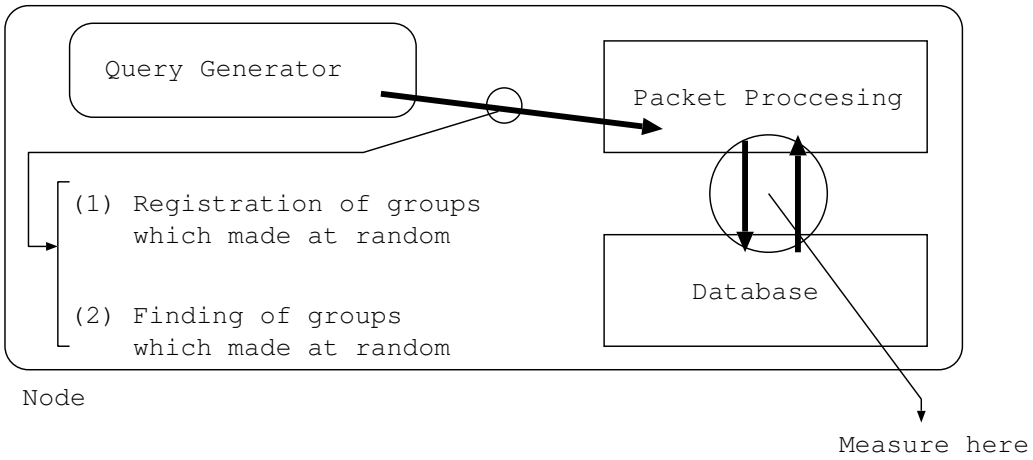


Figure 7.2: Experiment to measure the lookup performance

Table 7.1: Specification of experiment PC

CPU	Intel Pentium III Processor 846MHz
Memory	512MB RAM
HDD	IDE 40GB 4200rpm
Network	Intel PRO/100+ MiniPCI
OS	NetBSD 1.6.1-Release

7.1.2 Experiment 2: Query Processing

The following experiment was conducted in order to evaluate $T_{QueryProc}$, using our implementation. Figure 7.4 shows the experiment. (1) Packet Generator generates random queries to register random groups. (2) Packet Generator generates random queries to search for groups. The number of queries for each experiment is 1000. The queries are transmitted by a unit of 1 time. The processing time of lookup from database is deducted. The experiment was performed 1000 times.

Table 7.2 shows the result. To process a finding query, it takes 524 usecs on average.

7.1.3 Conclusion

As above these results of the two experiments, Formula 7.3 is verified.

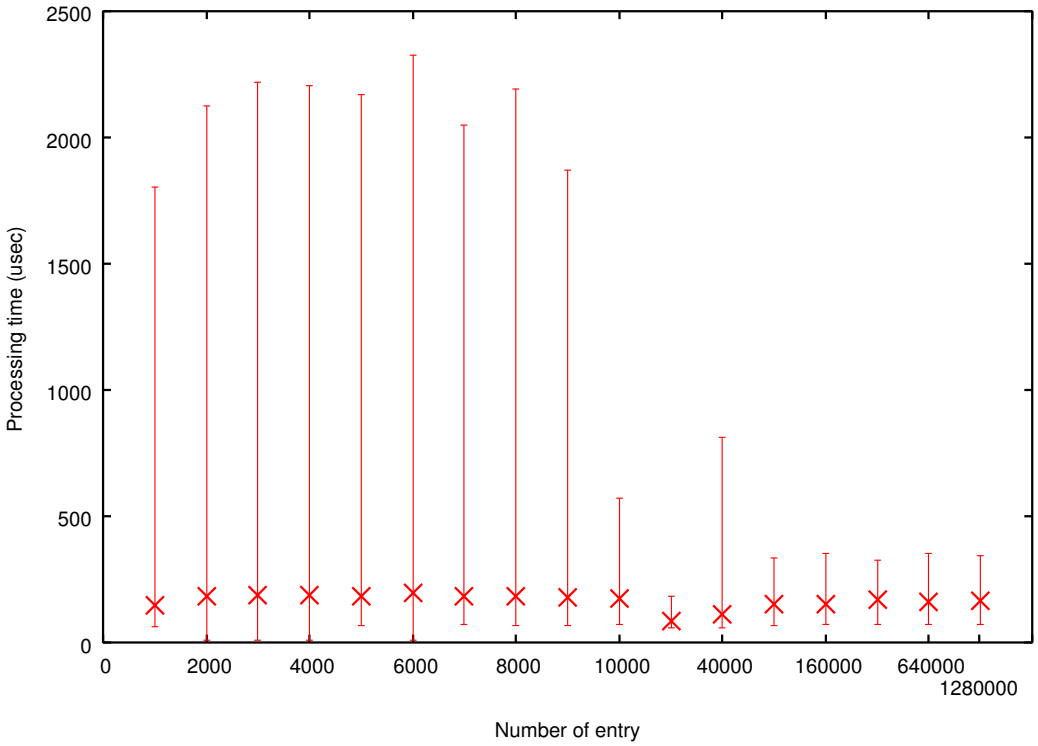


Figure 7.3: Lookup performance at a server

$$RTT >>> LP \tag{7.3}$$

Therefore, T_{search} is greatly depending to H, which is the number of hop.

In a distribution tree model, the hop is number of level. For example, if the levels are root level, degree level and minute level, it takes 4 hops. In contrast, the path length is expected 2 - 9 hops in our model; this is the result shown in [1].

A distribution tree model and our model has the similar performance. Our proposal is excellent in the point which load distribution is equally carried out among the servers.

7.2 Required Number of Directory Servers

In this section, we evaluate the number of servers required archive the GGCAST directory service.

The total number of server is described as Formula 7.4. N_{server} is total

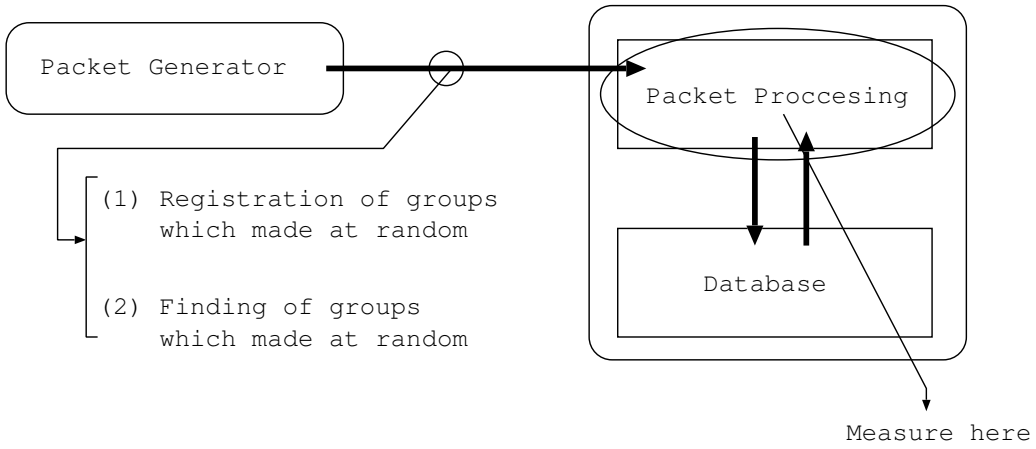


Figure 7.4: Experiment to measure the query processing performance

Table 7.2: A query processing performance at a server

Average	Max	Min	Stdev
524 usec	673 usec	375 usec	211 usec

number of server. C_{total}, Total processing cost is the total amount of the lookup/registration queries and the management cost. The management costs are caused by operations described in Section 5.3.2. The costs are expected not big than the costs caused by processing the queries, because we assume that those sever are stable. P_{server} is performance per one server.

$$N_{server} = \frac{C_{total}}{P_{server}} \tag{7.4}$$

For example, when each person make one group, there are 5 billon multicast groups in the world. From the estimation described as Table 3.1, a node sends the query every 65 seconds. Then, $5billon/65seconds = 76923075.9$, the directory should process about 80 million query per second (qps).

The performance of the server is measured in the same environment as the one in Figure 7.4. The expected number of entries is registered beforehand. The search queries and registration queries were generated randomly.

Figure 7.5 shows the result. The graph shows the processing performance (queries per seconds) in label y, and the number of the entries already held in label x.

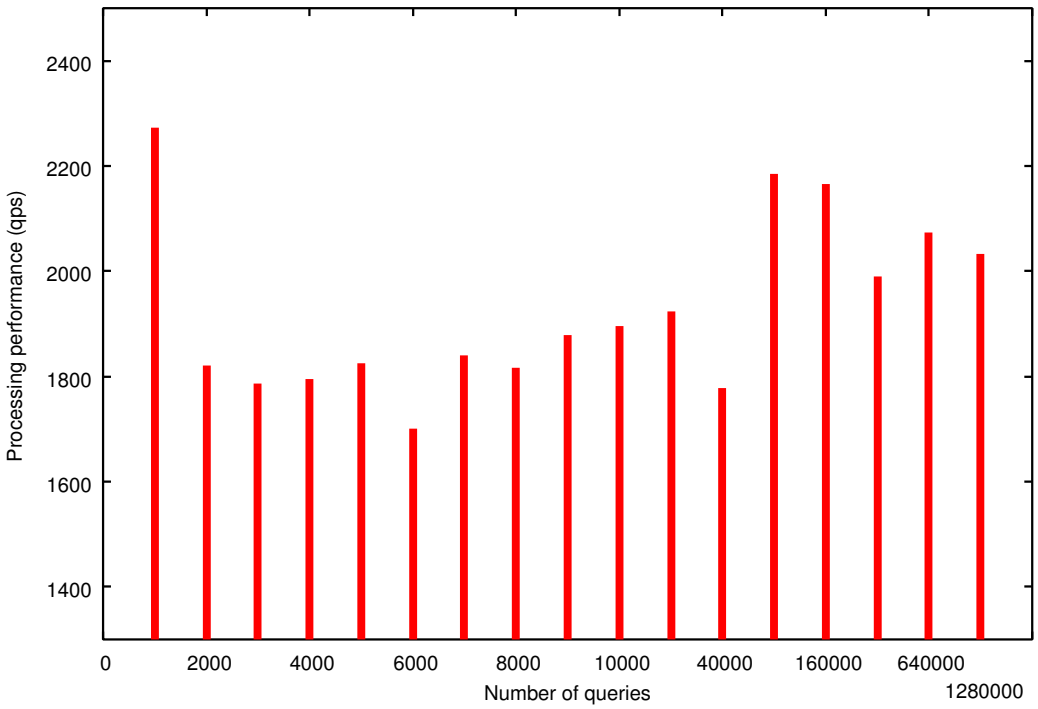


Figure 7.5: Processing Performance at a server

From the graph, we can see that the server has the performance to server about 2000 qps. As above preliminary estimate, there is about about 80 million qps totally. Therefore, we can estimate that about 4000 servers is needed to provide this directory services.

7.3 Modeling of Dynamic Group Memberships

In order to evaluate the robustness of GGCAST, dynamic group memberships should be modeling. In this research, the memberships consists of tree phases, shown as Table 7.3.

Phase 1 can be shown as Formula 7.5. G is growth rate, and t is the time past after the group is created.

$$N_{member} = G * t \quad (1 < t < T) \tag{7.5}$$

Phase 2 can be shown as Formula 7.6. N is the number of members at time t. Because the number of members fluctuates at around $G * T$, it's convenient to use the average of the queries of the deviations about $G * T$.

Table 7.3: Dynamic Group Memberships

Phase 1: Growth period	The number of members grows up at a increasing rate until reach a saturation point; the number of members in a multicast range has upper threshold.
Phase 2: Maturation period	The number of members may fluctuate around a point of saturation. Member comes in and out at a rate
Phase 3: Decline period	The number of members becomes fewer.

The average is taken over a time interval $T - T'$ that's much longer than the period of the fluctuations.

$$N_{member} = \frac{1}{T' - T} \int_0^{T'-T} (N - G * T)^2 dt \quad (T < t < T') \tag{7.6}$$

Formula 7.7 shows Phase 3 mathematically.

$$N_{member} = G * T - G * (t - T) \quad (T' < t < T'') \tag{7.7}$$

7.4 Robustness of the Overlay Network

The robustness of the overlay network is verified according to the dynamic group memberships model described in Section 7.3. The robustness indicates the number of whole connections which still being maintained when members change.

7.4.1 Experiment 3: Overlay Network Construction

In order to figure out the number of connections, Phase 1 and Phase 2 as described in Table 7.3 were simulated. The saturation point grants 350 nodes.

- Case 1: The middle of Phase1. About 150 nodes join to the multicast group. A node joins per 0.9 (Case 1-1), 1.8 (Case 1-2), 3.6 (Case 1-3), 7.2 (Case 1-4), 14.4 (Case 1-5), 28.8 (Case 1-6) seconds. Case 1-7 is in the best state.
- Case 2: The end of Phase1. A node joins per 0.9 (Case 2-1), 1.8 (Case 2-2), 3.6 (Case 2-3), 7.2 (Case 2-4), 14.4 (Case 2-5), 28.8 (Case 2-6) seconds. Case 2-7 is in the best state.

- Case 3: The end of Phase2. A node joins per 0.9 (Case 3-1), 1.8 (Case 3-2), 3.6 (Case 3-3), 7.2 (Case 3-4), 14.4 (Case 3-5), 28.8 (Case 3-6) seconds. Case 3-7 is in the best state.

Table 7.4 shows the environment variables in the simulation. The variables are defined in Section 6.3.

Table 7.4: Environment variables

NUM_BITS	24
AVG_PKT_DELAY	50 ms
PROC_REQ_PERIOD	500 ms
STABILIZE_PERIOD	30000 ms
TIME_OUT	500 ms
HASH_SIZE	10000
DEFAULT_NUM_FINGERS	20
DEFAULT_NUM_SUCCS	5
DEFAULT_MAX_LOCATION_CACHE	25

This simulation was performed 100 times for each scenario, changing the speed of a random generator for each test.

7.4.2 Topology

Figure 7.6 shows the topology of Case 1-2. The box represents a node, the number is node's id. The line represents a virtual connection between two nodes. We define the following cases;

Virtual connections between each node construct a fully meshed network as shown in Figure 7.6. The network constructed withholds the robustness against dynamic changes in number and of location of the nodes inside.

7.4.3 The number of connections

Figure 7.7 shows the number of virtual connections. The graph shows the number of connections in label y and the testing cases in label x. The X mark represents the average. The maximum of a line shows maximum values and the minimum shows the minimum value.

Although the fluctuation in number of connection is wide, minimum 1 and maximum 25, nodes hold, in average, a number of connections: e.g. 19 connections at Case 2-1. Therefore, the communication scheme using overlay

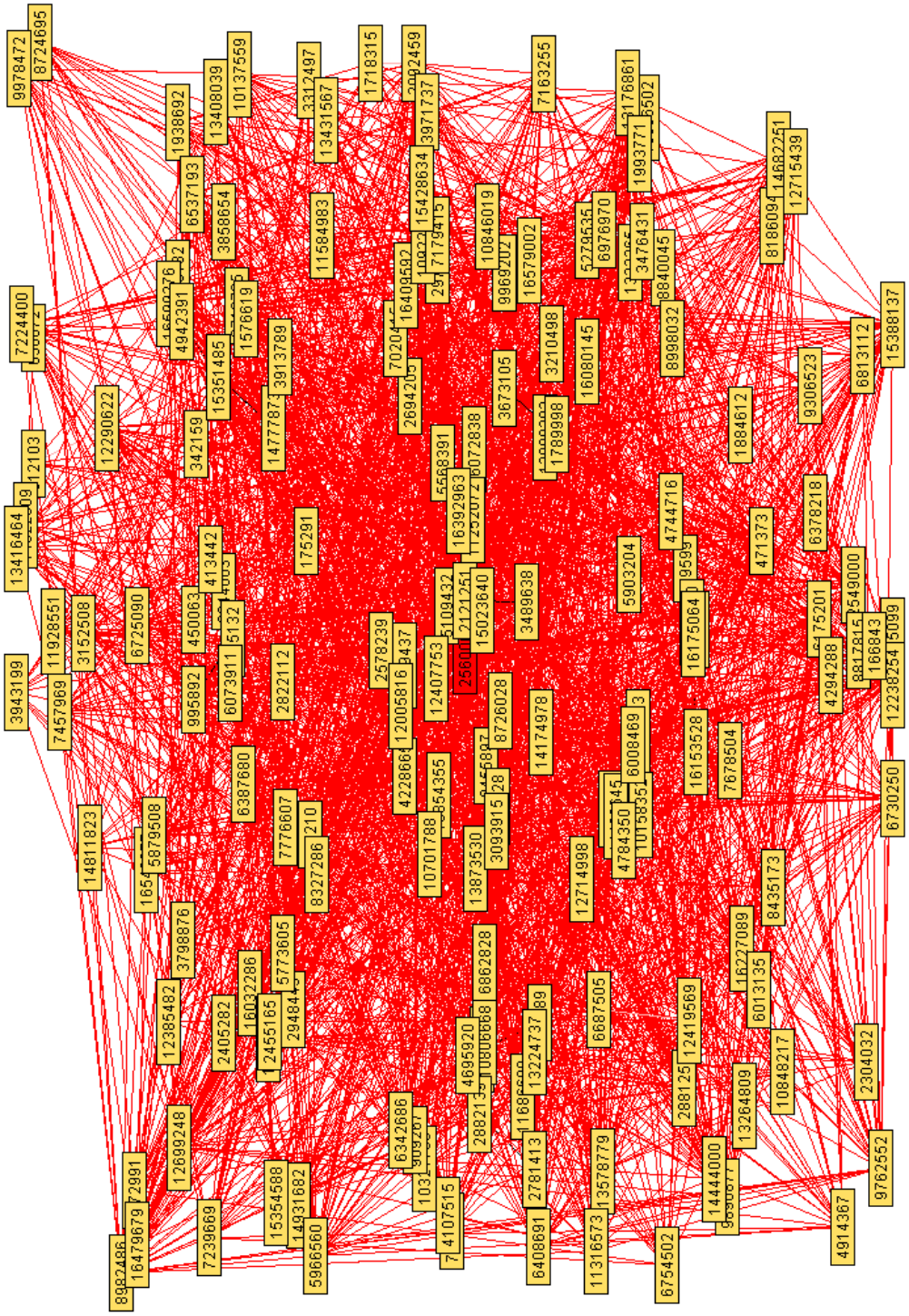


Figure 7.6: Topology of a overlay network

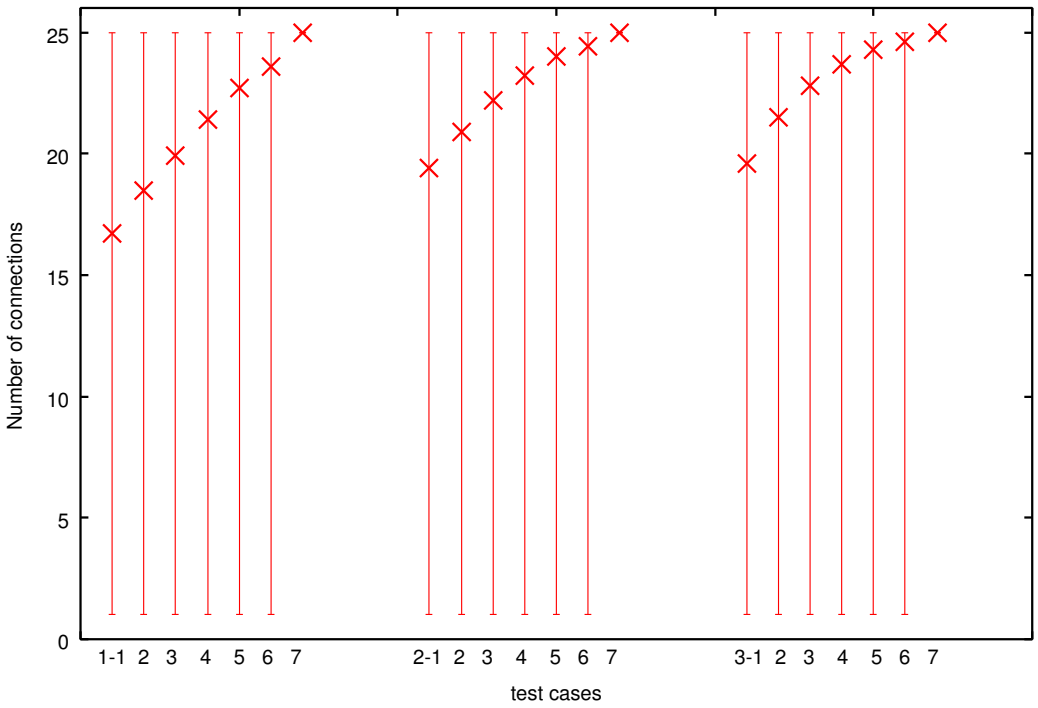


Figure 7.7: Number of virtual connections

network can be considered robust in the situation where the membership of multicast group changes dynamically.

Figure 7.8 shows the rate of reachable nodes. The number of hops represents in label x and percentage of reachable nodes in label y.

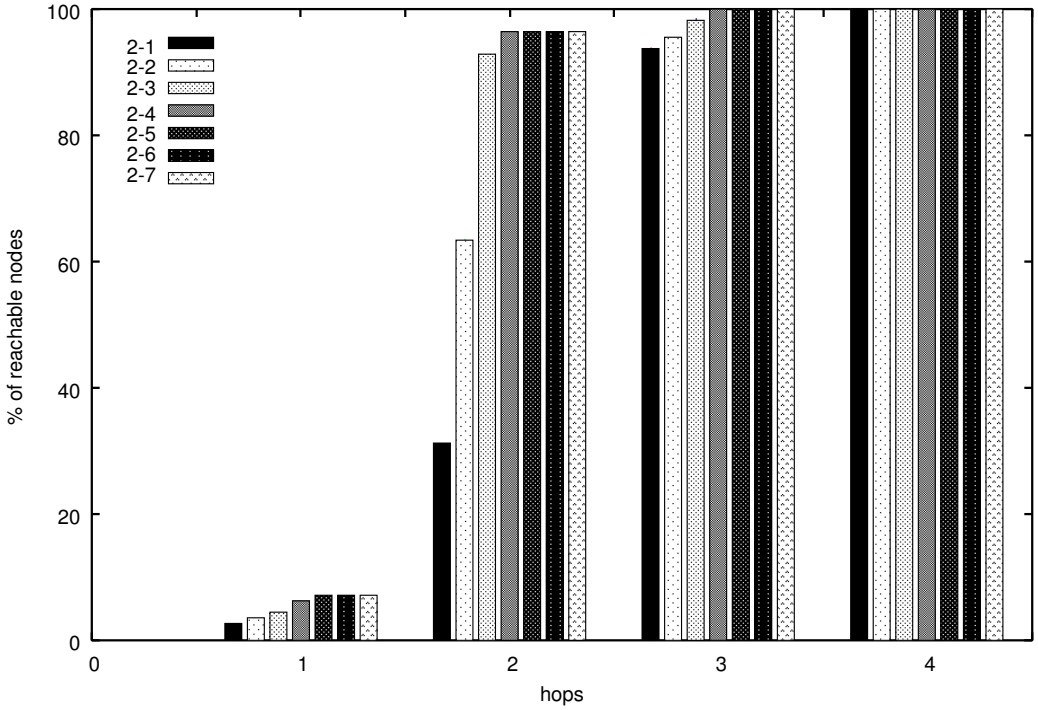


Figure 7.8: Rate of reachable nodes

Over 99 % of nodes could receive a datagram in only 3 hops. From this evaluation, GGCAST multicasting scheme can be considered practical since the scheme provides connectivity to 360 nodes within 3 hops range.

Table 7.5 shows the number and the percentage of unnecessary packets in flooding. The table shows the result of Case 2.

From the table, it can be seen that the percentage of unnecessary packets are about 40at nodes which are 2 hops away from the sender. If the range is extended to 3 hops, the percentage of unnecessary packets increase to 99

The result indicates that there's a need for better flooding scheme to realize efficient multicast communication using overlay network.

Table 7.5: Unnecessary Packets

Case	2hops (percentage)	3hops (percentage)
2-1	66 (40)	1857 (89)
2-2	69 (24)	4694 (97)
2-3	125 (29)	7466 (99)
2-4	216 (40)	8141 (99)
2-5	236 (42)	8499 (99)
2-6	255 (44)	8866 (99)

7.5 Summary

GGCAST has been implemented as an ITS-related application for evaluation. It has been confirmed that our directory server can process about 2,000 queries per second. It is estimated that about 4,000 servers would be sufficient to provide a globally available version of this directory service. Other results include that even in a case in which membership changes every 0.9 seconds, over 99members can receive the multicast delivery of information in only 3 hops by simple flooding method.

Chapter 8

Conclusion and Future Works

8.1 Conclusion

Geographic Grouping Multicast (GGCAST), newly introduced in this research, is one of the one-to-many communication methods in which receivers are determined by their geographic positions. Applications of GGCAST include dissemination of traffic information within relevant locations, such as passing the cause of a traffic jam to cars running behind. The objective of this research is to propose the architecture of GGCAST for a mobile ubiquitous computing environment.

Since, in GGCAST, a multicast group is defined by geographic domain, its membership changes dynamically, and it is difficult to keep a logical multicast tree for data delivery. The fundamental propositions of this research are dynamic group membership and efficient data transmission in such a multicast group.

In order to achieve dynamic group membership, following approaches are proposed; Multicast grouping with geographic information, Join/Leave model and End node organization. In addition, to achieve efficient data transmission in a multicast group, following approaches are proposed; End node organization, Overlay network which has robustness when a node fails, and Less packet duplication.

In this paper, GGCAST is proposed as an application layer multicast (ALM) using a peer-to-peer (P2P) lookup algorithm with a location based service directory. In the targeted environment, nodes are freely moving over several ISPs. This makes ALM to be more suitable for the situation because it doesn't need a network layer protocol support. In GGCAST, members of a multicast group construct a overlay network using routing information of a P2P lookup algorithm, and a logical multicast tree is built on the overlay

network. A member is discovered with the location based service directory as an initial node of the overlay network. The overlay network is using Chord [1], a P2P lookup algorithm. The routing information of Chord has little influence on the overlay network even when a node fails. Because membership of GGCAST changes frequently, the robustness is important.

GGCAST has been implemented as an ITS-related application for evaluation. It has been confirmed that our directory server can process about 2,000 queries per second. It is estimated that about 4,000 servers would be sufficient to provide a globally available version of this directory service. Other results include that even in a case in which membership changes every 0.9 seconds, over 99% of the members can receive the multicast delivery of information in only 3 hops by simple flooding method.

We conclude that GGCAST realizes a practical location-based multicast architecture.

8.2 Status and Future Directions

The directory service balances the load on the servers equally with high probability by using consistent hashing, and a query may be sent to the server which is far from the client. As described in Section 7.3, the performance of the directory service is greatly depending on the number of hops and the round trip time between servers. Thus, the performance degradation is caused in such situation. [43] has proposed an effective P2P distributed content location mechanism using content-relationship. In order to realize the locality of contents, their system assigns a hierarchical identifier according to content-relationship. The realization of contents locality may provide performance improvements.

As discussed in Section 5.1.3, a flooding method should be chosen in consideration of the characteristic of the overlay network. the architectonics of the flooding scheme is out of scope in this thesis. This will be studied further as one of the next steps in this research.

Appendix A

Current Position Sensing Technologies

Table A.1, Table A.2 and Table A.3 shows current major position sensing technologies and its characteristics [25].

A position system can provide two kind of information, physical and symbolic. GPS provides physical positions. For example, our campus, Shonan Fujisawa Campus of Keio University, is located on $35^o20'38''N$ by $139^o29'7''$. In contrast, symbolic position encompasses abstract ideas of where something is: in the kitchen, in Kalamazoo, next to a mailbox, on a train approaching Denver.

An absolute position system uses a shared reference grid for all located objects. For example, all GPS receivers use latitude, longitude, and altitude - or their equivalents, such as Universal Transverse Mercator coordinates - for reporting location. Two GPS receivers placed at the same position will report equivalent position readings, and $47^o39'17''$ N by $122^o18'23''$ W refers to the same place regardless of GPS receiver.

A position system should report positions accurately and consistently from measurement to measurement. Some inexpensive GPS receivers can locate positions to within 10 meters for approximately 95 percent of measurements. More expensive differential units usually do much better, reaching 1- to 3-meter accuracies 99 percent of the time. There distances denote the accuracy, or grain size, of the position information GPS can provide. The percentages denote precision, or how often we can expect to get that accuracy.

A position sensing system may be able to locate objects worldwide, within a metropolitan area, throughout a campus, in a particular building, or within a single room. Further, the number of objects the system can locate with a certain amount of infrastructure or over a given time may be limited. For

example, GPS can serve an unlimited number of receivers worldwide using 24 satellites plus three redundant backups. On the other hand, some electronic tag readers can not read any tag if more the one is within region.

For applications that need to recognize or classify located objects to take a specific action based on their location, an automatic identification mechanism is needed. For example, a modern airport baggage handling system needs to automatically route outbound and inbound luggage to the correct flight or claim carousel. A proximity-location system consisting of tag scanners installed at key locations along the automatic baggage conveyers makes recognition a simple matter of printing the appropriate destination codes on the adhesive luggage check stickers. In contrast, GPS satellites have no inherent mechanism for recognizing individual receivers.

We can assess the cost of a position-sensing system in several ways. Time costs include factors such as the installation process's length and the system's administration needs. Space costs involve the amount of installed infrastructure and the hardware's size and form factor.

Some systems will not function in certain environments. One difficulty with GPS is that receivers usually cannot detect the satellites. This limitation has implications for the kind of applications we can build using GPS. For example, because most wired phones are located indoors, even if its accuracy and precision were high enough to make it conceivable, GPS does not provide adequate support for an application that routes phone calls to the land-line phone nearest the intended recipient. A possible solution that maintains GPS interaction yet works indoors uses a system of GPS repeaters mounted at the edges of buildings to rebroadcast the signals inside.

Table A.1: Current Position Sensing Technologies 1

Technology	Technique	Physical	Symbolic	Absolute	Relative
GPS	Radio time-of-flight lateration	o		o	
Active Badges	Diffuse infrared cellular proximity		o	o	
Active Bats	Ultrasound time-of-flight lateration	o		o	
MotionStar	Scene analysis, lateration	o		o	
VHF Omini-directional Ranging	Angulation	o		o	
Cricket	Proximity, lateration		o	o	o
MSR RADAR	802.11 RF scene analysis and triangulation	o		o	
PinPoint 3D-ID	RF lateration	o		o	
Avalanche Transceivers	Radio signal strength proximity	o			o
Easy Living	Vision, triangulation		o	o	
Smart Floor	Physical contact proximity	o		o	
Automatic ID System	Proximity		o	o	o
Wireless Andrew	802.11 proximity		o	o	
E911	Triangulation	o		o	
SpotON	Ad hoc lateration	o		o	

Table A.2: Current Position Sensing Technologies 2

Technology	Accuracy and precision if available	Scale
GPS	1-5 meters 95-99%	24 satellites worldwide
Active Badges	Room size	1 base per room, badge per base per 10 sec
Active Bats	9 cm 95%	1 base per 10 square meters, 25 computations per room per sec
MotionStar	1 mm, 1 ms, 0.1^o	
VHF Omini-directional Ranging	1^o radial	Several transmitters per metropolitan area
Cricket	4x4 ft. regions	1 beacon per 16 square ft.
MSR RADAR	3-4.3m 50%	3 bases per floor
PinPoint 3D-ID	1-3m	Several bases per building
Avalanche Transceivers	Variable, 60-80 meter range	1 transceiver per person
Easy Living	Variable	3 cameras per small room
Smart Floor	Spacing of pressure sensors	Complete sensor grip per floor
Automatic ID System	Range of sensing phenomenon	Sensor per position
Wireless Andrew	802.11 cell size	Many bases per campus
E911	150-300 m	Density of cellular infrastructure
SpotON	Depends on cluster size	Cluster at least 2 tags

Table A.3: Current Position Sensing Technologies 3

Technology	Cost	Limitations
GPS	Expensive infrastructure $100 receivers	Not indoors
Active Badges	Administration cost, cheap tags and bases	Sunlight and fluorescent light interfere with infrared
Active Bats	Administration costs, cheap tags and sensors	Required ceiling sensor grid
MotionStar	Controlled senses, expensive hardware	Control unit tether, precise installation
VHF Omini-directional Ranging	Expensive infrastructure, inexpensive aircraft receivers	30-140 nautical miles, line of sight
Cricket	$10 beacons and receivers	No central management receiver computation
MSR RADAR	802.11 network installation, = $100 wireless NICs	Wireless NICs required
PinPoint 3D-ID	Infrastructure installation, expensive hardware	Proprietary 802.11 interference
Avalanche Transceivers	$200 per transceiver	Short radio range unwanted signal attenuation
Easy Living	Processing power, installation cameras	Ubiquitous public cameras
Smart Floor	Installation of sensor grid, creation of footfall training dataset	Recognition may not scale to large populations
Automatic ID System	Installation, variable hardware costs	Must know sensor locations
Wireless Andrew	802.11 deployment	Wireless NICs required, RF cell geometries
E911	Upgrading phone hardware of cell infrastructure	Only where cell coverage exists
SpotON	$30 per tag, no infrastructure	Attenuation less accurate then time-of-flight

Appendix B

The Chord Protocol

This section describes the Chord protocol. The Chord protocol specifies how to find the locations of keys, how new nodes join the system, and how to recover from the failure (or planned departure) of existing nodes. In this paper we assume that communication in the underlying network is both symmetric (if A can route to B, then B can route to A), and transitive (if A can route to B and B can route to C, then A can route to C).

B.1 Overview

At its heart, Chord provides fast distributed computation of a hash function mapping keys to nodes responsible for them. Chord assigns keys to nodes with consistent hashing [39], [40], which has several desirable properties. With high probability the hash function balances load (all nodes receive roughly the same number of keys). Also with high probability, when an Nth node joins (or leaves) the network, only a $O(1 = N)$ fraction of the keys are moved to a different location - this is clearly the minimum necessary to maintain a balanced load.

Chord improves the scalability of consistent hashing by avoiding the requirement that every node know about every other node. A Chord node needs only a small amount of "routing" information about other nodes. Because this information is distributed, a node resolves the hash function by communicating with other nodes. In an N-node network, each node maintains information about only $O(logN)$ other nodes, and a lookup requires $O(logN)$ messages.

B.2 Consistent Hashing

The consistent hash function assigns each node and key an m-bit identifier using SHA-1 [44] as a base hash function. A node's identifier is chosen by hashing the node's IP address, while a key identifier is produced by hashing the key. We will use the term "key" to refer to both the original key and its image under the hash function, as the meaning will be clear from context. Similarly, the term "node" will refer to both the node and its identifier under the hash function. The identifier length m must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible.

Consistent hashing assigns keys to nodes as follows. Identifiers are ordered on an identifier circle modulo 2^m. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) K in the identifier space. This node is called the successor node of key k, denoted by $successor(k)$. If identifiers represented as a circle of numbers from 0 to 2^m-1, then $successor(k)$ is the first node clockwise from k. In the remainder, we will also refer to the identifier circle as the Chord ring.

Figure B.1 shows a Chord ring with $m = 6$. The Chord ring has 10 nodes and stores five keys. The successor of identifier 10 is node 14, so key 10 would be located at node 14. Similarly, keys 24 and 30 would be located at node 32, key 38 at node 38, and key 54 at node 56.

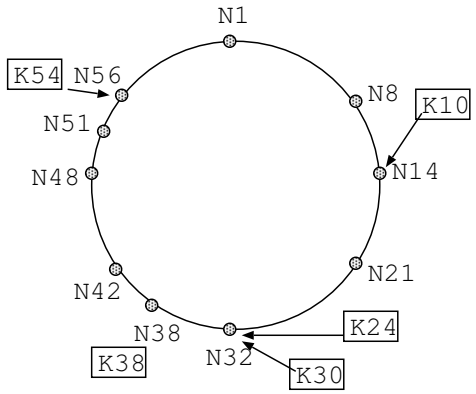


Figure B.1: An identifier circle (ring) consisting of 10 nodes storing five keys

Consistent hashing is designed to let nodes enter and leave the network with minimal disruption. To maintain the consistent hashing mapping when a node n joins the network, certain keys previously assigned to n's successor now become assigned to n. When node n leaves the network, all of its assigned keys are reassigned to n's successor. No other changes in assignment of keys

to nodes need occur. In the example above, if a node were to join with identifier 26, it would capture the key with identifier 24 from the node with identifier 32.

The following results are proven in the papers that introduced consistent hashing [39], [40]:

$TheoremIV.1$: For any set of N nodes and K keys, with high probability:

1. Each node is responsible for at most $(1+\in)K/N$ keys.
2. When an $(N+1)^{st}$ node joins or leaves the network, responsibility for $O(K/N)$ keys changes hands (and only to or from the joining or leaving node).

When consistent hashing is implemented as described above, the theorem proves a bound of $\in= O(longN)$. The consistent hashing paper shows that $\in$ can be reduced to an arbitrarily small constant by having each node run $\Omega(logN)$ virtual nodes, each with its own identifier. In the remainder of this paper, we will analyze all bounds in terms of work per virtual node. Thus, if each real node runs v virtual nodes, all bounds should be multiplied by v.

The phrase "with high probability" bears some discussion. A simple interpretation is that the nodes and keys are randomly chosen, which is plausible in a non-adversaria model of the world. The probability distribution is then over random choices of keys and nodes, and says that such a random choice is unlikely to produce an unbalanced distribution. A similar model is applied to analyze standard hashing. Standard hash functions distribute data well when the set of keys being hashed is random. When keys are not random, such a result cannot be guaranteed - indeed, for any hash function, there exists some key set that is terribly distributed by the hash function (e.g., the set of keys that all map to a single hash bucket). In practice, such potential bad sets are considered unlikely to arise. Techniques have also been developed [45] to introduce randomness in the hash function; given any set of keys, we can choose a hash function at random so that the keys are well distributed with high probability over the choice of hash function. A similar technique can be applied to consistent hashing; thus the "high probability" claim in the theorem above. Rather than select a random has function we make use of the SHA-1 hash which is expected to have good distributional properties.

Of course, once the random hash function has been chosen, an adversary can select a badly distributed set of keys for that hash function. In our application, an adversary can generate a large set of keys and insert into the Chord ring only those keys that map to a particular node, thus creating a badly distributed set of keys. As with standard hashing, however, we expect

that a non-adversaria set of keys can be analyzed as if it were random. Using this assumption, we state many of our results below as "high probability" results.

B.2.1 Simple Key Location

This section describes a simple but slow Chord lookup algorithm. Succeeding sections will describe how to extend the basic algorithm to increase efficiency, and how to maintain the correctness of Chord's routing information.

Lookups could be implemented on a Chord ring with little per-node state. Each node need only know how to contact its current successor node on the identifier circle. Queries for a given identifier could be passed around the circle via these successor pointers until they encounter a pair of nodes that straddle the desired identifier; the second in the pair is the node the query maps to.

Figure B.2 (a) shows pseudocode that implements simple key lookup. Remote calls and variable references are preceded by the remote node identifier, while local variable references and procedure calls omit the local node. Thus n.foo() denotes a remote procedure call of procedure foo on node n, while n.bar, without parentheses, is an RPC to fetch a variable bar from node n. The notation (a; b] denotes the segment of the Chord ring obtained by moving clockwise from (but not including) a until reaching (and including) b.

Figure B.2 (b) shows an example in which node 8 performs a lookup for key 54. Node 8 invokes $find\_successor$ for key 54 which eventually returns the successor of that key, node 56. The query visits every node on the circle between nodes 8 and 56. The result returns along the reverse of the path followed by the query.

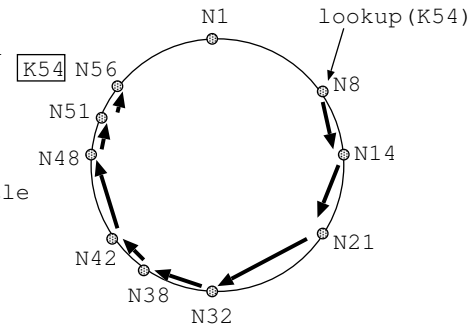
B.2.2 Scalable Key Location

The lookup scheme presented in the previous section uses a number of messages linear in the number of nodes. To accelerate lookups, Chord maintains additional routing information. This additional information is not essential for correctness, which is achieved as long as each node knows its correct successor.

As before, let m be the number of bits in the key/node identifiers. Each node n maintains a routing table with up to m entries (we will see that in fact only $O(logn)$ are distinct), called the $fingertable$. The i^{th} entry in the table at node n contains the identity of the $first$ node s that succeeds n by at least 2^{i-1} on the identifier circle, i.e., $s = successor(n + 2^{i-1})$, where

```
//ask node n to find the successor of id
n.find_sucessor(id)
  if(id ∈ (n,sucessor])
    return sucessor,
  else
    // forward the query around the circle
    return sucessor.find_sucessor(id);
```

(a)



(b)

Figure B.2: (a)Simple(but slow) pseudocode to find the successor node of an identifier id. Remote procedure calls and variable lookups are proceeded by the remote node. (b)The path taken by a query from node 8 for key 54, using the pseudocode in (a)

$1 \leq i \leq m$ (and all arithmetic is modulo 2^m). We call nodes the i^{th} finger of node n, and denote it by $n.finger[i]$ (see Table I). A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node. Note that the first finger of n is the immediate successor of n on the circle; for convenience we often refer to the first finger as the *successor*.

The example in Figure B.3 (a) shows the finger table of node 8. The first finger of node 8 points to node 14, as node 14 is the first node that succeeds $(8 + 2^0)$ mod $2^6 = 9$. Similarly, the last finger of node 8 points to node 42, as node 42 is the first node that succeeds $(8 + 2^5)$ mod $26 = 40$.

This scheme has two important characteristics. First, each node stores information about only a small number of other nodes, and knows more about nodes closely following it on the identifier circle than about nodes farther away. Second, a node's finger table generally does not contain enough information to directly determine the successor of an arbitrary key k. For example, node 8 in Figure B.3 (a) cannot determine the successor of key 34 by itself, as this successor (node 38) does not appear in node 8.

Figure B.4 shows the pseudocode of the find successor operation, extended to use finger tables. If id falls between n and its successor, $find_successor$ is finished and node n returns its successor. Otherwise, n searches its finger table for the node n' whose ID most immediately precedes id, and then invokes $find_successor$ at n'. The reason behind this choice of n' is that the closer n' is to id, the more it will know about the identifier circle in the region

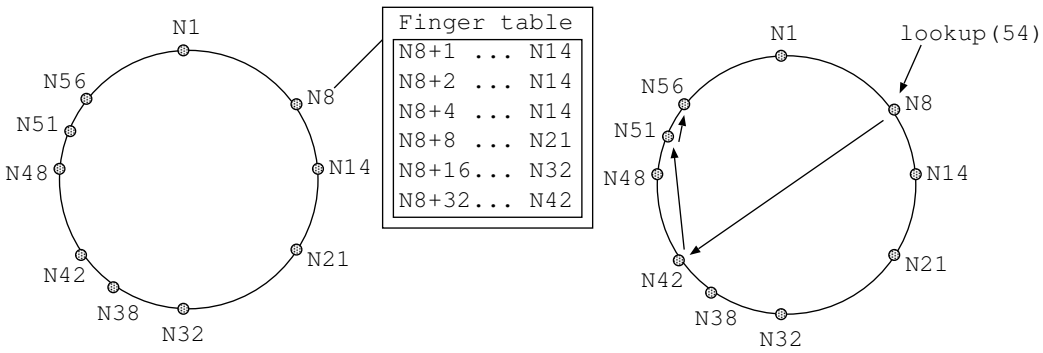


Figure B.3: (a)The finger table entries for node 8. (b) The path a query for key 54 starting at node 8, using the algorithm in Figure **??**

of id.

```
//ask node n to find the successor of id
n.find_sucessor(id)
  if(id ∉ (n,sucessor])
    return sucessor;
  else
    n' = closest_preceding_node(id);
    return sucessor.find_sucessor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if(finger[i] ∉ (n, id))
      return finger[i];
  return n;
```

Figure B.4: Scalable key lookup using the finger table

As an example, consider the Chord circle in Figure B.4 (b), and suppose node 8 wants to find the successor of key 54. Since the largest finger of node 8 that precedes 54 is node 42, node 8 will ask node 42 to resolve the query. In turn, node 42 will determine the largest finger in its finger table that precedes 54, i.e., node 51. Finally, node 51 will discover that its own successor, node 56, succeeds key 54, and thus will return node 56 to node 8.

Since each node has finger entries at power of two intervals around the identifier circle, each node can forward a query at least halfway along the remaining distance between the node and the target identifier. From this intuition follows a theorem:

TheoremIV.2 : With high probability, the number of nodes that must be contacted to find a successor in an N-node network is $O(logN)$.

Proof : Suppose that node n wishes to resolve a query for the successor of k. Let p be the node that immediately precedes k. We analyze the number of query steps to reach p.

Recall that if $n \neq p$, then n forwards its query to the closest predecessor of k in its finger table. Consider the i such that node p is in the interval $[n + 2^{i-1}, n + 2^i)$. Since this interval is not empty (it contains p), node n will contact its i^th finger, the first node f in this interval. The distance (number of identifiers) between n and f is at least 2^i, which means the distance between them is at most 2^{i-1}. This means f is closer to p than to n, or equivalently, that the distance from to is at most half the distance from n to p.

If the distance between the node handling the query and the predecessor p halves in each step, and is at most 2^m initially, then within m steps the distance will be one, meaning we have arrived at p.

In fact, as discussed above, we assume that node and key identifiers are random. In this case, the number of forwarding necessary will be $O(logN)$ with high probability. After $2logN$ forwardings, the distance between the current query node and the key k will be reduced to at most $2^m/N2$. The probability that any other node is in this interval is at most $1/N$, which is negligible. Thus, the next forwarding step will find the desired node.

In the section reporting our experimental results (Section V), we will observe (and justify) that the average lookup time is $\frac{1}{2}logN$.

Although the finger table contains room for m entries, in fact only $O(logN)$ fingers need be stored. As we just argued in the above proof, no node is likely to be within distance $2^m/N2$ of any other node. Thus, the i^th finger of the node, for any $i \leq m-2logN$, will be equal to the node's immediate successor with high probability and need not be stored separately.

B.2.3 Dynamic Operation and Failures

In practice, Chord needs to deal with nodes joining the system and with nodes that fail or leave voluntarily. This section describes how Chord handles these situations.

Node Joins and Stabilization

In order to ensure that lookups execute correctly as the set of participating nodes changes, Chord must ensure that each node's successor pointer is up to date. It does this using a "stabilization" protocol that each node runs

periodically in the background and which updates Chord's finger tables and successor pointers.

Figure B.5 shows the pseudocode for joins and stabilization. When node n first starts, it calls $n.join(n')$, where n' is any known Chord node, or $n.create()$ to create a new Chord network. The $join()$ function asks n' to find the immediate successor of n. By itself, $join()$ dose not make the rest of the network aware of n.

```
// create a new Chord ring
n.create()
  predecessor = nil;
  sucessor = n;
// join a Chord ring containing node n'
n.join(n')
  predecessor = nil;
  successor = n'.find_successor(n);
// called periodically. veryifies n's immediate successor,
// and tells the sucessor about n
n.stabilize()
  x = successor.predecessor;
  if(x ∈ (n, successor))
    successor = x;
  successor.notify(n);
// n' thinks it might be our predecessor
n.notify(n')
  if(predecessor is nil or n' ∈ (predecessor, n))
    predecessor = n';
// called periodically. refreshes finger table entries.
// next stores the index of the next finger to fix.
n.fix_fingers()
  next = next + 1;
  if(next > m)
    next = 1
  finger[next] = find_successor(n+2^ext-1 );
// called periodically. checks whether predecessor has faild.
n.check_predecessor()
  if(predecessor has failed)
    predecessor = nil;
```

Figure B.5: Pseudocode for stabilization

Every node runs $stabilize()$ periodically to learn about newly joined nodes. Each time node n runs $stabilize()$, it asks its successor for the successor's predecessor p, and decides whether p should be n's existence, giving the successor the chance to change its predecessor to n. The successor does this only if it knows of no closer predecessor than n.

Each node periodically calls $fix_fingers()$ to make sure its finger table

entries are correct; this is how new nodes initialize their finger tables, and it is how existing nodes incorporate new nodes into their finger tables. Each node also runs $check_predecessor()$ periodically, to clear the node's predecessor pointer if $n.predecessor$ has failed; this allows it to accept a new predecessor in $notify()$.

As a simple example, suppose node n joins the system, and its ID lies between nodes n_p and n_s. In its call to $join()$, n acquires n_s as its successor. Node n_s, when notified by n, acquires n as its predecessor. When n_p next run $stabilize()$, its asks n_s for its predecessor (which is now n); n_p then acquires n as its successor. Finally, n_p notifies n, and n acquires n_p as its predecessor. At this point, all predecessor and successor pointers are correct. At each step in the process, n_s is reachable from n_p using successor pointers; this means that lookups concurrent with the join are not disrupted. Figure B.6 illustrates the join procedure, when n's ID is 26, and the IDs of n_s and n_p are 21 and 32, respectively.

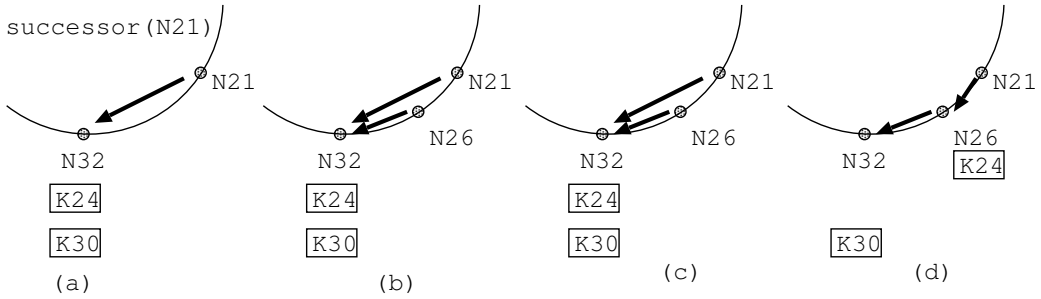


Figure B.6: Example illustrating the join operation. Node 26 joins the system between nodes 21 and 32. The arcs represent the successor relationship. (a) Initial state: node 21 points to node 32; (b) node 26 finds its successor (i.e. node32) and points to it; (c) node 26 copies all keys less than 26 from 32; (d) the stabilize procedure updates the successor of node 21 to node 26.

As soon as the successor pointers are correct, calls to $find_successor()$ will reflect the new node. Newly-joined nodes that are not yet reflected in other nodes' finger tables may cause $find_successor()$ to initially undershoot, but the loop in the lookup algorithm will nevertheless follow successor ($finger[1]$) pointers through the newly-joined nodes until the correct predecessor is reached. Eventually $fix_fingers()$ will adjust finger table entries, eliminating the need for these linear scans.

The following result, proved in [46], shows that the inconsistent state caused by concurrent joins is transient.

TheoremIV.3 : If any sequence of join operations is executed interleaved with stabilizations, then at some time after the last join the successor pointers will form a cycle on all the nodes in the network.

In other words, after some time each node is able to reach any other node in the network by following successor pointers.

Our stabilization scheme guarantees to add nodes to a Chord ring in a way that preserves reachability of existing nodes, even in the face of concurrent joins and lost and reordered messages. This stabilization protocol by itself won't correct a Chord system that has split into multiple disjoint cycles, or a single cycle that loops multiple times around the identifier space. These pathological cases cannot be produced by any sequence of ordinary node joins. If produced, these cases can be detected and repaired by periodic sampling of the ring topology [46].

Impact of Node Joins on Lookups

In this section, we consider the impact of node joins on lookups. We first consider correctness. If joining nodes affect some region of the Chord ring, a lookup that occurs before stabilization has finished can exhibit one of three behaviors. The common case is that all the finger table entries involved in the lookup are reasonably current, and the lookup finds the correct successor in $O(logN)$ steps. The second case is where successor pointers are correct, but fingers are inaccurate. This yields correct lookups, but they may be slower. In the final case, the nodes in the affected region have incorrect successor pointers, or keys may not yet have migrated to newly joined nodes, and the lookup may fail. The higher-layer software using Chord will notice that the desired data was not found, and has the option of retrying the lookup after a pause. This pause can be short, since stabilization fixes successor pointers quickly.

Now let us consider performance. Once stabilization has completed, the new nodes will have no effect beyond increasing the N in the $O(logN)$ lookup time. If stabilization has not yet the N in the $O(logN)$ lookup time. If stabilization has not yet the new nodes. The ability of finger entries to carry queries long distances around the identifier ring does not depend on exactly which nodes the entries point to; the distance halving argument depends only on ID-space distance. Thus the fact that finger table entries may not reflect new nodes does not significantly affect lookup speed. The main way in which newly joined nodes can influence lookup speed is if the new nodes' IDs are between the targeted predecessor and the target. In that case the lookup will have to be forwarded through the intervening nodes, one at a time. But unless a tremendous number of nodes joins the system, the number of nodes

between two old nodes is likely to be very small, so the impact on lookup is negligible. Formally, we can state the following result. We call a Chord ring *stable* if all its successor and finger pointers are correct.

TheoremIV.4 : If we take a stable network with N nodes with correct finger pointers, and another set of up to N nodes joins the network, and all successor pointers (but perhaps not all finger pointers) are correct, then lookups will still take $O(logN)$ time with high probability.

Proof : The original set of fingers will, in $O(logN)$ time, bring the query to the old predecessor of the correct node. With high probability, at most $O(logN)$ new nodes will land between high probability, at most $O(logN)$ new nodes will land between traversed along successor pointers to get from the old predecessor to the new predecessor.

More generally, as long as the time it takes to adjust fingers is less than the time it takes the network to double in size, lookups will continue to take $O(logN)$ hops. We can achieve such adjustment by repeatedly carrying out lookups to update our fingers. It follows that lookups perform well so long as $\omega(log^2 N)$ rounds of stabilization happen between any N node joins.

Failure and Replication

The correctness of the Chord protocol relies on the fact that each node knows its successor. However, this invariant can be compromised if nodes fail. For example, in Figure B.3, if nodes 14, 21, and 32 fail simultaneously, node 8 will not know that node 38 is now its successor, since it has no finger pointing to 38. An incorrect successor will lead to incorrect lookups. Consider a query for key 30 initiated by node 8. Node 8 will return node 42, the first node it knows about from its finger table, instead of the correct successor, node 38.

To increase robustness, each Chord node maintains a *successorlist* of size r, containing the node's first r successors. If a node's immediate successor does not respond, the node can substitute the second entry in its successor list. All r successors would have to simultaneously fail in order to disrupt the Chord ring, an event that can be made very improbable with modest values of r. Assuming each node fails independently with probability p, the probability that all r successors fail simultaneously is only p_r. Increasing r makes the system more robust.

Handling the successor list requires minor changes in the pseudocode in Figures B.4 and B.5. A modified version of the stabilize procedure in Figure B.5 maintains the successor list. Successor lists are stabilized as follows: node n reconciles its list with its successor s by copying s's successor list, removing its last entry, and prepending s to it. If node n notices that its successor has failed, it replaces it with the first live entry in its successor list

and reconciles its successor list with its new successor. At that point, n can direct ordinary lookups for keys for which the failed node was the successor to the new successor. As time passes, $fix_fingers$ and stabilize will correct finger table entries and successor list entries pointing to the failed node.

A modified version of the $closest_preceding$ node procedure in Figure B.4 searches not only the finger table but also the successor list for the most immediate predecessor of id. In addition, the pseudocode needs to be enhanced to handle node failures. If a node fails during the $find_successor$ procedure, the lookup proceeds, after a timeout, by trying the next best predecessor among the nodes in the finger table and the successor list.

The following results quantify the robustness of the Chord protocol, by showing that neither the success nor the performance of Chord lookups is likely to be affected even by massive simultaneous failures. Both theorems assume that the successor list has length $r = \omega(logN)$.

$TheoremIV.5$: if we use a successor list of length $r = \omega(logN)$ in a network that is initially stable, and then every node fails with probability 1/2, then with high probability $find_successor$ returns the closest living successor to the query key.

$Proof$: Before any nodes fail, each node was aware of its r immediate successors. The probability that all of these successors fail is $(1/2)^r$, so with high probability every node is aware of its immediate living successor. As was argued in the previous section, if the invariant that every node is aware of its immediate successor holds, then all queries are routed properly, since every node except the immediate predecessor of the query has at least one better node to which it will forward the query.

$TheoremIV.6$: In a network that is initially stable, if every node then fails with probability 1/2, then the expected time to execute $find_successor()$ is $O(logN)$.

$Proof$: Due to space limitations we omit the proof of this result, which can be found in the technical report [46].

Under some circumstances the preceding theorems may apply to malicious node failures as well as accidental failures. An adversary may be able to make some set of nodes fail, but have no control over the choice of the set. For example, the adversary may be able to affect only the nodes in a particular geographical region, or all the nodes that use a particular access link, or all the nodes that have a certain IP address prefix. As was discussed above, because Chord node IDs are generated by hashing IP addresses, the IDs of these failed nodes will be effectively random, just as in the failure case analyzed above.

The successor list mechanism also helps higher-layer software replicate data. A typical application using Chord might store replicas of the data

associated with a key at the k nodes succeeding the key. The fact that a Chord node keeps track of its r successors means that it can inform the higher layer software when successors come and go, and thus when the software should propagate data to new replicas.

Bibliography

[1] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 2003.

[2] V. Jacobosn and S. McCanne. Visual audio tool. Technical report, Lawrence Berkeley Laboratory. ftp://ftp.ee.lbl.gov/conferencing/vat.

[3] I. Kouvelas, V. Hardman, and J. Crowcroft. Network adaptive continuous-media applications through self organized transcoding. In *Proceeding of the Network and Operating Systems Support for Digital Audio and Video*, Cambridge, U.K., July 1997.

[4] S. McCanne. A distributed white board for network conferencing. Technical report, U.C Berkeley CS268 Computer Networks term project and paper, May 1992.

[5] S. McCanne and V. Jacobson. Vic: video conference. Technical report, Lawrence Berkeley Laboratory and University California, Berkeley. ftp://ftp.ee.lbl.gov/conferencing/vic.

[6] *Probe Information Service*. http://www.ipcar.org/.

[7] InternetITS Consortium. *InternetITS Project*. http://www.internetits.org/.

[8] Keio University and WIDE Project. *InternetCAR Project*. http://www.sfc.wide.ad.jp/InternetCAR/.

[9] Infrastructure Japanese Ministry of Land and Road Bureau Transport. *What's ITS*. http://www.its.go.jp/ITS/topindex/topindex_c1.html.

[10] Infrastructure Japanese Ministry of Land and Road Bureau Transport. *ETC Service*. http://www.its.go.jp/ITS/topindex/topindex_ETC.html.

[11] D. Johnson, C. Perkins, and J. Arkko. *Mobility Support in IPv6*. the IETF, June 2003. Internet-Drafts, draft-ietf-mobileip-ipv6-24.txt.

[12] the IETF. *Network Mobility Working Group*. http://www.ietf.org/html.charters/nemo-charter.html.

[13] Mark Weiser. Some computer science issues in ubiquitous computing. *ACM Communications*, March 1993.

[14] S. E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, Dec. 1991.

[15] Telecommunications Carriers Association Japan. *Number of Subscribers*, 2003. http://www.tca.or.jp/eng/database/daisu/index.html.

[16] Integration of new technologies weaves datacomm. ubiquitous networks. Technical report, Multimedia Research Institute Corp., 2003. http://www.mric.jp/english/tdcdma.html.

[17] UMTS World. TDCDMA specification and information page. Technical report, 2003.

[18] *The Internet Engineering Task Force*. http://www.ietf.org/.

[19] David B. Johnson, David A. Maltz, and Yih-Chun Hu. *The Dynamic Source Routing Protocol*. the IETF, April 2003. Internet-Drafts, draft-ietf-manet-dsr-09.txt.

[20] C. Perkins, E. Belding-Royer, and S. Bas. *Ad hoc On-Demand Distance Vector (AODV) Routing*. the IETF, July 2003. RFC3561.

[21] T. Clausen and P. Jacquet. *Optimized Link State Routing Protocol*. the IETF, October 2003. RFC3626.

[22] R. Ogier, F. Templin, and M. Lewis. *Topology Dissemination Based on Reverse-Path Forwarding*. the IETF, October 2003. Internet-Drafts, draft-ietf-manet-tbrpf-11.txt.

[23] The Geographer's Craft Project, Department of Geography, The University of Colorado. *The Global Positioning System Overview*, 2000. http://www.colorado.edu/geography/gcraft/notes/gps/gps.html.

[24] Ministry of Land, Infrastructure and Transport, Japan. *Shipment volume of Car Navigation System and VICS*, November 2003. http://www.its.go.jp/ITS/j-html/ITSinJapan/navi.html.

[25] Jeffrey Hightower and Gaetano Borriella. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[26] P. Francis. Yoid: Extending the internet multicast architecture. http://www.aciri.org/yoid/docs/index.html, April. 2000.

[27] Y. HUA CHU, S. RAO, and H. ZHANG. A case for end system multicast. In *Proceedings of SIGMETRICS 2000*, Santa Clara, CA, June 2000.

[28] S. McCanne. Symposium on the acceleration of rich media on the internet. In *Proceedings of Internet Broadcast Networks: Mass Media Distribution for the 21st Century*, May 1999.

[29] H. W. Holbrook and D.R. Cheriton. IP multicast channels EXPRESS support for large-scale single-source application. In *Proceedings of ACM SIGCOMM 99*, August 1999.

[30] I. Stoica, T.S.E. NG, and H. Zhang. Reunite: A recursive unicast approach to multicast. In *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.

[31] Y. Chawathe, S. McCanne, and E. Brewer. An architecture for internet content distribution as an infrastructure service. http://www.cs.berkeley.edu/ỹatin/papers/, 2000.

[32] N. Mimura, K. Nakauchi, H. Morikawa, and T. Aoyama. Relaycast: A middleware for application-level multicast services. In *Proceedings of 3rd International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed System(GP2PC 2003)*, pages 434–441, Tokyo, Japan, May 2003.

[33] S.Q. ZHAUNG, B. ZHAO, A. JOSEPH, R. KATZ, and J. Bayeus KUBIATOWICZ. An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of 7th International Workshop on Network and OS Support for Digital Audio and Video*, New York, July 2001. ACM.

[34] Robbert van Renesse, Kenneth P. Birman, Adrian Bozdog, Dan Dumitriu, Manpreet Singh, and Werner Vogels. Heterogeneity-aware peer-to-peer multicast. In *Proceedings of International Symposium on Distributed Computing (DISC 2003)*, Sorrento Italy, October 2003.

[35] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level multicast using content-addressable networks. *Lecture Notes in Computer Science*, 2233:14–??, 2001.

[36] S. Ratnasamy, P. Francis, M. Handlay, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of SIGCOMM 2001*. ACM, August 2001.

[37] T. Imielinski and J.C. Navas. Geographic addressing, routing, and resource discovery with the global positioning system. *Communications of the ACM Journal*, 1997.

[38] Young-Bae Ko and Nitin H. Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms.

[39] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving host spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 654–663, El Paso, TX, May 1997.

[40] D. Lewin. Consistent hashing and random trees: Algorithms for caching in distributed networks. Master's thesis, Department of EECS, MIT, 1998. http://thesis.mit.edu/.

[41] A. Rowstron and P. Druschel. A large-scale, persistent peer-to-peer storage utility. *ACM SOSP*, October 2001.

[42] B. Y. Zhao, J. Kubiatowicz, and A. D. Joesph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, April 2001.

[43] Kiyohide Nakauchi, Hiroyuki Morikawa, and Tomonori Aoyama. Distributed content location for ubiquitous environments. *Technical report of IEICE*, 2002.

[44] FIPS 180-1. Secure hash standard. Technical report, U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, April 1995.

[45] J.L. Carter and M.N.Wegman. Universal classes of hash functions. *Computer and System Sciences 18*, 2:143–154, 1979.

[46] I. Stoica, R. Morris, D. Liben-nowell, D. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Technical Report TR-819, MIT LCS, 2001. http://www.odos.lcs.mit.edu/chord/papers/.