行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告
□期中進度報告

多功能個人/群體通訊系統設計

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：96-2221-E-009-074-
執行期間：96 年 8 月 1 日至 97 年 7 月 31 日

計畫主持人：張明峰
共同主持人：
計畫參與人員：李忠育、范坤揚、蔡玄亞、游伯瑞

成果報告類型(依經費核定清單規定繳交)：□精簡報告 ■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：交通大學 資訊工程系

中　華　民　國　九十七 年　十 月　二十 日

行政院國家科學委員會專題研究計畫成果報告

多功能個人/群體通訊系統設計

Multi-Function Personal/Group Communication System Design

計畫編號：96-2221-E-009-074-
執行期限：2007.08.01 至 2008.07.31

主持人：張明峰　交通大學資工系
計畫參與人員： 李忠育、范坤揚、蔡玄亞、游伯瑞 交通大學資工系

中文摘要

現今的通訊系統，包含電信系統、即時通訊、電子郵件、和網路電話等，都利用明確的使用者識別碼來指定使用者。如不知道接收端的識別碼即不能通訊。本計畫開發一多功能通訊平台(MFPGC)能利用指定接收端的各項屬性，例如姓名、年紀，學校等。一個使用者屬性的集合可視為使用者的辨識碼，具有好記、有意義的優點。我們的系統同時支援明確辨識碼和多屬性之通訊功能。透過屬性發佈和搜尋的機制，媒合發話端和受話端。MFPGC 架構在 Chord 之上，使用布隆過濾器(Bloom filter)來儲存多屬性的資料，包含字串，數值和混合型態的屬性。系統可以讓發話端和受話端可以設定必須要符合的屬性以過濾不想接收的訊息。媒合發話端和受話端是透過比對代表受話端屬性的布隆過濾器與代表發話要求的布隆過濾器。第三者無法從布隆過濾器得知發話端的訊息或受話端的屬性，因此保護了使用者隱私。我們也支援離線使用者處理機制 — 使用者在各個狀態中離線，而通話要求會被保留到下次使用者上線後繼續進行。和傳統的通訊系統比起來，MFPGC 提供使用者新穎的，更彈性的通話方式。

關鍵字：網路通訊，多屬性查詢，布隆過濾器

ABSTRACT

Communication services today, such as telephony, instant message, email, and VoIP, use a specific user or device ID to specify the called party. If the callee's ID is unknown, communication becomes impossible. Another way to indicate the callee(s) is to specify

the callee’s attributes, such as the callee’s name, age, etc. A set of user attributes, which is meaningful and easy to remember, can be used to set up a communication. Developed in this project, a Multi-Function Personal/Group Communication (MFPGC) system supports communications using specific IDs and/or multiple under-specified attributes. Communications using multiple user attributes is feasible through publishing and querying users’ attributes on a DHT. The DHT of MFPGC system is based on Chord, and Bloom filter is used to represent user attributes, which can be string, numeric, and hybrid data. Communications are set up by matching the Bloom filters representing callees’ attributes and callers’ requests. Since only Bloom filters are published and queried, the users’ personal information is protected; a third party cannot obtain user information from the Bloom filters published. In addition, callees can specify necessary attributes that must be matched to filter unwanted calls. On the other hand, callers can also specify necessary attributes to limit the number of matched callees, and non-necessary attributes to increase likelihood of matching callee’s necessary attributes. To enhance the flexibility of communications, MFPGC system also stores and forwards communication requests for off-line users. Compared to traditional communication system, MFPGC system provides a new, flexible communication method for users.

Keywords: Internet communications, multiple-attribute query, Bloom filter

Tables of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Traditional telephony services have been transformed by the developments of mobile technologies and Internet technologies. More and more Internet communication services, such as MSN and Skype, have been widely used through wired and wireless networks. The advantages of using the Internet as the communication platform include low cost and more powerful service functionalities. However, the PLMN (Public Land Mobile Network) system is still the most used voice communication platform, so that most of Internet communication system developers have been making efforts to integrate their systems into mobile devices for the large user base, and even integrate their systems with the PSN (Public Switching Telephone Network).

1.1 Motivation

Communication services today, such as telephony, instant message, email, and VoIP, use a specific user or device ID, such as telephone number, e-mail address, and SIP URI, to specify the called party. A call cannot be set up if the callee's ID is unknown to the caller. It would be very useful if a caller can initiate a communication with a callee(s) without knowing the callee's specific ID. For example, Billy graduated from NCTU 20 years ago, and he wants to hold a reunion, but the telephone numbers and email addresses of some classmates have become invalid. It would be very useful if he can send a message to all his classmates without the knowledge of their telephone number of email address. The drawback of using specific ID for communications is that the ID may be no longer valid, and it may be difficult to associate an ID, such as telephone number, with its owner.

One way to indicate a callee(s) is to specify the callee's attributes, such as the name,

the age, and the school he or she studies, etc. A user can be associated with sets of user attributes. In other words, a set of user attributes can be used to specify the callee of a call. Such specifications are meaningful and easy to remember for the caller, and they do not change over time. However, a set of user attributes may not uniquely identify a specific callee in the caller's mind. It is the caller's responsibility to give the specifications with enough information about the callee.

To set up a communication with the callee's attributes specified by the caller, the callee needs to publish his or her user profile on the Internet first. The user profile may consist of attribute value pairs (AVPs). A call request is also made up of AVPs. To serve a call request, we need to do multi-attribute data matching between the call request and the callee's user profile. This can be done by a client-server architecture where the server stores all user profiles and matches call requests with the user profiles. Since a client-server architecture does not scale up well and has the disadvantage of single point of failure, we used peer-to-peer (P2P) technologies for multiple attributes matching.

Although P2P technologies has be used by many researchers for multiple attribute query, the routing overhead of most current P2P solutions increases linearly with the number of attributes. This can be a great drawback for our applications because the number of user attributes of a call request is usually large. An efficient publishing/query of user attribute profile on peer-to-peer overlay network is needed.

Another major issue to be addressed in a communication platform using user attributes to set up connections is user privacy. No one, except the matched callers, should obtain the user information of a user from the user's published user profile. A user profile should be encrypted first, and then published. However, the P2P overlay network should be able to match call requests and user profiles without the knowledge of it real

information. Another ignored issue is how to filter spam messages. A user should be able to select necessary attributes that must be matched for a call request, so that unwanted call request is ignored.

1.2 Objectives

The goal of this project is to build a communication system providing the following features:

1. Support communications using specific user ID and/or unspecific user attributes.

2. Flexible attributes including numeric and numeric range attributes.

3. Efficiently Match call requests and user profiles; filter unwanted call requests.

4. Protect all users' privacy and prevent maliciously gathering user information.

5. Store and forward call-related message for off-line users.

In order to implement such a communication system, we adopted structured peer-to-peer architecture - DHT (distributed hash table) - as the platform and proposed a novel publishing/query mechanism to accomplish above requirements.

The remaining of this report is organized as follows. Chapter 2 describes current work in peer-to-peer researches related to our system. Chapter 3 presents our system design in details. Chapter 4 describes the actual implementation, and Chapter 5 discusses the system performance. Finally, conclusions and future work are given in Chapter 6.

Chapter 2 Related Work

We used Chord [1], a DHT, as the under-layer routing platform, and other P2P technologies to support multiple-attribute query including range query. In this chapter, we briefly describe related research, and their limitations that cannot satisfy the requirements of our communication platform. In addition, we will describe the design of Bloom filter [2], which is a space-efficient, randomized data structure representing a data set. Bloom filter was used for user attribute publishing and call request matching in our system.

2.1 Chord and DHT routing

In the early stage of P2P development, there were two ways to locate resources on an overlay P2P network. In one way, resource indexing and searching were performed by centralized servers, and resource sharing was directly operated between peers. However single point of failure may imperil this approach. The most well-known system of this approach was Nasper [3]. In another way, resource searching was done by flooding the resource requests to peers. Each node randomly records a number of nodes as its neighbors and maintains direct connection with the neighbors. Gnutella is a typical example of this approach [4]. However, flooding would cause massive redundant messages, and result in inefficient usage of network bandwidth. It is clear that the aforementioned two approaches of P2P architecture do not scale up well, as the number of the nodes dramatically increases.

2.1.1 Structure peer-to-peer architecture

The applications of peer-to-peer technologies were not very popular until the invention of distributed hash tables (DHT). DHTs take advantage of consistent hash to locate resources in an overlay network. By using consistent hash, nodes and resources

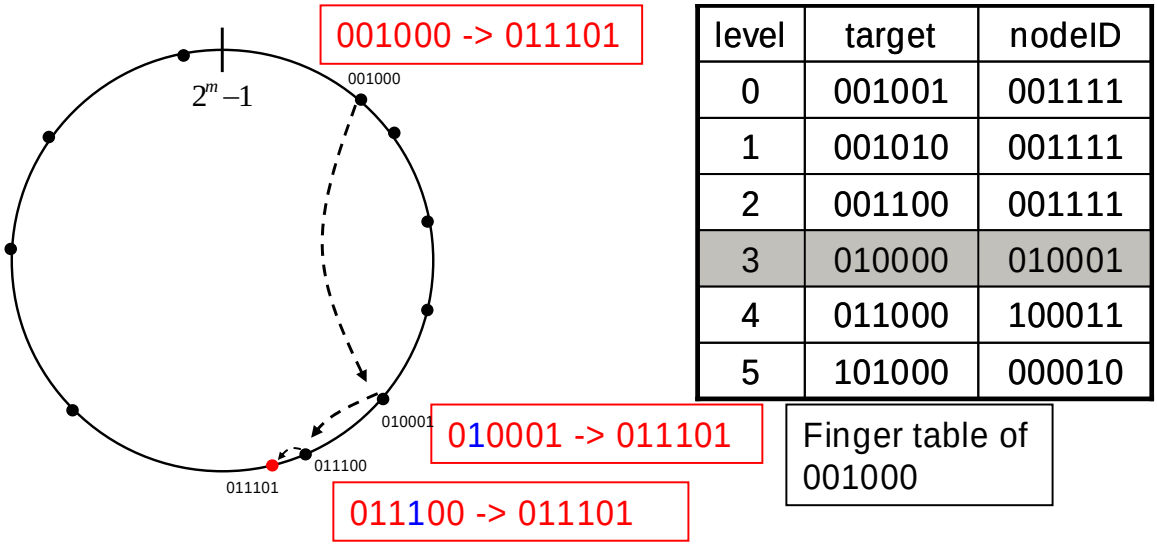
hash their names to keys or IDs, which is usually longer than 128 bits. A resource registers its location with the corresponding node with the same ID. If such a node does not exit, it registers with the previous node, the node with an ID less than, but the nearest to, the resource's ID. Any peer can find the location of a resource by using the same hash function. Due to the property of consistent hash, the corresponding node can be uniquely determined in the overlay network. By requesting the corresponding node, a peer can obtain the location of the requested resource. The inherent decentralized, efficient, and load-balancing properties of DHT make DHT the major research topic of recent P2P networks.

2.1.2 Chord

Although DHT is the basic idea of many structured peer-to-peer networks, there are still problems to be solved, for example, how to route a message to a node with a particular ID in the overlay network, how to handle ungracefully peer departure or churn, and how to join the overlay network. After DHT had been proposed, many researchers designed various kinds of routing algorithms, including Chord, Pastry [5], Tapestry [6], and CAN [7]. These algorithms provided characteristic routing with the same order of hop counts, $O(\log n)$, where n is the total number of nodes. In addition, they adopt different churn handling mechanisms.

Chord is one of the early DHT-based routing algorithms, and it is used as the backbone of our system. Chord works as follows. First, each node maps its name to an ID in the range of 0 to $2^m - 1$ where m is the scale of the DHT. All nodes form a ring with an increasing order of their IDs. Similar to a linked list, each node stores the links to its successor and neighbors in the ring. To route a message to a node with a particular ID, the message traverses the link of the ring. By adding a finger table in each node to keep links

to additional nodes, the routing can be done in $O(\log n)$ time, where n is the number of nodes. The finger table is a structure to save links to additional nodes whose node ID are $(2^i + k)\%2^m$, where k is the node ID where the finger table is saved and i is in the range from 0 to m-1. When a node generates a query, it first looks up its finger table to find an ID smaller than the target ID of the query, and then forwards the query. This greedy algorithm would pass the query to the closest node with ID smaller than the target ID. The node that receives the query repeats the same forwarding process until the corresponding node of the target ID. Figure 2.1 depicts a routing process from node 001000 to node 011101. In node 001000, the level 3 link of the finger table is the closest ID to the destination. node 001000 thus forwards the message to node 010001, and nodes in Chord will repeat the process until the message reaches the corresponding node of key 011101.



level	target	nodeID
0	001001	001111
1	001010	001111
2	001100	001111
3	010000	010001
4	011000	100011
5	101000	000010

Figure 2.1 A routing example and a finger table example

Each node of a Chord ring maintains $O(\log n)$ links to other nodes and a message can be routed to the corresponding node of a particular ID in $O(\log n)$ hops. Because a successful query requires every node in the routing path to forward the message, the correctness of the finger tables must be kept, especially the link to the successor. Chord

has a stabilization mechanism to handle ungraceful node departure and new node arrival. Periodically each node pings the successor and asks whether his predecessor has been changed. If so, the node updates his new successor to the old successor's new predecessor; this complete the joining of a new node. Each node also maintains several backup successors, which will be used when the successor ungracefully disappears. The stabilization mechanism provides reparation in churn, and works well with high probability. The nodes linked in a finger table also need to be checked periodically by sending keep-alive message to them. If any node does not reply a keep-alive message, the link is rebuilt. The aforementioned mechanisms make Chord stable during serious churn and provide more reliable publishing and query results for upper layer applications.

2.2 Multiple attributes

Multiple-attribute matching needs to perform an "and" operation, i.e., a query carries several attributes, and items that own all of those attributes match the query. In an ordinary DHT design, every attribute is hashed to a different node. As a result, a multiple-attribute queryis usually handled by all the nodes that store the attributes of the query. The overhead of DHT routing leads to inefficiency of a multiple-attribute query.

The most intuitive idea behind multiple-attribute query is through iterative search. The querying node first sends the query to the corresponding node of a selected attribute of the query. The corresponding node finds the matched results, and forwards the query and the matched results to the corresponding node of another attribute of the query. The process repeats and each corresponding node would keep only the intersection of its matched results and the matched results it received, until all attributes of the query have been handled. The responding node of the last attribute sends the final results back to the querying node. This is the simplest iterative way to achieve multiple attributes query.

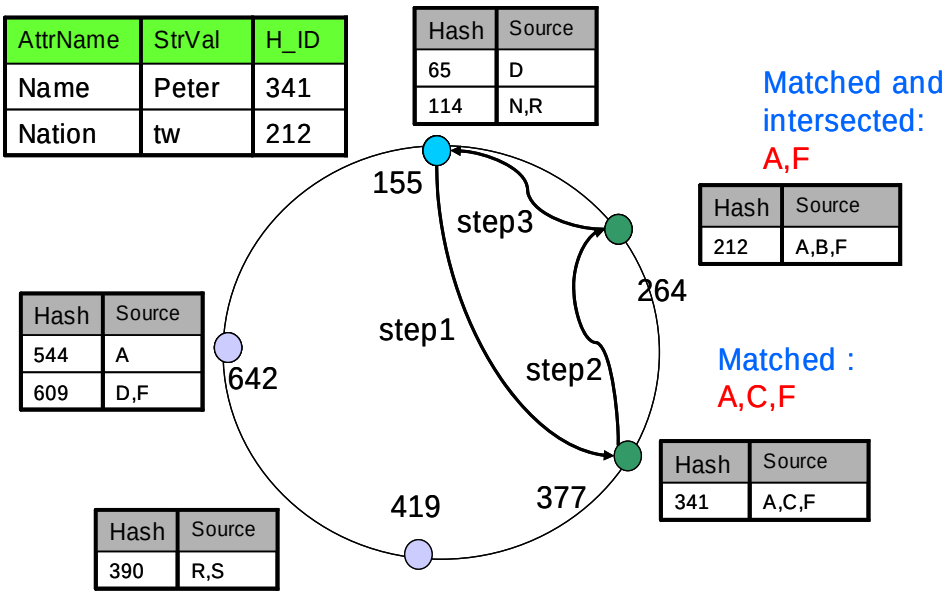


Figure 2.2 Iterative search for multi-attribute query.

Figure 2.2 depicts an iterative multiple attribute query. First, the querying node, node 155, sends the query to node 377, the responding node of attribute “Nation:tw”. Then, node 377 forwards the matched results {A,C,F} to node 264, the responding node of attribute “Name:Peter”. After intersecting the matched results, node 264 returns the results {A, F} to node 155.

Since the results matching an attribute may be a very large set in an iterative query, Reynolds [8] proposed that transmitting the Bloom filter of the matched set to reduce the transmission bandwidth. Bloom filter is a efficient data compression method; a data set can be compressed to an array with a small possibility of false positives. We will describe Bloom filter in more details later. Figure 2.3 shows a query example. The querying node searches attributes A and B. The left-hand side of Figure 2.3 depicts an iterative query; the right-hand side an iterative query using Bloom filter. The querying node first forwards the query to the responding node of A, that is, SA. SA hashed the matched items to a Bloom filter F(A), then forward F(A) to SB, the corresponding node of B, SB finds the matched items of B, and only keep the matched items that are also in F(A). If there

aremore than two attributes, the process repeated until all attributes are matched, and the matched results returned to the querying node. Due to the inherent false positive problem of Bloom filter, the final results of the query can be transmitted backward in the reverse order and checked twice. Although using Bloom filter reduces the transmission overhead, the additional check may cause overhead.

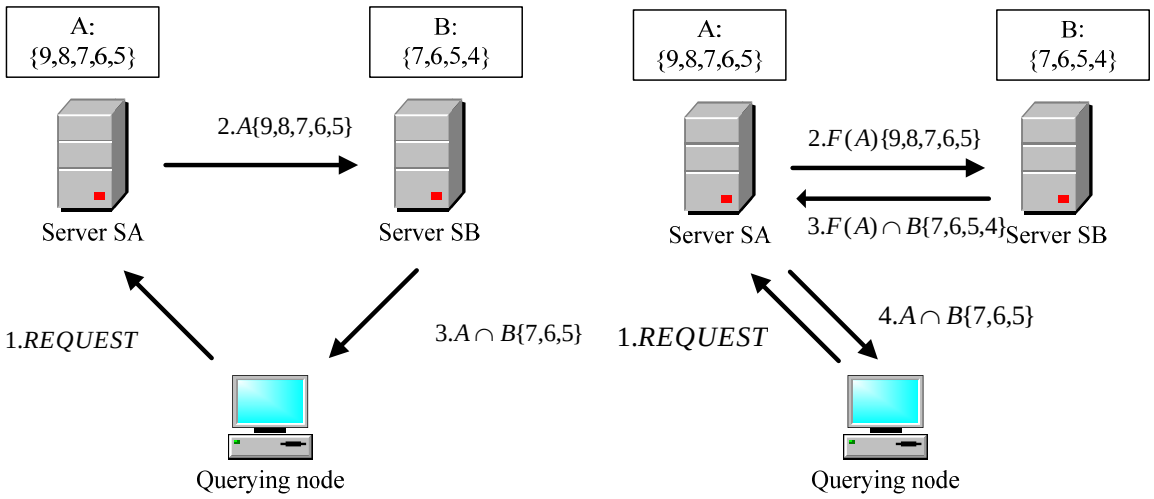


Figure 2.3 Relay search and Bloom filter used in relay search

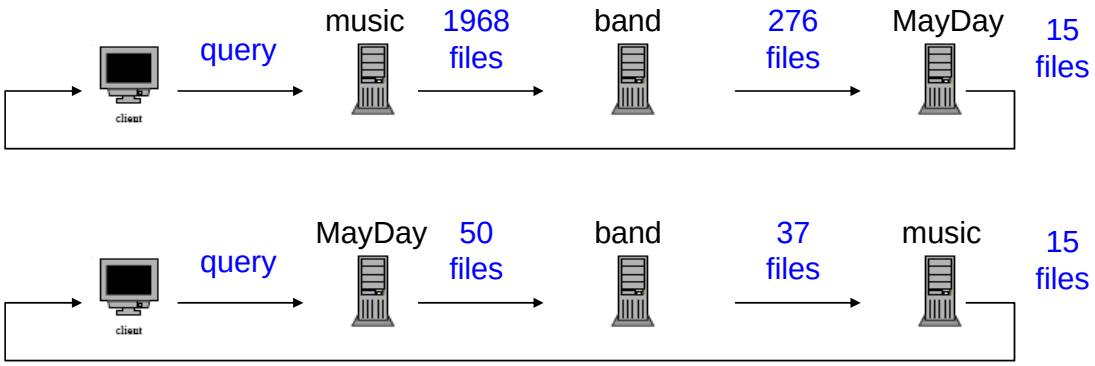


Figure 2.4: A query example of different query order

Lintao [9] observed that if we query more specific attributes first, we could reduce the amount of transmission data during an iterative query. More specific attributes mean fewer users have the attribute, so a query would match fewer items. They also proposed a fusion dictionary to dynamically detect hot keywords that many users use, and put hot keywords in the last of a query order. Each node maintains a fusion dictionary and

updates it by the responding nodes of keywords flooding adding-keyword message to every node. Furthermore, this keywords fusion mechanism can merge two hot keywords as one less hot keyword to enhance the query process, and a merged keyword can be added to the fusion dictionary to return hot. Figure 2.4 show the effects of query process with the fusion dictionary, the example on the top queries hot keywords first, while the example below queries the specific keywords first. We can clearyly observe the difference of transmission data. However, this algorithm would pay additional overhead to maintain and synchronize the fussion dictionary in each node.

2.3 Range query

A range query searches for items with an attribute value in a range. For example, find a file of size 100K-200K bytes. Consistent hash functions used in most DHTs break the locality of attributes for load balancing, i.e., two similar keywords with their attribute values close may be hashed to two corresponding nodes far distributed. However, a range query needs to find keywords with similar attributes, and may not be efficiently performed by a DHT. There have been some researches toward efficient range query. Most of them used location preserving hash function and provided another way for load balancing.

MAAN [10] supported both multiple-attribute and range queries based on Chord. It useed uniform location preserving hash for range query and single attribute dominated query for multiple attributes query. Location preserving hash used in MAAN simply linearly maps a numeric attribute domain to the namespace of Chord ID. The minimum of the numeric attribute maps to 0 and the maximum to $2^m - 1$ in Chord. Although the hash has load balancing problems, the authors claimed that MAAN does suffer load unbalance by consider the real distribution of the numeric attribute in mapping instead of a linear

mapping. Single attribute dominated query selects an attribute to query instead of querying all attributes iteratively. In order to achieve this goal, a resource must be published to the corresponding nodes of all attributes of the resource with all the attribute information. Therefore, a multiple-attribute query can be compared all attributes in any of the corresponding node. Single attribute dominated query would suffer privacy problem and require more storage.

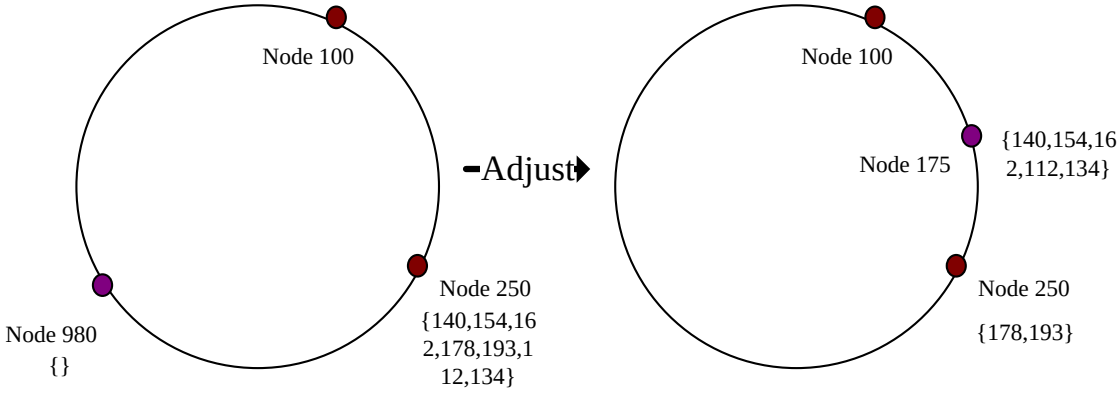


Figure 2.5 The load balance mechanism in Mercury

Mercury [11] adopted single attribute dominated query and location preserving hash, and further designed a dynamic load detecting and balancing mechanism that can effectively solve the load balancing problem. First, a Mercury node randomly and periodically sends a “load probing” packet to one of its neighbors. The “load probing” packet also contains pre-defined TTL value which decreases when it arrives at a new node. While the TTL value reaches zero, the final node sends back to the sender the collected load information. If a node observes that its own load is heavier than the collected load by a constant threshold, it sends a “light probing” packet to search a node with light load, and then ask that node to gracefully leave and rejoin at the heavier load position. According to the property of consistent hash, the heavy load would be shared by the rejoining node. In Figure 2.5 node 980 rejoins at the request of node 175 and shares the load between 100 and 250. Periodically executing this process makes the load of

every node balanced.

Another approach of multiple attributes and range queries is using space filling curves (SFC). SQUID [12] and SCARP [13] transforms multiple attributes range query into multidimensional query. Each dimension represents a numeric attribute; strings are treated as ASCII numbers. SQUID also uses SFC to map multi-dimension data into one dimension line and location preserving hash to map the line into Chord. Therefore multiple-attribute query can be handled in a node to reduce the transmission overhead. Figure 2.6 shows a 2-dimensional example using SFC to map a 2-dimensional plane to a 1-dimensional line, and on the right is the mapping of the range X:0-1 and Y:1-3 in a line.

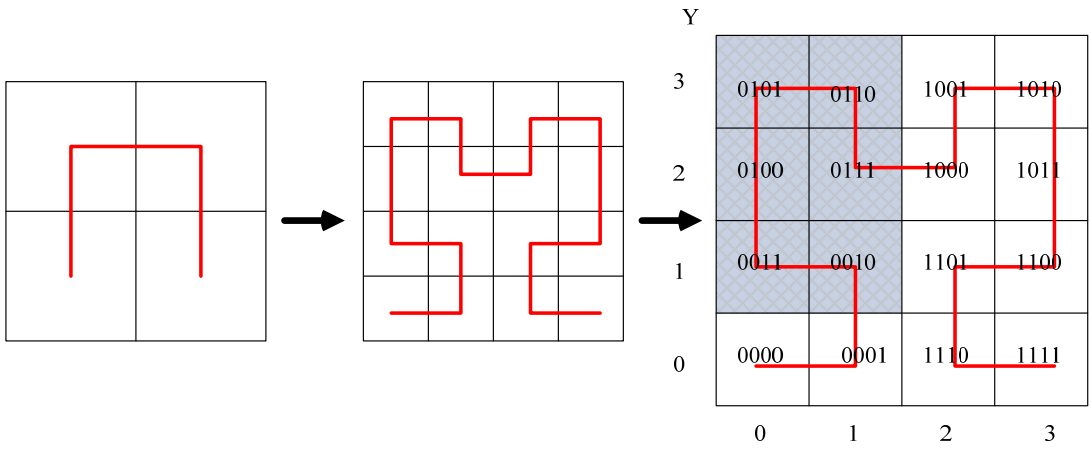


Figure 2.6 A space filling curve example

However, range search in SFC can still generate fragmentation because a contiguous range in a high dimension does not always map to a continuous segment in the line, especially in a higher dimension. As a result, range query using SFC only suits to a fixed and lower dimension, such as, longitude and latitude information.

2.4 Bloom filter

Bloom filter is a space-efficient data structure representing a data set. It supports elements insertion but not deletion. It check whether a certain element is in the set with a

probability of false positives. A Bloom filter contains an *m*-bit array and *k* independent hash functions. Initially, all bits of the *m*-bit array are set to 0. The element insertion process works as follow. First, use *k* hash functions to hash the element into *k* integers $h_1, h_2 \ldots h_k$ whose range is between 0 and *m*, and then set the bits in position $h_1, h_2 \ldots h_k$ to 1. All elements in a data set are inserted in the same way. To check if a certain element (testee) is in the data set represented by a Bloom filter, simply hash the testee into $h_1, h_2 \ldots h_k$ and check if all the *k* positions are set to 1. If so, the testee is assumed to be in the set. Bloom filter is very efficient in element insertion and checking using a constant bit array that provides good space utilization.

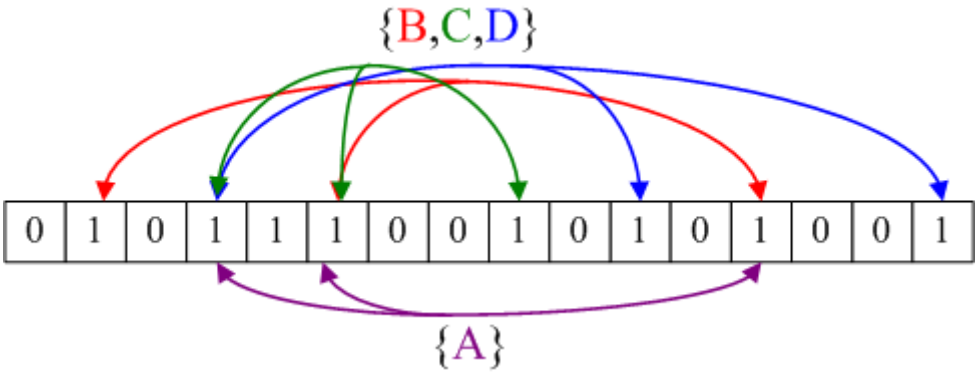


Figure 2.7 Multiple attributes with Bloom filter

However, the most serious hazard of bloom filter is false positives. Even though all hashed positions of a testee are set to 1, it is still possible that the testee does not belong to the data set. The positions might be set to 1 by other elements of the data set. Figure 2.7 shows an example false positive, the hashed positions of A collide with the hashed positions of elements B, C, and D. A is falsely determined to be in {B, C, D}. General speaking, the more number of bits set to 1 in a Bloom filter, the more likely false positives can occur. Furthermore, the number of elements (*n*) inserted to a Bloom filter of size *m* bits, and the number of hash function (*k*) determine the false positive probability. Therefore, applications using Bloom filter must adjust those parameters to achieve

desired false positive probability, which isusually below than 1%, and provide additional examination methods to handle false positives. In spite of false positive, Bloom filter has very good performance in terms of storage space and computation, and thus has been widely used in network applications. Our communication system also adopted Bloom filter to store data, and we will describe in the next chapter.

Chapter 3 System Design

3.1 System overview

MFPGC system used Chord as the application layer routing method and SIP as the communication protocol. The Figure 3.1 depicts that the system operates on the top of a Chord ring, and PDA users can only communicate with others through a P2P node. Users communicate with each other using SIP UA.

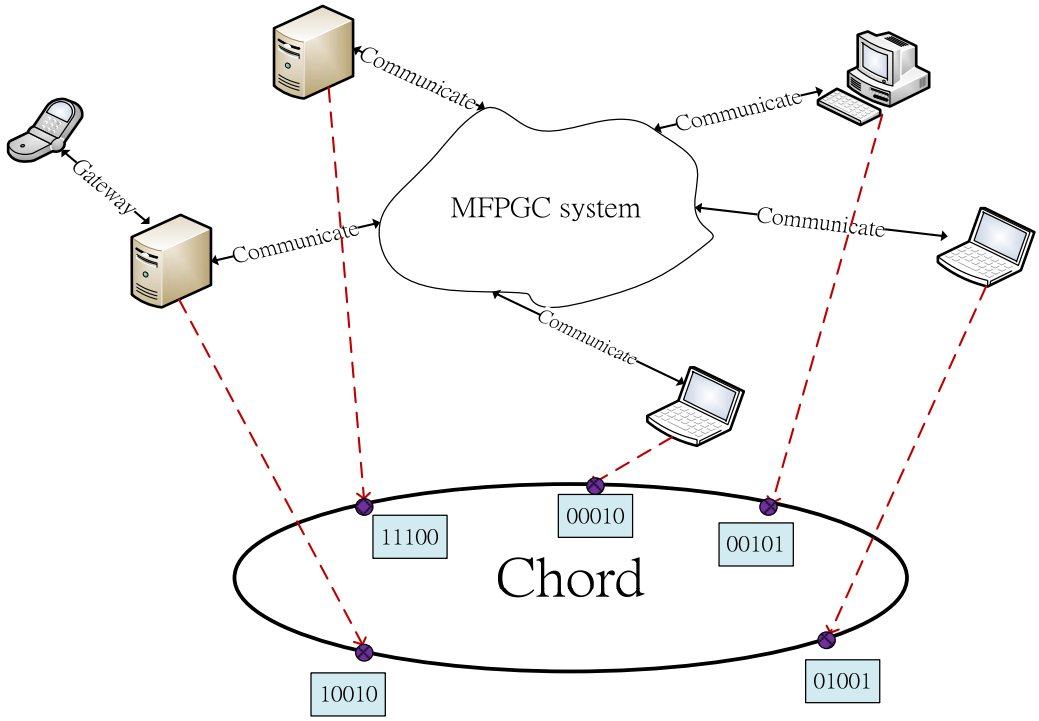


Figure 3.1 System Architecture Overview

The message routing on DHT follows the design of Chord, and thus the detaials will not be presented in this report. MFPGC system publishes and queries information over Chord and we do not make any change in message routing. Although we used Chord as the under-layer routing protocol, we can implement our system on any DHT-based routing protocol, if it supports single-attribute publishing and query.

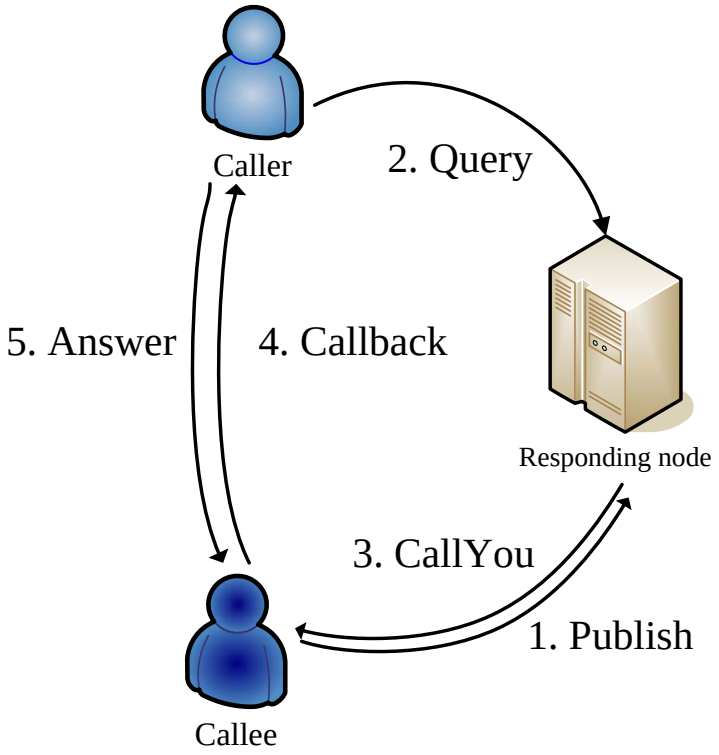


Figure 3.2 The call flow of MFPGC system

Since a communication using a specific ID is a conventional SIP call, it is not presented in this report; we only present communication using unspecific user attributes in MFPGC system. The communication works as follow:

0. A MFPGC user joins the communication system though a well-known node in the MFPGC system.

1. The user publishes his or her attributes to the DHT network, so that the responding nodes of each of the attributes maintain a copy of the user information. (Note that it is represented by a Bloom filter and will be described latter in this report).

2. A caller initiates a call request by querying users with certain attributes, and the query will be forwarded to the corresponding node of one of the attributes in the query.

3. The corresponding node compares the attributes of its data and the query, and then sends the call requests to all the matched users.

4. The matched users receive the call request and decide if they want to reply the call request. If so, a callback message is returned to the caller.

5. The caller can answer the call and start the communication with the callee.

The five steps is the basic call model of our system. However, the attributes can be numeric, string, or hybrid type, and they are handed by different methods.

3.2 Publish with Bloom filter

One of the most important features of our system is the usage of Bloom filter. As we have presented in Chapter 2, Bloom filter is a space-efficient data structure for representing a data set and widely used in network applications. Therefore, we use Bloom filter to store user attributes and to represent call request queries.

When a user registers the user's attributes in MFPGC system, the user would first a user profile stored in the local database of the user's device. A user profile consist of user attributes; each attribute is an (attribute name, value) pair. The attribute value can be of numeric, numeric range, string, or hybrid type. There are system-defined attributes whose attribute names are defined by the system, and user-defined attributes whose attribute names are defined by users to suit their purposes. For each attribute, we can obtain its corresponding node by hashing the (attribute name, value) pair by SHA1 hash function. When a user publishes his or her profile, the system sends publishing messages to the corresponding nodes of all attributes. As a result, a call request can be served by any corresponding node.

Instead of publishing user attributes in clear text, MFPGC publishes the Bloom filter of the attributes. Each published message includes the user contact information including the IP address, port, etc, and a Bloom filter representing all attributes. Figures 3.3 and 3.4

depicts the publishing process of Peter's attributes

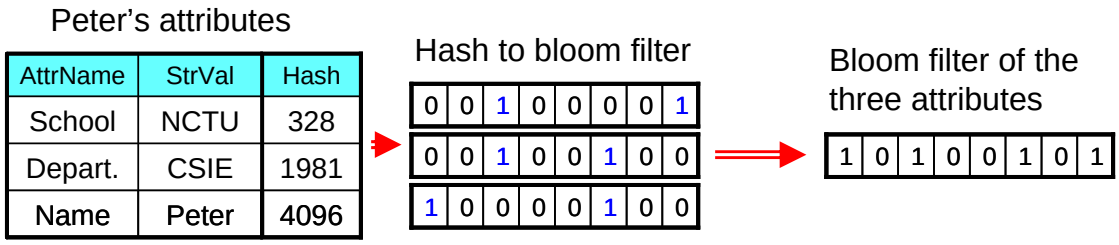


Figure 3.3 An example that maps three attributes of a profile into a Bloom filter

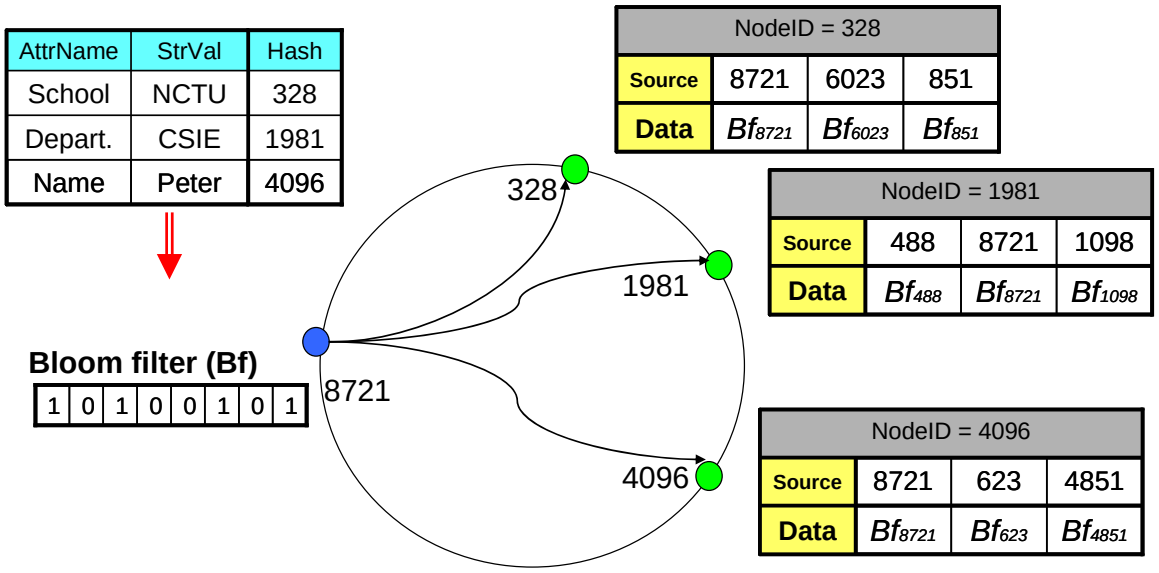


Figure 3.4 The publish process of the profile in Figure 3.3

3.3 Query with Bloom filter

A call request is a query to search users whose profiles match with the attributes specified in the call request. Similar to publishing, a query first maps the attributes of the call request to a Bloom filter, and then sends a query message including the Bloom filter to any one of the corresponding node of the attributes. The most significant difference between publishing and query is that publishing needs to be sent to all of the attributes but query only needs to be sent to one. Since the Bloom filter representing user

information has been published to all corresponding nodes, the query can match the Bloom filters in any one of those nodes. Furthermore, if we can choose a more specific attribute to query, we can balance the query load in the DHT network. MFPGC system currently still lacks of a mechanism that dynamically detects specific attributes.

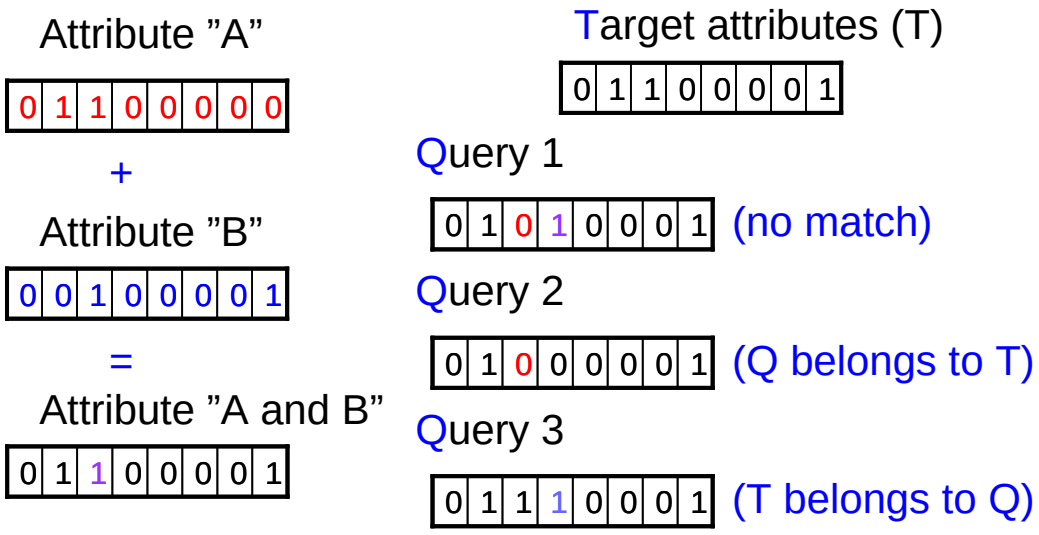


Figure 3.5 A Bloom filter matching example

When a node receives a query message, it compares the Bloom filter in the query with the Bloom filters published by users and stored in the local database. An example Bloom filter matching is depicted in Figure 3.5. On the left is the Bloom filter of user attribute set {A, B}. On the right is the matching between a target user profile (T) and queries (Q1-3). In general, Q $\subseteq$ T means a match, but we will describe additional constraints for the matching in next section.

When a corresponding node detects a match between a call request and a user profile, it sends a "CallYou" message to the node publishing the profile. There may be several published profiles matching the query, and all the publishing nodes will receive a "CallYou" message. When a user receives a "callyou" message, he or she can decide whether to call back or not. If so, the user initiates a SIP call to the caller. Note that if the

user decide not to call back, the caller is unknown of the existence of the matched user.

Another problem in using Bloom filter is false positive. Our solution is double check the call request at the callee's node where both user profile and the "CallYou" message are available. The "CallYou" message is encrypted so that only the matched nodes can decrypted; the corresponding nodes of the query cannot decrypt the message.

3.4 Numeric attributes

We have described that MFPGC system provides not only string type attributes but also numeric and hybrid attributes. Many user attributes, such as age, income, location, etc., contain numeric values, and the query may be searching for a range of the values, such as, age from 20 to 29. It is inefficient if we query each age at a time using the simple publish/query mechanism described. Therefore, a new query method for numeric attributes is needed. This problem is known as range query that we have presented in Chapter 2.

The method we used in MFPGC system for range query is dividing the domain of a numeric attribute into levels. Values in each level are treated as the same value. For example, attribute AGE can be partitioned into levels of every five years. Therefore, 1-5 is the first level, 6-10 is the second, and so on. However, this dividing results in more false positives in using Bloom filter. For example, we query a range "AGE 3-13", and the query include three age levels, "AGE 1-5", "AGE 6-10", and "AGE 11-15". It is clear that some values in the three levels do not match our query, such as, "AGE 14-15" and "AGE 1-3". These false positives in Bloom filter can also detected by the callee node. In other words, we check whether the range in a query covers the number of callee when a "callyou" message is received by the callee.

We store range attributes in a Bloom filter by the same way as string attributes. For example, inserting Latitude 23.0000 in a Bloom filter is to set array positions h_i("Latitude230000") to 1. For example, a square area of Latitude 25.0000-25.0999 and Longitude 121.1000-121.1999 represents approximately 1.1km*1km area. To be accurate to the four decimal places, the area can be described by two numeric ranges (250000-250999) and (1211000-1211999), each of which consists of 1000 numeric values. Note that the precession of this representation is about 10 meters. This type of data can be valuable in supporting location-based communications, for example, broadcasting a message to users in this area. However, it is clear that the classic Bloom filter cannot store this type of data in a space-efficient way. The simple dividing method may not be adequate for this application. We will investigate this problem in the future.

An important factor of dividing is the size of each level. The smaller the partition is the more numbers of levels are contained in a query. This means that more bits will be set to 1 in the Bloom filter. On the other hand, the bigger the partition is, the fewer number of levels are contained in a query, but the false positives caused by dividing occurs more often. Thus, the size of partition is a trade-off between the number of bits inserted in Bloom filter and the false positive rate generated by dividing.

3.5 Necessary attributes

We have described that MFPGC system enable users to filter unwanted calls. By specifying necessary attributes that must be matched for a call request, a user can filter unwanted calls, and only users who have sufficient personal information of a callee can reach the callee. MFPGC system provides two kinds of necessary attributes in both the sender side and the receiver side.

3.5.1 Receiver-specified necessary attribute set

To screen out unwanted call requests, a user can specify certain attributes that must be matched by call requests. These attributes will be referred to as the receiver-specified necessary attribute set and represented by a Bloom filter in publishing to the responding nodes. We modify the publishing message to include two Bloom filters, one representing all receiver-specified attributes set (RAS), and the other representing receiver-specified necessary attributes set (RNAS).

When a node receiving a query, in addition to checking whether the query is a subset of RAS, the node also checks whether the query contains RNAS, i.e., matches all receiver-specified attributes. If both conditions are satisfied, the query matches the user profile. If only the former condition satisfies but the latter one does not, it means the caller intends to call the callee but the callee would not like to receive this request. This mechanism costs overheads in maintaining and comparing two Bloom filters for each user profile on the responding node, but it can filter unwanted calls before the call requests are sent to callee. Figure 3.6 depicts an example of publishing Peter's attributes including RNAS and RAS to three responding nodes.

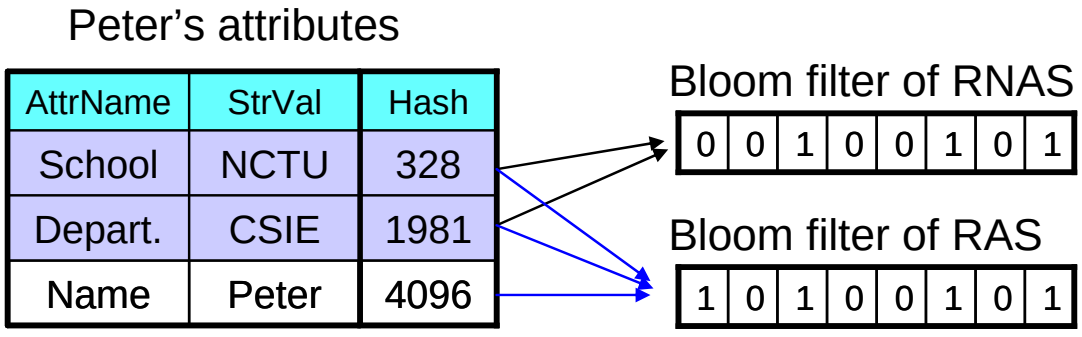


Figure 3.6.a Peter's user attributes; the first and second attributes are RNAS

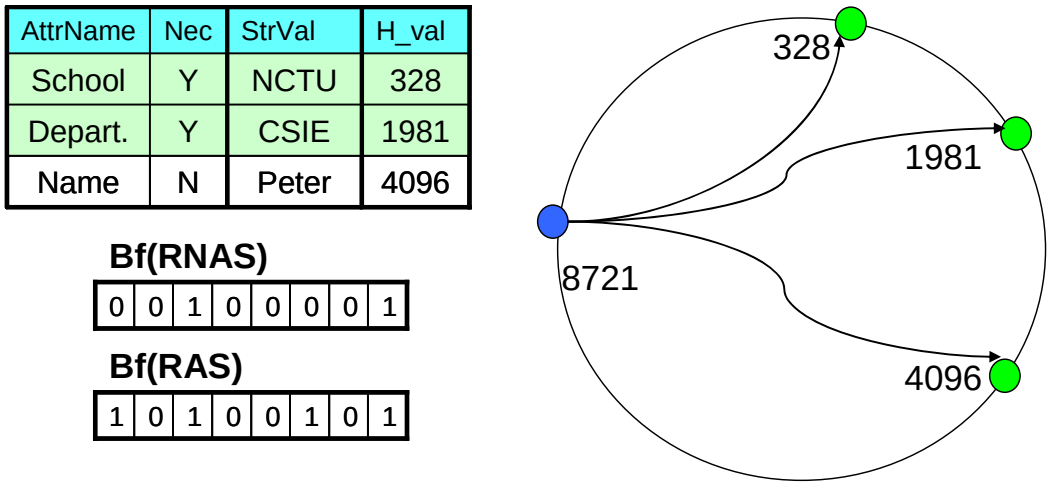


Figure 3.6.b The publishing Peter's attributes

3.5.2 Sender-specified necessary attribute set

To limit the number of users who will receive a call request, a caller can specify sender-specified necessary attribute set (SNAS) containing attributes that must be matched in the callee's user profile. Therefore, a call request is also a query represented by two Bloom filters. One representing SNAS; the other all sender-specified attribute set (SAS). The set SAS-SNAS consists of all non-necessary attributes specified by the sender; this set increase the likelihood of satisfying RNAS. In MFPGC system, a call request and a user profile match if and only if the following condition is satisfied.

$$(RANS \subseteq SAS)\,\&\,(SNAS \subseteq RAS)$$

Since attributes in SNAS must be matched, the query of a call request must be sent the corresponding node of a sender-specified necessary attribute. Consider that Peter's profile with necessary attributes School and Department has been published as shown in Figure 3.6. Figure 3.7.a depicts an example where SNAS is satisfied but RNAS is not. Since attributes "Depart:CSIE" and "Name:Peter are SNAS, MFPGC system chooses one of the responding nodes of SNAS to query. Node 4096 was chosen in this example. Although SNAS match Peter's RAS, but Peter's RNAS does not match because

"School:NCTU" is not included in query A. Figure 3.7b shows a 4-attribute query where RNAS is satisfied but SNAS is not. Although Peter's RNAS are subset of SAS, a sender-specified necessary attribute, "Name:John", is not contained in Peter's attributes, and thus the query is not forwarded to Peter. In Figure 3.7.c, although attribute "Name:John" does not match "Name:Peter", the two attributes are in neither RNAS nor SNAS, and both RNAS and SNAS are satisfied. The call request is forwarded to Peter.

Query A

AttrName	Nec	StrVal	H_val
Depart.	Y	CSIE	1981
Name	N	Peter	4096
Club	N	Bridge	3100

$RNAS \subseteq SAS$ false

$SNAS \subseteq RAS$ true

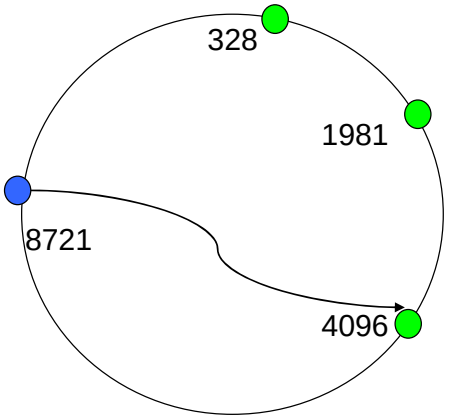


Figure 3.7.a SNAS is satisfied but RNAS is not

Query B

AttrName	Nec	StrVal	H_val
School	Y	NCTU	328
Name	N	John	882
Depart.	Y	CSIE	1981
Club	N	Bridge	3100

$RNAS \subseteq SAS$ true

$SNAS \subseteq RAS$ false

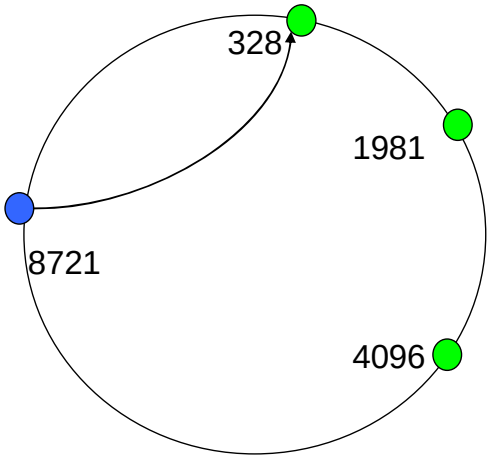


Figure 3.7.b RNAS is satisfied but SNAS is not

Query C

AttrName	Nec	StrVal	H_val
Depart.	Y	CSIE	1981
Name	N	John	882
School	Y	NCTU	328
Club	N	Bridge	3100

$RNAS \subseteq SAS$ true

$SNAS \subseteq RAS$ true

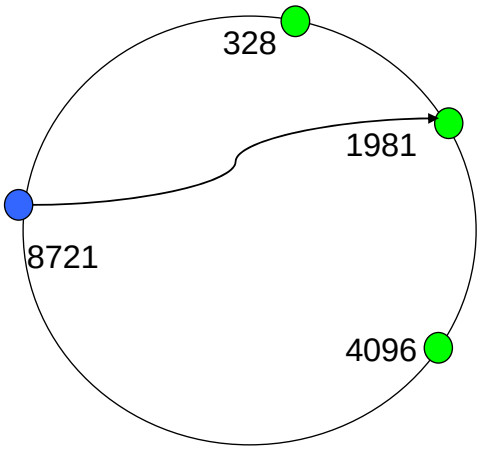


Figure 3.7.c A successfully matching example

3.6 Call Handling for Off-line Users

Recall the call flow described in Section 3.1, a call flow contains 5 steps, which are publishing, query, CallYou, Callback, and answer. Since at each step, the caller or callee may be off line, MFPGC system provides off-line user handling mechanisms to store the state of a call flow for off-line users, and resume the call flow when the users are on line.

3.6.1 Delayed query

A delayed query means a matched callee publishes after a call request queries the matched callees. We support delayed query by giving a TTL (Time-to-live) value to each query, and a CallYou message is sent to matched calless who publish after the call request but before its TTL expires. The delay query flow is displayed in Figure 3.9. To achieve delayed query we must save the query in the corresponding node, and check the query when receiving each publishing message.

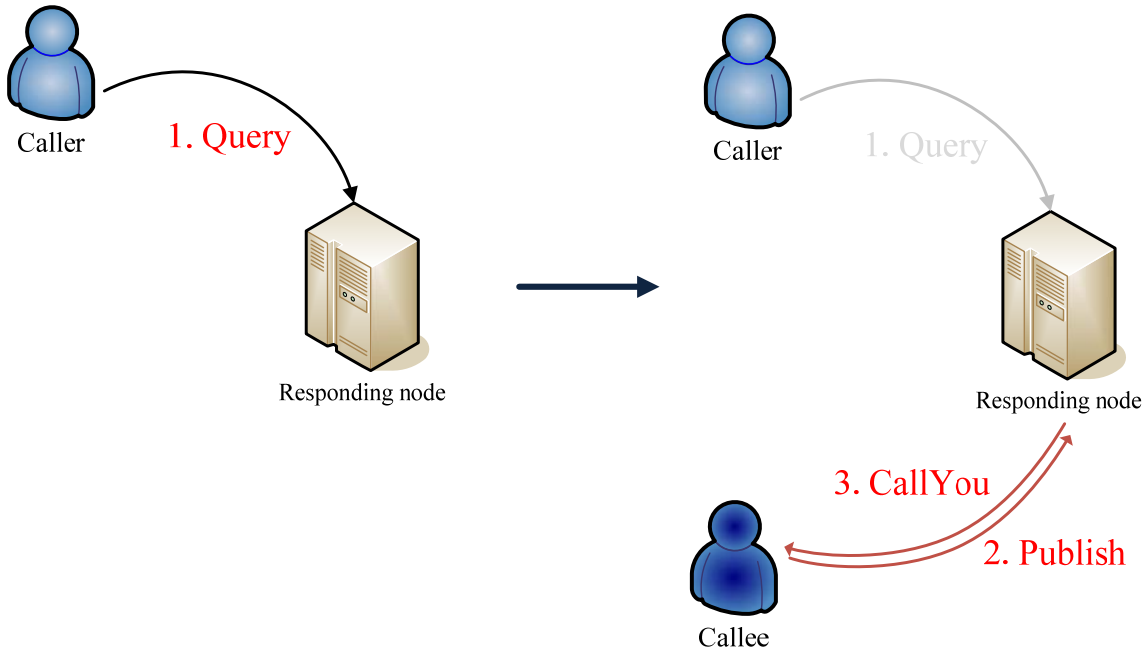


Figure 3.8 A delayed query flow

In order to identify each user, we use a profile ID for each publishing message. A profile ID is an integer number generated by hashing all attributes of that user. We assume each user would input enough specific attributes and could be identified by those attributes, so a profile ID could roughly identify a user profile in MFPGC system. After a node received a query message and compared the user profile in its local database, it saves the query and a list of the matched profile IDs until the query is expires. If a publishing profile is received during this time interval, the profile is compared with the non-expired queries and their list of matched profile IDs. A newly publishing user would receive a CallYou message from the corresponding node as depicted in Figure 3.8.

3.6.2 Delayed CallYou

A delay CallYou means a CallYou message for an off-line user is stored and forwarded to the user when he or she is on line later. We mark a user profile on-line or off-line in the corresponding node and require a “CallYouReply” message from a callee to acknowledge the “CallYou” message the callee receives. If the corresponding node

does not receive a “CallYouReply” message after sending a callyou message, the corresponding node marks the profile off-line. When the profile is marked off-line, every matched query is stored in the corresponding node. When the user of this profile becomes on line and registers the profile with the corresponding nodes, the CallYou messages will be forward to the user according to the matched queries. Figure 3.9 depicts a delayed CallYou flow.

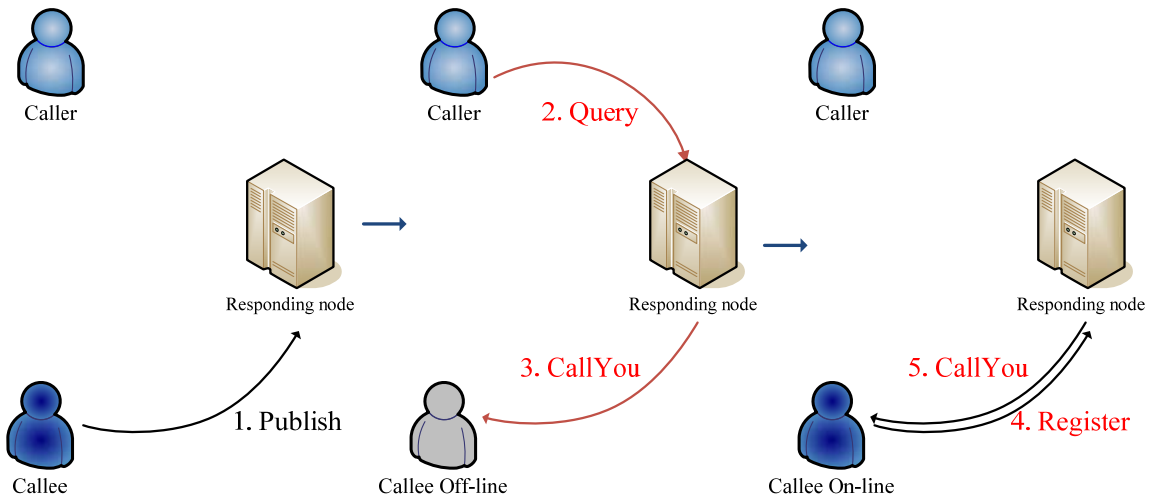


Figure 3.9 A delayed CallYou flow

3.6.3 Delayed CallBack

When a matched callee received a CallYou message and calls back to the caller, the caller may not be on line to answer the call. In this case, the callee can inform the corresponding node that the CallYou message received by the callee is invalid. The corresponding node saves the CallYou message and send another CallYou message to the callee when the caller register with the corresponding node later. To do this, we add a query ID for each query message and keep each query for a TTL interval in the local machine. Next time when the caller becomes on line, the unexpired queries are sent to the corresponding nodes with query ID, and the stored CallYou messages will be forwarded to the matched callees again. The delayed CallYou and Callback processes could repeat

until the caller and caller are on line at the same time.

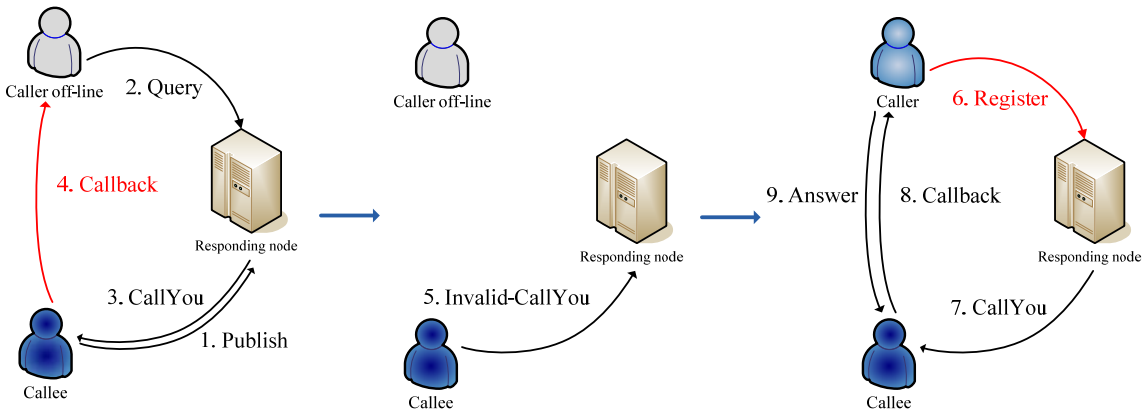
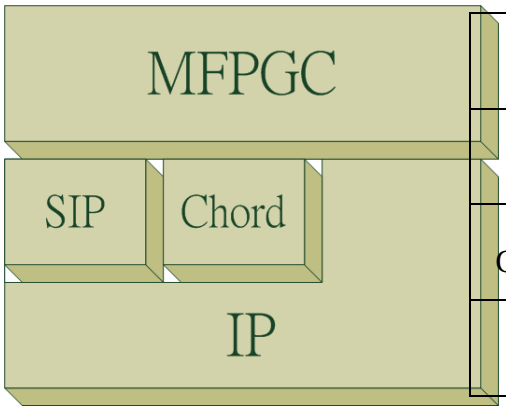


Figure 3.10 A delayed CallBack flow

Chapter 4 System Implementation

In this chapter we present our system implementation in details. Our system was developed using C++. CCLSIP UA, a SIP communication system, Chord DHT and IP were used to built MFPGC system. Figure 4.1 depicts the layered structure of MFPGC system and all MFPGC messages.



Join	Reply	ChangePre	FindPre
RePre	Publish	Query	Store
CallYou	CYReply	KAL	Re_KAL
Leave			

Figure 4.1 Layers of MFPGC system and the MFPGC messages

4.1 System components

Our system supports three types of nodes with different functionalities according to the capability of nodes; There are P2P nodes, non-P2P nodes and PDA nodes. Figure 4.2 depicts the communication modes between the components, and Table 4.2 lists their differences that we will describe in more details.

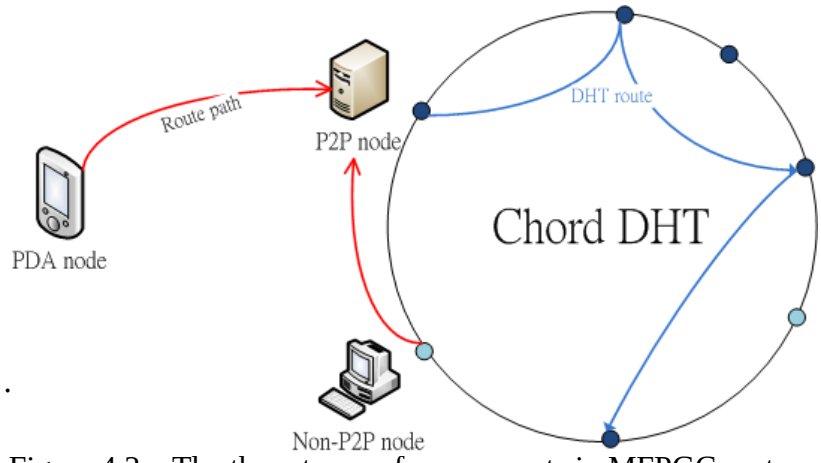


Figure 4.2 The three types of components in MFPGC system

Table 4.1 The differences between the three components

components	DHT routing	Store user profiles	Publish and query	GPS Location information
P2P node	Y	Y	Y	N
Non-P2P node	N	N	Y	N
PDA node	N	N	Y	Y

4.1.1 P2P node

P2P nodes are the back-bone of our system and provide all of the peer-to-peer functionality including routing, storing user profiles, and communication functions - publishing, query, and communications. Based on the design of Chord, a P2P node should maintain a finger table and a backup successor list for routing purpose. Each P2P node maintains a thread that periodically sends keep-alive messages to each node in its finger table, so that the node can rebuild its finer table when any node ungracefully left the Chord ring.

Each P2P node maintains another thread listening a network socket and handles MFPGC messages received. The node also maintains a database to store user profiles published by MFPGC users. When users publish their profile or attributes, MFPGC will read the attributes from database and publish them. When a P2P-node receives the published attributes of other users, it will store them in the database too.

4.1.2 Non-P2P node

Non-P2P nodes are peer-to-peer nodes without node IDs, and thus cannot provide peer-to-peer functionalities; they provide a lightweight scheme for nodes with less computing, storage, and/or networking capabilities. A non-P2P node would use a P2P node as a gateway to send MFPGC messages. The gateway P2P node of a non-P2P node could be a famous node or a node assigned by famous nodes. Since non-P2P nodes have no node ID, P2P nodes will not add non-P2P nodes to their finger tables, and thus no DHT message will be forwarded to non-P2P nodes. The only messages sent by a non-P2P node are Publish and Query which contain the IP address and port number of the non-P2P node for future communications. By using the IP information, other nodes can directly connect to non-P2P nodes without node IDs. In other words, the basic functionalities of non-P2P nodes are publishing user profiles and sending the query of a call request..

4.1.3 PDA node

PDA nodes are non-P2P nodes on PDA devices; they may be equipped with GPS functionality for location-based applications. The MFPGC system on a PDA is based on a mini SIP UA, CCL-UA, implemented on Windows Mobile platform. Note that CCL-UA was developed by CCL, ITRI. The user’s location attribute can be directly obtained from the GPS module on board. The design of PDA nodes is the same as that of non-P2P nodes.

4.2 Bloom filter implementation

As we have described in Chapter 3, MFPGC system uses Bloom filter to represent user attributes. The Bloom filter we used is a 512-bit array, and MD5 hash function is used to hash a user attribute to 8 array positions. In order to generate 8 indices, we used the same hash function but with 8 different input texts for a given user attribute. The 8 different input texts are obtained by concatenating the original text and a number of 0's. For example, consider a system-defined attribute "Club:Bridge". We hash "Club:Bridge", "Club:Bridge0", "Club:Bridge00", …, and "Club:Bridge0000000" to 8 indices and set those positions of the bit array to 1.

4.3 Message encryption

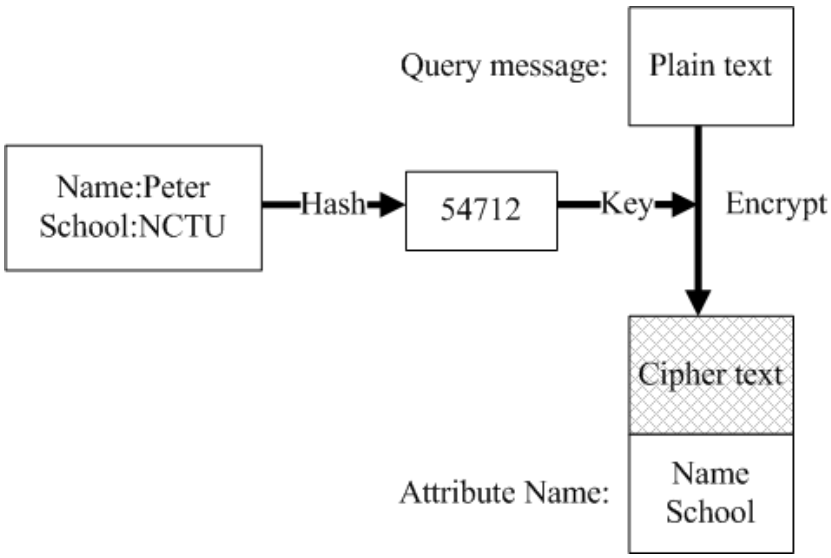


Figure 4.3 The encrypt process in query

In MFPGC system, a caller can add a short message in the query of a call request to invite the callee to call back. The Callee can read the message and decide whether to call back or not. Since a query message is first sent to the corresponding node of an attribute in SNAS, to provide confidential communications, the query message can be encrypted

using Advanced Encryption Standard (AES) algorithm. The encryption key used in AES is the hash value of the string concatenating all attribute names and values in SNAS in an increasing order of the attribute name. The query message also includes, in plain texts, all the attribute names of SNAS, but not their values. The corresponding node cannot decrypt the message since it does not have the attribute values of SNAS. When a matched callee receives the query, the callee can use the attribute names of SNAS, which is in plain texts) and the callee's own attribute values of SNAS to find the key of AES because each attribute of SNAS must be the same as the attributes of the matched callee. Furthermore, the attribute and value pairs of SAS can also be included in the query message of a call request, and SAS can also be encrypted using the same approach. In this way, a matched callee can decrypt the SAS of a query can check false positives.

Chapter 5 Performance Evaluation

The core functionalities of MFPGC system are multiple-attribute query and range query using DHT. In addition, MPFGC system used Bloom filter and encrypted message to protect user privacy and confidential communications. Many existing DHT-based systems support multiple-attribute and/or range queries, but very few of them, if any, consider data confidentiality. In this Chapter, we compare MFPGC with DHT-based systems in their efficiency supporting multiple-attribute query and/or range query. First, we introduce performance metrics used for evaluation.

Hops: The number of nodes traversed in routing a message is the most general metric to evaluate the routing overhead of P2P overlay networks. It measures the distance and time from the source to the destination. The fewer number of hops in a route results in lower communication overhead and quicker response time. If there are multiple destinations, the maximum number of hops to all destinations is used for comparison. Since a message transmitted between two nodes means an increment of hop count, the number of transmitted messages is proportional to the number of hops.

The number of nodes serving a query: To serve a multiple-attribute query, the query may be relayed to several nodes, each of which searches a certain attributes. The number of nodes serving a multiple-attribute query indicates the response time of the query. The more number of serving nodes for a query, the longer the response. Furthermore, the extra routing overhead on a P2P overlay network, this long response problem can be more severe.

Number of nodes serving a publishing: A resource described by multiple attributes may need to be published to a number of nodes. The number of nodes storing the publishing

information represents the number of messages transmitted when a resource is published. It is an indicator of the communication cost of publishing.

Storage for a publishing in a node: The size of the storage used in a corresponding node to store a published item indicates the storage cost. Note that the overall storage cost is the size of the storage in a node times the number of nodes storing the publishing information..

Hops with respect to selectivity: The selectivity is a variable in range query and implies the percent of the numeric scope that a range occupies. Bigger selectivity means bigger range. The hops might increase with increasing the selectivity in general range query systems.

We compare the four DHT systems we have presented in Chapter 2 with MFPGC system in Tables 5.1 and 5.2. Table 5.1 lists the comparison of multiple attribute query. Let *n* denote the number of nodes in the system, n_{attr} the number of attributes in a query and a publishing, *M* the size of the Bloom filter used. The term *misc.* represents the storage for information of constant length, such as, IP address, node ID, and port number. P_0 denotes the probability that a bit set to 0 in a Bloom fitler. $1-P_0(n_{attr})$ denotes the probability of a bit set to 1 after inserting N_{attr} attributes.

The results in Table 5.1 indicate that MFPGC provides comparable performance measures as MAAN. However, MFPGC provide confidential communications and protect user privacy, while MAAN does not. SCARP and MURK outperform MFPGC in term of the number of hops in routing and the number of nodes to store a published object. However, this advantage is obtained with the limitation that the set of attributes is fixed, i.e., no attribute can be added after the system is developed. Therefore, no user-defined attributes can be used. By contrast, MFPGC provides great flexibility in specifying

user-defined attributes. This flexibility is very important for the communication scenarios targeted by MFPGC.

Table 5.1 The comparison between five systems supporting multiple attribute query

class	**Multiple attribute query**			
Metrics	*Number of nodes for a query*	*Hops*	*Number of nodes for a publish*	*Storage in a node for a publish*
MFPGC	1	$\log(n)$	n_{attr}	$M + misc.$
SCARP	1	$\log(n)$	1	$n_{attr} + misc.$
MURK	1	$\log^2(n)$	1	$n_{attr} + misc.$
MAAN	1	$\log(n)$	n_{attr}	$n_{attr} + misc.$

Table 5.2 displays the comparison for range query. We only compare MFPGC and MAAN because MKey does not support range query. The performance of SCARP and MARK on range query varies by a large degree with different sizes of the range, and it is difficult to obtain the performance metrics in a closed form. Let *s* denotes the selectivity of a range attribute, and $s_{\min}$ the minimum selectivity of the ranges in a query. $R_{\max}$ and $R_{\min}$ denote the lower bound and the upper bound of the domain of the numeric attribute. *level* denotes the size to dividing the range in MFPGC system. The rightmost column lists the computation complexity when the responding node compares a query with its maintained data, where n_t denotes the number of records that have the same resource ID as the query. The results in Table 5.2 clearly indicate MFPGC provide better or comparable performance measures in all metrics.

Table 5.2 The comparison of five systems supporting range query

class	**Single attribute range query**	**Multiple attribute range query**		**computation**
Metrics	*Hops with respect to selectivity*	*Hops with respect to selectivity*	*Number of nodes for search*	*Match a query in one node*
MFPGC	$\frac{s\times(R_{\max}-R_{\min})}{level}\times\log(n)$	$\frac{s\times(R_{\max}-R_{\min})}{level}\times\log(n)$	$\frac{s\times(R_{\max}-R_{\min})}{level}$	$n_t \times M$
SCARP				$n_t \times n_{attr}$
MURK				$n_t \times n_{attr}$
MAAN	$n \times s$	$\log(n)+n\times s_{\min}$	$n \times s_{\min}$	$n_t \times n_{attr}$

Chapter 6 Conclusions

Traditional communications, such as telephony, email and VoIP, use a specific ID to specify the callee. In this thesis we design a novel communication system using user attributes to specify the callee(s). Communication is possible even if the callee's ID is unknown. Chord and Bloom filter have been used to publish and match user attributes. By using Bloom filter to represent user attributes and encrypting the communication messages, user privacy has been protected; only a matched callee can receive and decrypt the caller's message. To support necessary attributes specified by the caller and the callee, two Bloom filters, one for necessary attributes and the other for all attributes, were used in publishing a user profile, and in querying matched callees. MFPGC system also provides off-line user handling mechanisms to store communication messages of off-line users, and to forward the messages the user when he or she is on line.

For the corresponding nodes of hot keywords, the loading in networking, storage, and computation may dramatically increase as the number of users increase. An efficient load balancing mechanism for our system is critical to deploy MFPGC system in a large scale. The mechanism should consider hot keyword attributes and improve the level dividing method for range query. Another limitation of MFPGS is a sender-specified necessary range attribute was not supported and this could result in complex query process. An efficient solution is needed for this problem.

Reference

[1] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, p.149-160, August 2001, San Diego, California, United States.

[2] Burton H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, v.13 n.7, p.422-426, July 1970

[3] “Napster.” http://www.napster.com/

[4] “Gnutella.” http://gnutella.wego.com.

[5] A. Rowstron and P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems,” Lecture Notes in Computer Science, Vol. 2218, 2001.

[6] B. Zhao, J. Kubiatowicz and A. Joseph, “Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing,” Technical Report UCB/CSD-01-1141, 2001.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. “A scalable content-addressable network.” In Proceedings of the 2001 ACM SIGCOMM, pages 161-172.

[8] P. Reynolds, A. Vahdat, Efficient peer-to-peer keyword searching, in: ACM/IFP/USENIX Int’l Middleware Conference, Middleware 2003,.

[9] Lintao Liu, Kyung Dong Ryu, and Kang-Won Lee.Keyword fusion to support ecient keyword-based search in peer-to-peer le sharing. In 4th Int Work-shop on Global and P2P Computing (GP2PC in con-junction with IEEE/ACM CCGRID), Chicago IL,April 2004.

[10] Min Cai, Martin Frank, Jinbo Chen, Pedro Szekely, “MAAN: A Multi-Attribute Addressable Network for Grid Information Services,” Proceedings of the Fourth International Workshop on Grid Computing, p.184, November 17-17, 2003

[11] A. Bharambe, M. Agrawal, and S. Seshan. “Mercury: Supporting scalable multi-attribute range queries.” In Proc. SIGCOMM, 2004.

[12] Cristina Schmidt, and Manish Parashar, "EnablingFlexible Queries with Guarantees in P2P Systems,"IEEE Internet Computing, Vol. 8, No. 3, pp. 19-26,May/June 2004.

[13] Prasanna Ganesan, Beverly Yang, Hector GarciaMolina,"One Torus to Rule them All: Multidimensional Queries in P2P Systems," Proc. WebDB’04, Paris, France, 2004.