國立交通大學

網路工程研究所

碩 士 論 文

使用布隆過濾器之多功能個人/團體通訊系統

Multi-Function Personal/Group Communication System

with Bloom Filter

研 究 生：李忠育

指導教授：張明峰　教授

中 華 民 國　九 十 七　年 七 月

使用布隆過濾器之多功能個人/團體通訊系統

Multi-Function Personal/Group Communication System with Bloom Filter

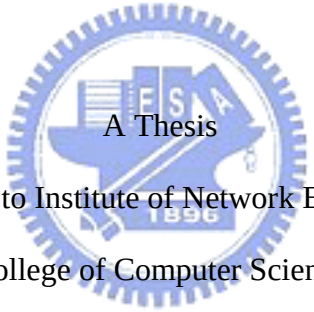
研 究 生：李忠育　　Student：Chung-Yu Li

指導教授：張明峰　　Advisor：Ming-Feng Chang

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文



A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

使用布隆過濾器之功能個人/團體通訊系統

學生：李忠育　　指導教授：張明峰 博士

國立交通大學網路工程研究所

摘要

現今的通訊系統例如電信系統、即時通訊、電子郵件、和網路電話等，都利用特定識別元來辨識使用者。然而還可以透過使用者的各項屬性來便是指用者例如姓名、年紀等等。一個屬性的集合也可以視為更明確的辨識元並具有好記、有意義和具代表性的優點。我們提出一個系統同時支援特定辨識元和多屬性來通訊。透過發佈和搜尋的機制，一個使用者可以很快的和想要找的人通訊。

MFPGC 架構在 Chord 之上並且使用布隆過濾器來儲存多屬性的資料，包含字串，數值和混合型態的屬性。另外讓發佈者和接收者可以設定必須要符合的屬性以用來過濾不想要接收的訊息，同時不會因為額外的條件增加搜尋的方便性。離線使用者處理機制處理當使用者在各個狀態中離線，而通話還是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 提供使用者間的通話情境更高的彈性。

Multi-Function Personal/Group Communication System with Bloom Filter

Student: Chu-Yu Li Advisor: Dr. Ming-Feng Chang

Institute of Network Engineering

National Chiao Tung University

ABSTRACT

Communication services today, such as telephony, instant message, email, and VoIP, use a specific user or device ID to specify the called party. Another way to indicate the callee(s) is to specify the callee's attributes, such as the name, the age, etc. A set of user attributes, which is meaningful, rememberable, and representable, can be used to indicate the callee. Multi-Function Personal/Group Communication (MFPGC) system supports communications using both specific IDs and multiple under-specified attributes. Communications using multiple under-specified attributes is feasible through publishing and querying users' attributes on DHT. The DHT of MFPGC system is based on Chord and Bloom filter is used to represent user attributes, which can be string, numerical, and hybrid data. Communications are set up by matching the Bloom filters of callers and callees. 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 messages for off-line users. Compared to traditional communication system, MFPGC system provides more flexible scenarios for users.

誌謝

一開始我要感謝我的指導老師，張明峰教授。教授仔細而耐心的指導，讓我在實作和閱讀中學會獨立思考與研究的方法；在構思本論文時，也多次指正我思考的盲點，才能順利完成這篇論文。我很感謝在求學過程中，能得到張教授的指導，研究所這兩年獲益匪淺。

接下來我要感謝實驗室的同仁，冠璋同學、坤揚同學、君飛同學，和我一起修課奮鬥，一起熬夜寫程式，一起打桌球，同甘共苦，讓我兩年過得一點也不孤單。也感謝玄亞、威凱、柏瑞等學弟，讓我這段時間的生活增添不少色彩，很高興能認識你們。最後要感謝我親愛的家人，由於你們精神和經濟上的支持，讓我可以在求學的過程中一路順坦，沒有後顧之憂的完成學業。

Tables of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Current development

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 are cheaper charging rate and more powerful functionalities of the services. However, the PLMN (Public Land Mobile Network) system still plays an important role in our daily life, so most of those communication system developers have made efforts to integrate their systems into mobile devices for user convenience, or even integrate their systems with the PSTN (Public Switching Telephone Network).

In recent years, peer-to-peer technology has been one of the most popular technologies employed in Internet applications. More and more applications using a peer-to-peer overlay network for information multicasting, object searching, and load balancing. Those functionalities can be provided efficiently in P2P architecture. P2P file sharing systems and real-time streaming video services are the most popular applications on the Internet and consume most of the Internet bandwidth in recent years. However, P2P systems still have some shortcomings with it. For example, security is an important issue. Because centralized server does not exist in P2P systems, how to trust other nodes is an unsolved problem. Another security problem is user/device authentication. Because of un-trusted nodes, user/device authentication without a trusted server is almost impossible to achieve; this bottleneck limits P2P in commercial applications. More and more researches toward robust and reliable P2P architecture, but the most popular applications are still based on anonymity.

General speaking, a peer-to-peer system is used for storing data, and supporting efficient publishing and searching mechanisms. Most of the current peer-to-peer researches focus on improving the efficiency of routing and searching under various conditions. Different assumptions of the network environments, like churn handling and location awareness, result in different approaches to optimizing the system. Searching is one of the most important functionalities provided by peer-to-peer systems. Contrast to centralized servers, peer-to-peer systems store data in each peer node. The overhead of communications on network replaces the overhead of database access on the central server, and becomes a critical bottleneck under complex network conditions. Furthermore, due to the existence of unstable nodes, data synchronization problem makes data in peer-to-peer unreliable. Although storing multiple replicas of data in more nodes may decrease this disadvantage by adding endurable overhead, searching in the peer-to-peer network is usually considered as a best-effort function. Another extend problem is load balancing. Because hot keywords dominate major part of the searching load, the responding nodes of those keywords in the peer-to-peer network may cause heavy performance overhead. Most researchers have attempted to break the continuation of hot keywords, but single hot keywords couldn't have been completely resolved until now.

As we know, the information retrieval technology has wide influence on network, and semantic keyword searching is the foundational part for information retrieval. Although semantic searching is hardly implemented in peer-to-peer network because of the property of distribution, there are some researches about how to achieve more complex searches in peer-to-peer network, such as numerical query and and/or query. With the popularity of peer-to-peer technology, researches of searching algorithm will be a new trend in network.

1.2 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. In the beginning, the naming of user/device ID was restricted by device capability in order to simply implementation. However, with more powerful user devices, there are less constrains on the naming; IDs that are more meaningful and representable can be used. The trend of user/device ID is toward meaningful, representable, distinct and rememberable.

Although a specific ID can uniquely specify a user, it would be very useful if we can initiate a communication with a callee(s) without knowing the callee's specific ID. For example, Billy has graduated from NCTU 20 years ago, and some day he wants to hold a reunion, but the contact methods of some classmates have been invalid. Since using specific ID has drawbacks of record invalidity and the troubles to update the record, a more intuitive communication method can be useful.

One way to indicate the callee(s) of a communication is to specify the callee's attributes, such as the name, the age, and the school he or she studies, etc. A user is associated with a set of attributes. In another point of view, a set of user attributes is a kind of powerful user ID that is rememberable, meaningful, and representable. Although attributes often lack for distinguishability and convenience, communication with this kind of ID provides an alternative way to supply more semantic callee.

To set up a communication with specified callee's attributes, we need to do multi-attribute data matching between the call request and the users' profiles. An intuitional idea is caller specify some attributes as ID, then any callee(s) have those

attributes will receive the call request. The client-server architecture with database is a direct and simple way to implement the functionality. However, peer-to-peer technology has been widely extended in multiple attributes query by researchers. It provides an alternative way to client-server architecture with better scalability and without single point of failure. Therefore it is a better choice for communication systems which support stronger searching function to adopt peer-to-peer as the backbone platform.

Even though using peer-to-peer architecture in communications is a suitable solution for next generation communication systems, there are some bottlenecks impeding the trend. First problem is performance, a link in overlay network may be an unexpected long path in the physical network, and routing in overlay network may cost more time than the systems using direct connection. Furthermore, in recent proposed methods of multiple attribute matching, the routing overhead linearly increases with the number of attributes. Lacks of a efficient publish/query mechanism for peer-to-peer leads to the bottleneck of revolutionary communication systems developed in peer-to-peer network.

Another ignored point is how to match the caller and callee's intension for talking to each other. Most systems emphasized on the demand of caller, which caused several critical problems like junk mail or ad; and further effected users whether they would choose this system or not. Advanced communication systems should contain filtering mechanism for callee, so as to reduce unnecessary messages and transmission costs.

1.3 Objective

Our main goal is to build a communication system with the following features:

1. Support communications using specific ID and unspecific ID attributes.

2. Flexible attributes for callers and callees including numerical attributes.

3. Match the desires of both the caller and the callee, and filter unwanted call requests.

4. Efficient routing in peer-to-peer and endurable overhead to maintain other users' queries and publishes.

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

6. Flexible off-line handling mechanism in every states of call flow.

In order to implement such a communication system, we adopted structured peer-to-peer architecture or DHT (distributed hash table) as platform and proposed a novel publish/query mechanism to accomplish above requirements, and we will describe that in details in later chapters.

1.4 Summary

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

Chapter 2 Related Work

In this chapter we describe recent peer-to-peer researches, including Chord [1], DHT, which we used as the under-layer routing platform, and other P2P systems that support range query or multiple-attribute query, or both. Although those systems have distinctive features, we will explain their limitations that do not fully satisfy the requirements of our system. In addition, we will describe the design of Bloom filter, which is a space-efficient, randomized data structure representing a data set. Bloom filter [2] is the core of user data publication and query in our system. We will also describe systems that use Bloom filter for multiple-attribute query.

2.1 Chord and DHT routing

In the early stage of peer-to-peer development, there were two ways to locate resources. In one way, resource indexing and searching was performed by centralized servers and resource sharing was directly operated between peers. However single point of failure may imperil this mode. The most representable system of this mode was Nasper [3]. In another way, resource searching was done by flooding the resource requests to peers. Each node randomly records some nodes as neighbors and maintains direct connection with the neighbors; Gnutella is a typical example of this mode [4]. However, flooding would cause mass redundant messages and inefficient usage of network bandwidth. It is clear that the aforementioned two ways of peer-to-peer 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 were not popular until the invention of distributed hash table (DHT). DHT takes advantage of consistent hash to locate resources in an

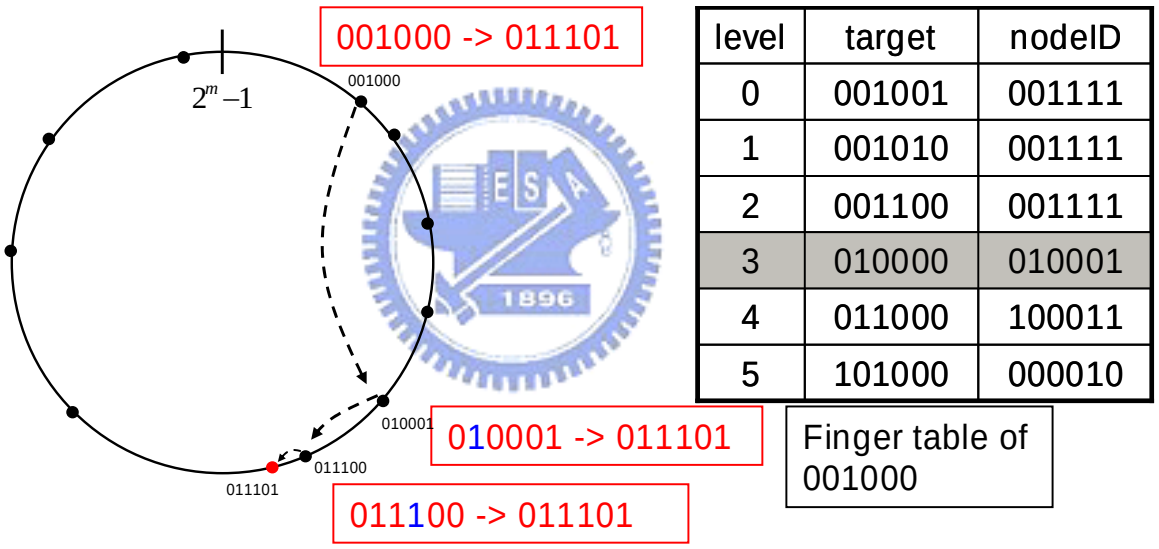
overlay network. By using consistent hash, nodes and resources hash themselves to IDs, which is usually longer than 128 bits. A resource registers its location with the corresponding node of the same ID. Anyone can find the location of a resource by using the same hash function to determine the corresponding node of the resource. Due to the property of consistent hash, the corresponding node can be uniquely determined in the overlay network. By requesting the corresponding node one can obtain the location of the needed resource. The efficiency, load balancing, and complete distribution properties of DHT were so powerful that made DHT become the major research target of recent peer-to-peer network.

2.1.2 Chord

Although DHT is the basic idea of current structure peer-to-peer, there are still problems to be solved, for example, how to route a message to a node of a particular ID in the overlay network, how to handle ungracefully leaving or churn, and how to join the overlay network. After DHT had been proposed, many researchers designed various kinds of routing algorithms, such as 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 algorithm, and used as the backbone platform of our system. Chord works as follows. First, Chord maps each node ID from 0 to $2^m - 1$ to the ring, where m is the scale of DHT. Similar to linked list, each node saves his successor or neighbor in the ring. The major functionality of Chord, routing to a particular node ID, could be treated as simply traversing the ring. However, by adding finger table to keep extra node information, the routing could achieve in $O(\log n)$ time.

The finger table is a structure to save additional nodes which nodes ID are successor of $(2^i + k)\%2^m$, where k is own node ID and i is the number from 0 to m-1. When a node generates a query, it first looks up finger table to search an ID small than the target ID then forward the query. This greedy algorithm will pass the query to the closest ID smaller than target ID, by the definition of consistent hash the successor of that ID is the responding node of the target ID. Figure 2.1 shows a route from node 001000 to node 011101, and the level 3 of the finger table of node 001000 is the approximate ID to the destination. The node thus forwards the message to node 010001 and nodes in Chord will repeat the process until the message reaches the most approximate node.



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

Finger table of 001000

Figure 2.1 Left side is a routing example, and right side is a finger table example.

Each node of Chord ring maintains $O(\log n)$ nodes information and a message can be routed to a particular ID in $O(\log n)$ hops. Because a successful query needs every node in routing path to forward to existing nodes, the correctness of finger table must be kept especially the successor. Chord has stabilization mechanism to handle ungracefully leaving and join of new node. In a period of time, each node ping the successor and ask whether his predecessor has been changed. If so, the node sets his new successor as the old successor's new predecessor and finishes joining process. Nodes also maintain

several backup successors which are used when the successor disappear. The stabilization mechanism provides reparation in churn and works well with high probability. The nodes in finger table also need to be checked periodically by fix finger mechanism. If any item does not reply the keep-alive message then rebuild the finger of that target ID. These mechanisms make Chord stable during serious churn and provide more reliable publish and query result for upper layer applications.

2.2 Multiple attributes

The most important functionality of DHT is searching. By using consistent hash function, any participant in DHT network could locate a resource without server. Just find the responding node by hashing resource to a constant sized ID, the resource information is kept at that node. Because of the nearly perfect distribution property, DHT became the major research domain about peer-to-peer network.

The multiple attribute search problem is known as “and” operation, that is, a query carries several attributes, and items that own all of those attributes match the query. In ordinary DHT design, every attribute will be hashed to different node, so a multiple attributes query can’t be handled in one node, and even the overhead of DHT routing will lead to low efficiency in multiple attributes query. Because multiple attributes is one of the basic operation in complex query, many researches toward reliable and efficient multiple attributes query have been proposed recently.

The most intuitive idea of multiple-attribute query is iterative search. The querying node first queries an attribute of the set to the responding node, then saves the results in the query and forwards query to responding node of next attribute iteratively. Besides the first queried node, the remainder responding node only needs to intersect the list in query

and its own and filter unmatched results, and the responding node of last attribute would reply the final result back to the querying node. This is the simplest iterative way to achieve multiple attributes query.

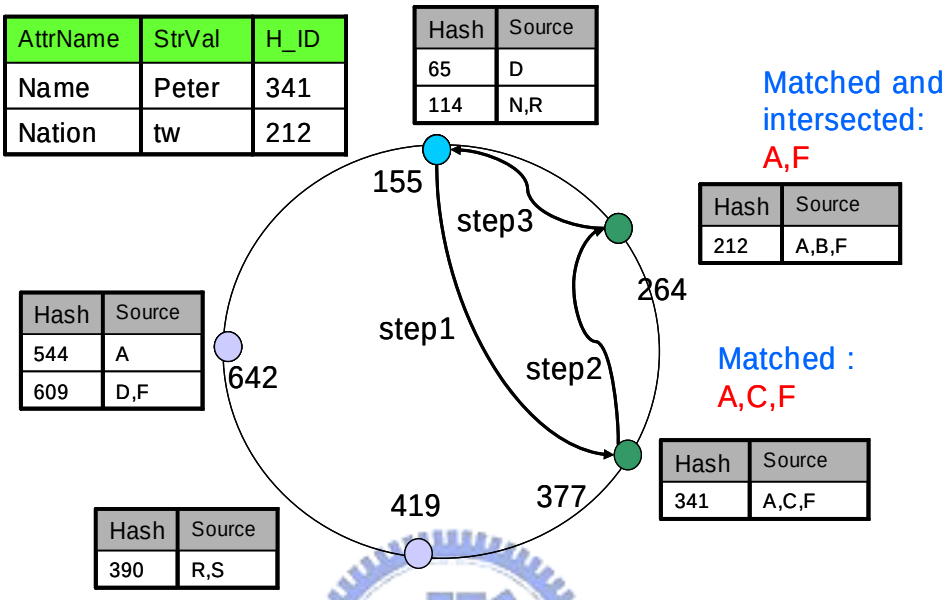


Figure 2.2 Iterative search for multi-attribute query.

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

However, the indexing data which match an attribute may be too large to be transmitted during iterative query. Reynolds [8] proposed that transmitting the Bloom filter instead of the data will simplify the query process. Bloom filter is a data compression method, and multiple data could be compressed to an array with possibility of false positive. The array of bloom filter could be dynamically adjusted so much smaller the data. We will describe this algorithm in detail in next chapter. Figure 2.3 shows a query example; client is the querying node and queries the attributes $A \cap B$. The

left picture is ordinary iterative query and the right side is transmitting by bloom filter. The querying node first forwards the query to the responding node of A, that is, SA. SA hash the matched item to bloom filter F(A), then forward F(A) to SB doing intersect and iteratively repeat the process until all attributes are searched. Due to the false positive problem, the final result of the query should be transmitted backward in reverse order and checked twice. And the right side query causes error of false positive in 6 of {3,4,6}; however this error will be corrected during backward check. Although using bloom filter reduces the transmission overhead, yet the additional check might cause another overhead.

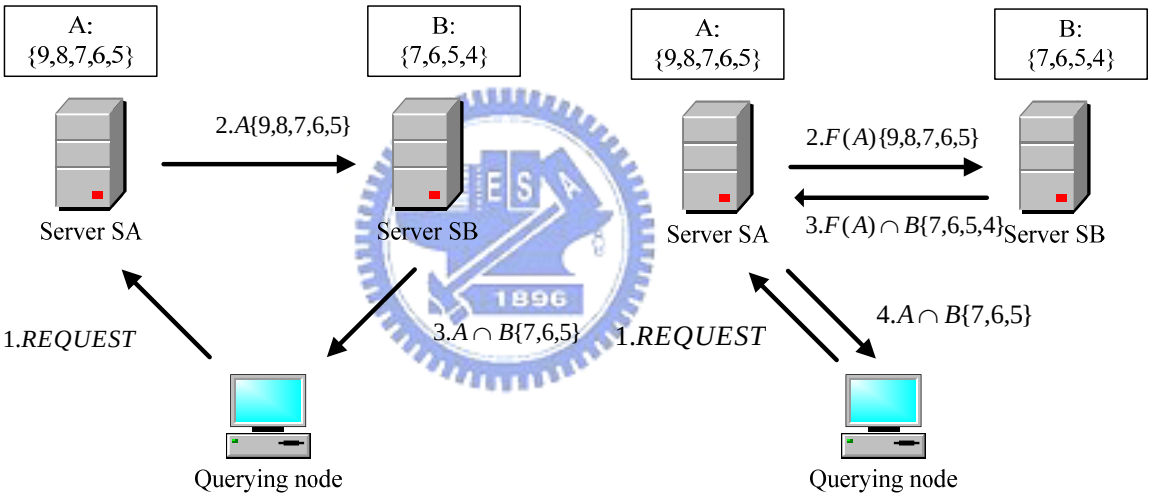


Figure 2.3 Relay search and Bloom filter used in relay search.

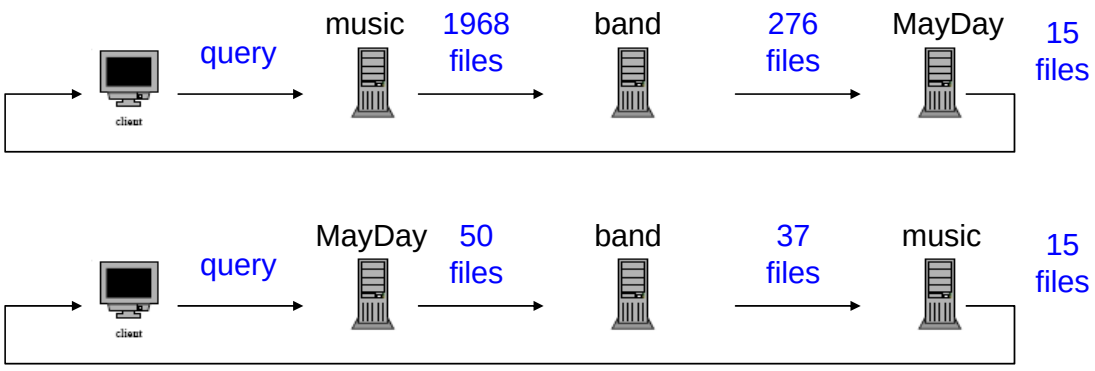


Figure 2.4: A query example of different query order.

Lintao [9] observed that if we query more specified attributes first, we could reduce

the transmission during the iterative query. The specified attributes means few users have the attribute, so a query would match fewer items. They also proposed fusion dictionary mechanisms to dynamically detect hot keywords and put those keywords in the last of query order. Each node maintains fusion dictionary as a file and updates by responding nodes of keywords flooding to every node. Furthermore, keywords fusion mechanism merges two hot keywords as one less hot keyword to reduce query process, and merged keyword can be added to the fusion dictionary to return hot. Figure 2.4 show the query process with fusion dictionary, the upper side query the hot keyword first and the below side query the specified keyword first, we could obviously observe the difference of transmission data. However, this algorithm would pay additional overhead to maintain and synchronize the dictionary in each node.

2.3 Range query

Similar to multiple-attribute query, range query is also a searching problem in DHT. Consistent hash functions break the locality of attributes for load balancing, that is, no matter how similar keywords are the result after hashing will be far distributed. However, many kinds of queries might contain near attributes such as a range of numbers. There were some researches toward efficient range query. Most of them used location preserving hash function and provided another way for load balancing.

MAAN [10] supports both multiple attributes and range query based on Chord. It provides uniform location preserving hash for range query and single attribute dominated query for multiple attributes query. Location preserving hash used by MAAN simply linearly maps the numerical range to the namespace of Chord ID. The minimum of the numerical attribute maps to 0 and the maximum maps to $2^m - 1$ in Chord. Although the hash had load balancing problem, the author claimed that MAAN won't suffer load

unbalance by using the distribution of numerical attributes to map instead of linearly mapping. Single attribute dominated query is using one attribute to query instead of querying all attributes iteratively. In order to achieve the goal, MAAN must publish with all attributes and save all the attributes information in responding nodes of all attributes. Therefore q query could compare all attributes in one of the responding node. Single attribute dominated query will suffer privacy problem and additional storage.

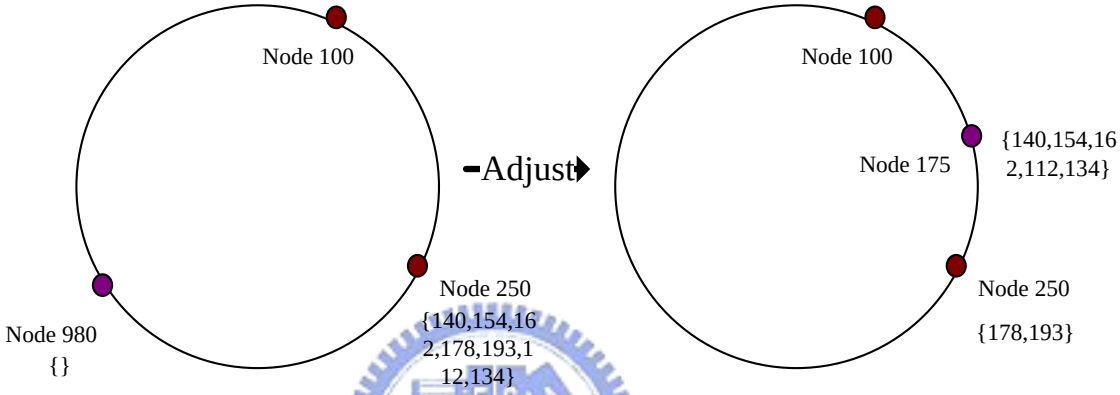


Figure 2.5 The load balance mechanism in Mercury.

Mercury [11] adopted single attribute dominated query and location preserving hash, and further designed a dynamically load detecting and balancing mechanism that can effectively solve the load balancing problem. First, Mercury node randomly chooses a neighbor to send a “load probing” packet. The “load probing” packet randomly and repeatedly forwards to neighbor and records their load. The “load probing” packet also contains pre-defined TTL value which decreases when it arrive a new node. While the TTL value returns to zero, the final node sends back to the sender for collecting load. If a node observe that its own load heavier than the collected load over a constant threshold, it sends a “light probing” packet to search a light load node, and then ask that node to gracefully leave and rejoin to the heavier position. According to the property of consistent hash, the heavy load would be shared by the two nodes. In Figure 2.5 the node 980 rejoin to node 175 and share the load between 100 and 250. Periodically executing this process

makes the load of every node to be balanced.

Another approach of multiple attributes and range query is using space filling curve (SFC). SQUID [12] and SCARP [13] transforms multiple attributes range query into multidimensional query. Each dimension represents a numerical attributes, and strings are treated as ASCII number. SQUID also uses SFC to map multi-dimension into one dimension line and location preserving hash to map the line into Chord. Therefore the query could be handled in one node to reduce the complexity. Figure 2.6 shows the 2D example using SFC to map a 2D plane to a 1D line, and the right side is mapping the range X:0~1 Y:1~3 in the line.

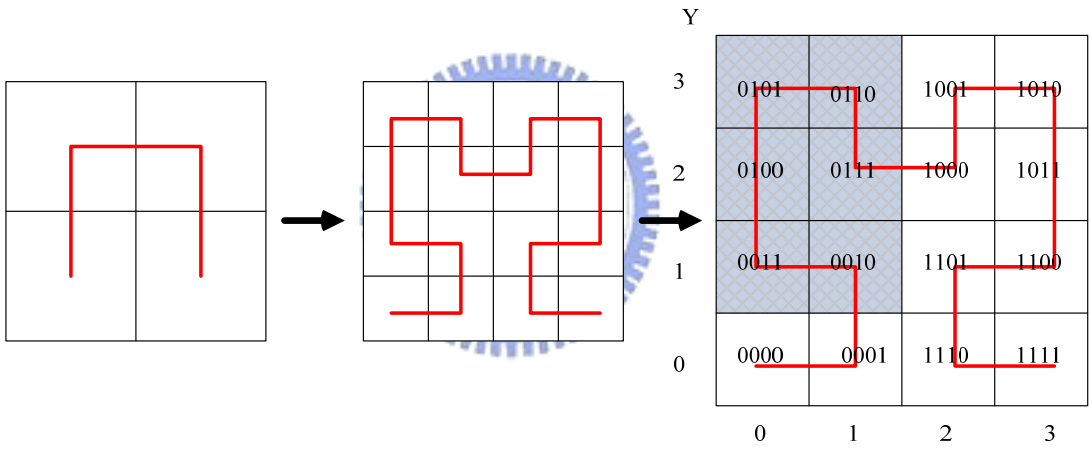


Figure 2.6 A space filling curve example

However, range search in SFC will generate fragmentation because a contiguous range in high dimension does no surely map to a continuous segment in the line, especially in higher dimension. As a result, range query using SFC only suits to fixed and lower dimension such as longitude and latitude information.

2.4 Bloom filter

Bloom filter is a space-efficient data structure for representing a data set. It support insert attributes and check whether a certain attribute is in the set. A bloom filter contains

an *m* bit array and *k* independent hash functions where *m* is size of bloom filter and *k* is number of hash functions used in bloom filter. The insertion process is as follow: First, set each bit of the *m* bits array to zero, then use *k* hash function to hash 1 attribute into *k* integers $h_1, h_2 \ldots h_k$ which range between 0 and *m*, and then set the bits in position $h_1, h_2 \ldots h_k$ to 1, that is, *k* bits is set to 1 and the other is set to 0. Repeat above step for each attribute until all attributes are inserted into the *m* bits array, and the bloom filter could check whether an attribute is in the set by using this array. The checking process is 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, then the testee is in the attributes set. Bloom filter has very efficient insertion and checking method and using constant bit array for multiple attributes also provides good space utilization.

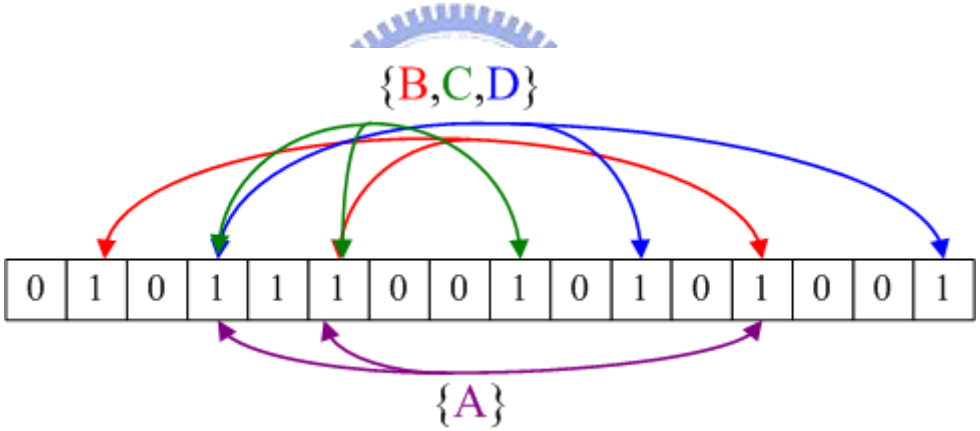


Figure 2.7 Multiple attributes with Bloom filter

However, the most serious hazard of bloom filter is false positive, that is, although all positions of attribute A's hash value are set to 1, it is still possible that A does not belong to the attributes set. The position might be set to 1 by another attribute hashing to exactly the same value. The attribute A in Figure 2.7 is an example of false positive, the hash values of A respectively collide the hash values of attribute B, C, D. General speaking, the more bits set to 1 in the bloom filter, the more possibility of false positive. Furthermore, the number of attributes *n* which insert to the bloom filter, the array size *m*,

and the number of hash function k is factors of the false positive probability. So applications using bloom filter must adjust those parameters to achieve endurable false positive probability, usually below than 1%, and also provide additional examination method to handle false positive.

In spite of false positive, bloom filter has very good performance in terms of space and computation, and thus has been widely used in network. Our communication system also adopted bloom filter to store data, and we will describe in next chapter.

Chapter 3 System Design

In this chapter we describe MFPGC system design in details, including the system operations , such as, publish or query, the algorithm to store and compare the user data, and how to find suitable objects in the peer-to-peer network. We also provide a flexible mechanism to handle churn during a calling procedure.

3.1 System overview

Our system adopts Chord as the application layer routing method and SIP as the communication protocol. The Fig 3.1 depicts that our system operates on top of Chord ring, and mobile users connect other through P2P users. Users communicate to each other by SIP UA.

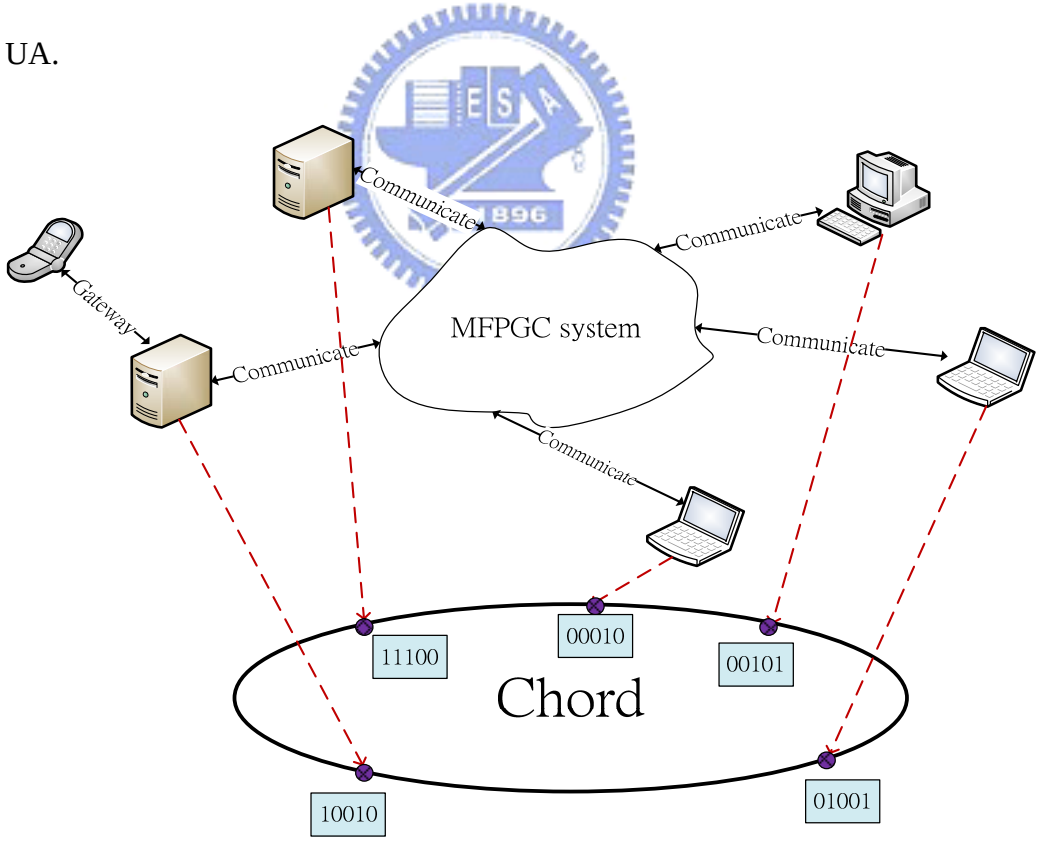


Figure 3.1 System Architecture Overview.

The routing on DHT follows the design of original Chord and thus we will not present the details in this paper. All we have used in our system is publish and query

mechanism provided by Chord and our system does not make changes in routing. Although we used Chord as under-layer routing protocol, actually we could implement our system on any DHT-based routing protocol, if they support single-attribute publish and query.

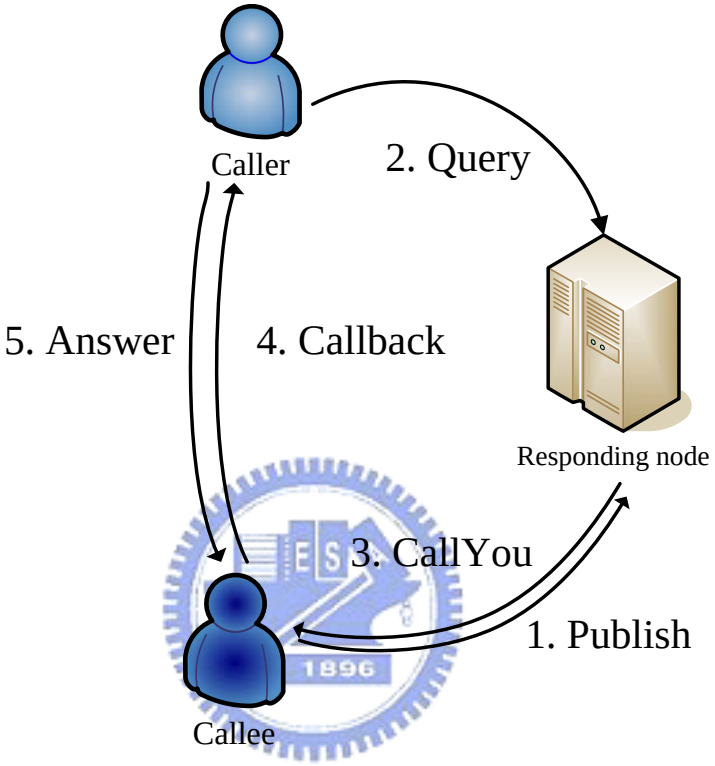


Figure 3.2 The call flow of MFPGC system

Because a communication using specific ID is a conventional SIP call, we only present the unspecific ID communication model in MFPGC system which is achieved using attributes. The communication flow is 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 those attributes maintain a copy of the user information

2. A caller initiates a query to call the users who he wants to find, and the query will be forwarded to the responding node of one of the attributes in the query.

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

4. One of the callee receives the call request and decides to reply the call. The callback message returns to the caller.

5. The caller answers the call and starts the communication with the callee.

The five steps is the basic call model of our system. However, the attributes contain numerical, string, and hybrid type and we have different handling method described in next section.

3.2 Publish with Bloom filter

The most notable feature in 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, so we use Bloom filter to store multiple attributes and to match queries with user profiles.

When a user registers the user's attributes in MFPGC system, first the user would fill their attributes into a profile stored in the local database of the user's device. The attributes can have different properties such as numerical, string, or hybrid type and system defined, or user defined. And then system will calculate the responding nodes of each attributes he has filled in the profile by SHA1 hash function. After the user presses publish button, system sends publish messages to those responding nodes. MFPGC system should publish to every attribute so that users could just query one of those attributes to compare all.

Instead of publishing the attributes themselves in clear test, our system publishes the Bloom filter of those attributes. Each published message includes the user contact

information such as IP address, port, etc, and a Bloom filter containing all attributes. Figure 3.3 and 3.4 depicts the whole publish process of Peter's attributes

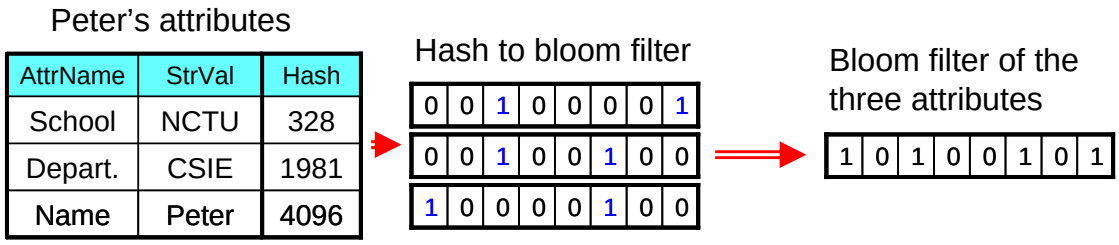


Figure 3.3 An example that maps three attributes of a profile into a Bloom filter.

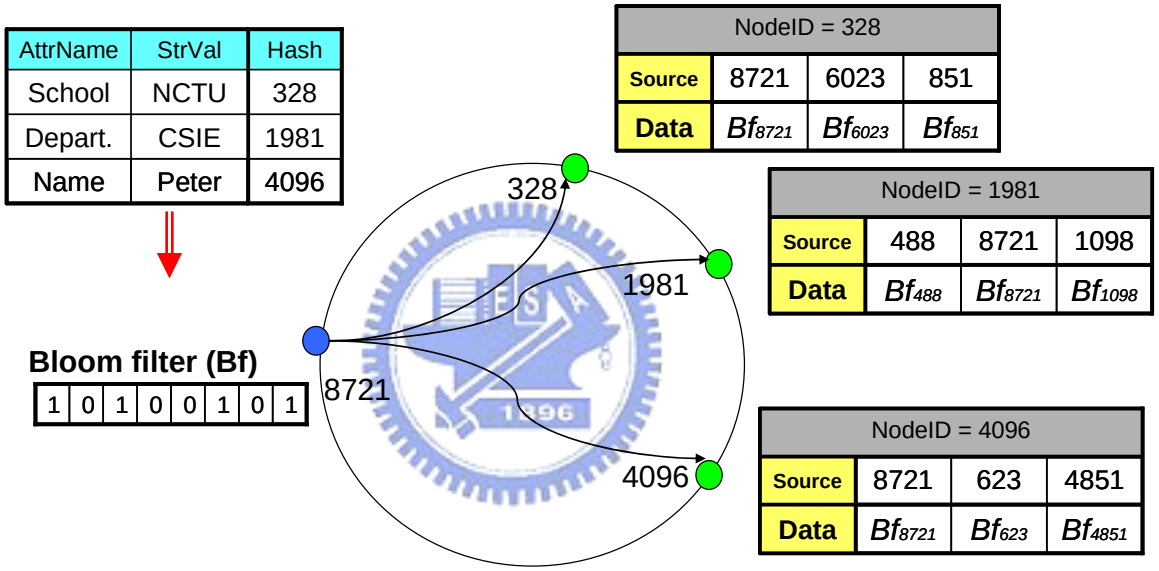


Figure 3.4 The publish process of the profile in Figure 3.3.

3.3 Query with Bloom filter

Query is processing when a user would like to communicate with another user who has certain attributes. Similar to publish, the query process first maps the attributes which the user fills in into a Bloom filter, and then sends the query including the Bloom filter to one of the responding node of those attributes. The most significant difference between publish and query is that publish needs to be sent to all of the attributes but query only needs to be sent to one. Because the user information has been published to all

responding nodes, the query could easily match the Bloom filters in one of those nodes. Furthermore, if we could choose a more specific attribute to query, we may balance the query load in DHT network; however, MFPGC system currently still lacks for the mechanism dynamically detecting specific attributes.

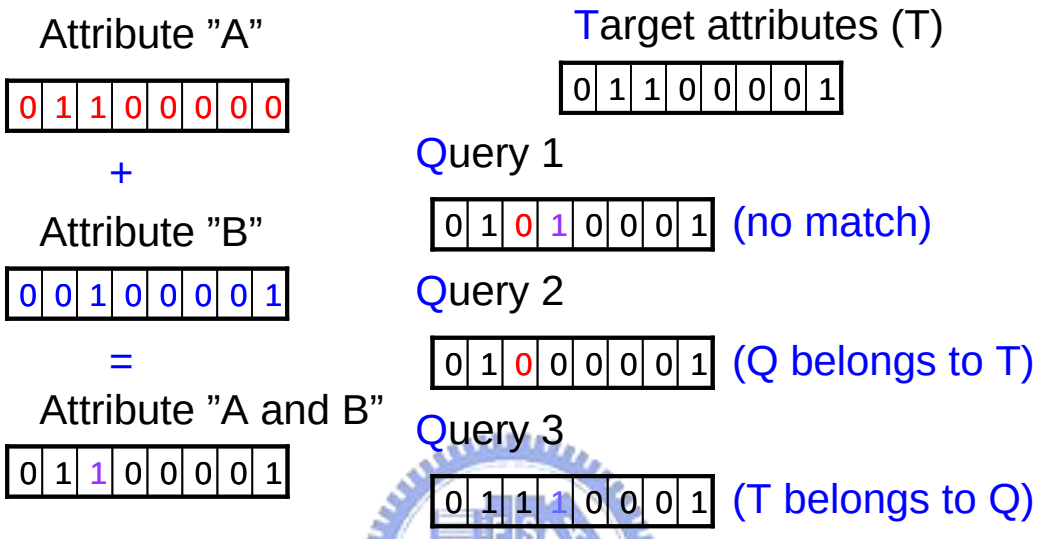


Figure 3.5 A Bloom filter matching example.

After query message was sent to the responding node, system compares the Bloom filter in the query with the Bloom filters in the local database which maintains publishes of other nodes. We represents an example of Bloom filter matching in Figure 3.5, and the left side is generation of Bloom filter "A+B", and right side is matching between target (T) and query (Q). In a general case a query should belong to target, but we will show another idea next section.

If the destination IDs are the same and the Bloom filter of query belongs to the Bloom filter of the publish, that is, the query match this published item, then the responding node sends a "callyou" message to the publishing node. There are maybe several published items matched the query, and all of their publishing nodes will receive the "callyou" messages. Next those users who receive the "callyou" messages could

decide whether they want to call back. If so, they will process a SIP call using SIP UA. One of them will be answered by caller and others will get a busy message.

Another problem is using Bloom filter may result in false positive. Our solution is processing double check in the callee while receiving the “callyou” message. It just costs a little computation in callee to completely break the possibility of error.

3.4 Numerical Attributes

We have mentioned that MFPGC system provides not only string type attributes but also numerical and hybrid attributes. Many kinds of user information such as age, income, location, etc…, contain numerical part and the query may be a range of number such as “Age from 5 to 20”. It is not efficient if we query age 5, age 6, to age 20 using simple publish/query mechanism, so a qualified method for numerical attributes is needed in current communication system. This problem is known as range query that we represented in Chapter 2, and we will propose an innovative algorithm for range query later.

The method we used in MFPGC system is dividing. A pre-defined numerical attribute will be divided into several levels. Each level is treat as an individual attribute, for example, the “AGE” attribute could be partition by five years. Therefore 0-5 is the first level, 5-10 is the second, etc…, and “AGE 0-5” is a special attribute implies the user’s age is in this range.

However dividing will cause false positive just like using Bloom filter. For example, we query a range “AGE 3-13”, but there are not defined this range, so we could only transform the query into three ranges “AGE 0-5”, “AGE 6-10”, and “AGE 11-15”. Obviously some numbers in the three ranges do not match our query, such as “AGE 15”

or “AGE 2”. We adopted the same mechanism as handling the false positive in Bloom filter to solve this problem. In other words we check whether the range in a query covers the number of callee when the “callyou” messages are sent to callee.

We store range attributes in a Bloom filter by using the same method of string attributes. However, a user usually tends to query a larger range of a numerical attribute. The large range is transformed to many levels and then inserts mass bits to Bloom filter. The false positive will become serious because almost every bit in Bloom filter is set to 1. We adopted another algorithm in [] that could avoid setting mass bits by hashing the range to fewer bits, and therefore perfectly fit in our system for range query.

An important factor of dividing is the size of partition. The smaller the partition is the more levels are contained in a query. That means the more bits will be inserted into the Bloom filter. Oppositely, the bigger the partition is, the fewer levels are contained in a query, but the false positive possibility in a range will increase. 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 range errors. We used the optimized solution proposed in [] to define the partition size of our system attributes, and it could approximate the lowest error in MFPGC system.

3.5 Necessary attributes

In Chapter 1 we motioned that current communication systems lack a mechanism to filter unwanted call. Necessary attributes is the functionality we designed for matching the caller and callee’s intensions. MFPGC system provides two kinds of necessary attributes in sender side and receiver side respectively.

3.5.1 Receiver necessary set attributes

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 receiver set necessary attributes and carried in publish to the responding nodes. We modify the publish message to include two Bloom filters, one is the original Bloom filter which all receiver-specified attributes set(RAS) are inserted into, and the other is an additional Bloom filter contains only receiver-specified necessary attributes set(RNAS). The responding nodes of each attribute will maintain these two Bloom filter after publishing.

When a node receiving a query, besides checking whether the attributes in each user record contain all attributes in the query, the node also checks whether the attributes in the query contain the RNAS of the user record. If both of the two conditions match, the record is treat as matching the query; if only the former condition matches, it means the called wants to call the callee but the callee would not like this request. The mechanism costs extra overheads to maintain and compare the two Bloom filter on each responding node, but it could perfectly filter unwanted call before the call requests are sent to callee. In Figure 3.6 we represent an example that publishes Peter's attributes including RNAS and RAS to the three responding nodes.

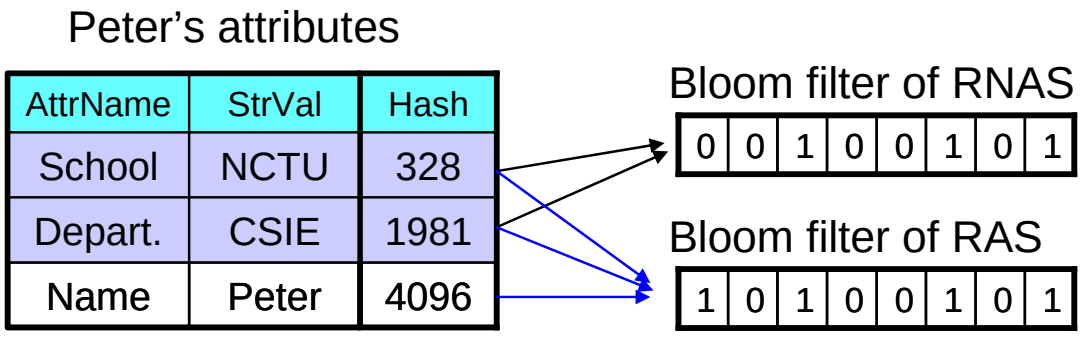


Figure 3.6.a Peter's user attributes. The first and second item is RNAS

AttrName	Nec	StrVal	H_val
School	Y	NCTU	328
Depart.	Y	CSIE	1981
Name	N	Peter	4096

Bf(RNAS)

0	0	1	0	0	0	0	1

Bf(RAS)

1	0	1	0	0	1	0	1

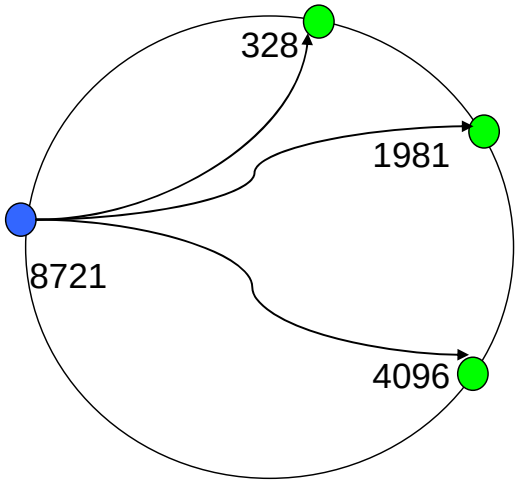


Figure 3.6.b The publishing process of Figure 3.6.a.

3.5.2 Sender necessary set attributes

In sender sides, query is based on sender-specified necessary attributes set (SNAS), the common part of queries in all system. But there is an important reason why we must have non-necessary attributes. Because of receiver necessary set attributes, the sender must query all attributes in RNAS in order to call that user. Setting RNAS protects users from unwanted query but make the query more difficult. If the number of attributes decreases, the query will not meet the RNAS of most users. Oppositely, if the number of attributes increases, the matched users become fewer. Both increasing and decreasing will resist the query process.

Using non-necessary attributes could solve this problem. We use necessary attributes as the basis of a query and add some related non-necessary attributes in the query. Although the RNAS of a user record does not belong to the RNAS of query, as long as it belongs to the sender-specified attributes set (SAS) in a query, thus this user matches the query. We conduct the final comparison equation of MFPGC system.

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

The *SAS* and *RAS* are total set attributes of sender and receiver respectively, and the *SNAS*

and *RNAS* are necessary set attributes of sender and receiver respectively.

The responding node will check the comparison equation for each record that has the same destination ID with the query. We shows a query process after publish process in Figure 3.6 and each matching case in Figure 3.7. In Figure 3.7.a the attributes “Depart:CSIE“ and “Name:Peter are SNAS. MFPGC system will choose one responding node of SNAS as destination to query, and we choose node 4096 in this example. Although the SNAS match Peter’s RAS, but the RNAS does not match because “School:NCTU” is not included in query A. Moreover, Figure 3.7b shows a 4-attributes query. Although the attributes in Peter’s RNAS all belong to SAS; however the SNAS attribute “Name:John” does not contained in Peter’s attributes, so the query still won’t forward to Peter. Even through the attribute “Name:John” does not match “Name:Peter” in Figure 3.7.c, the two attribute are neither RNAS nor SNAS, so the query still match Peter’s attributes.

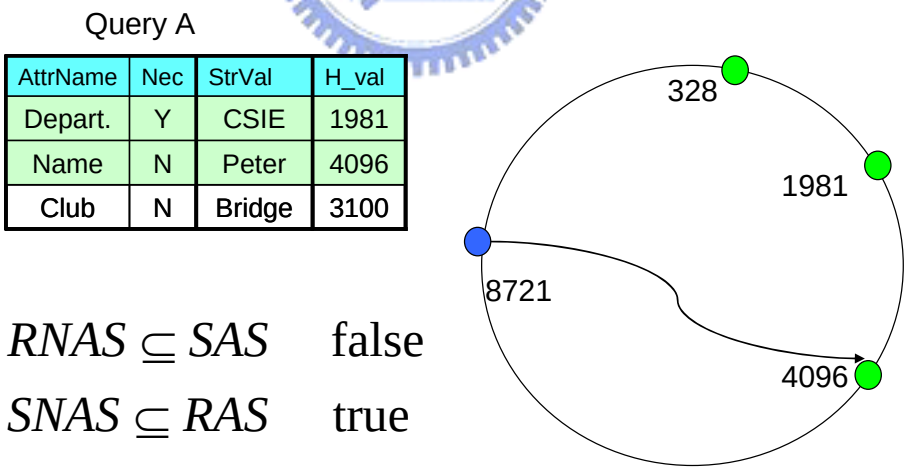


Figure 3.7.a SNAS are satisfied but RNAS are not

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

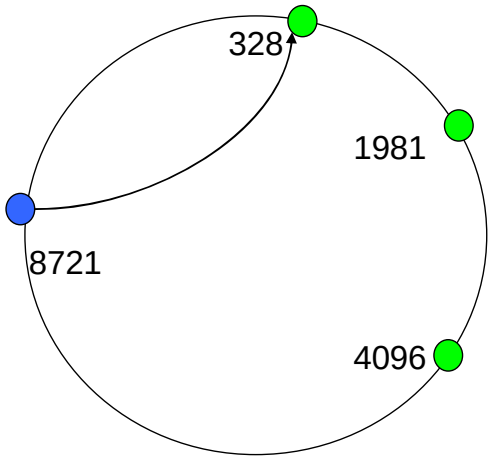


Figure 3.7.b RNAS are satisfied but SNAS are not.

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

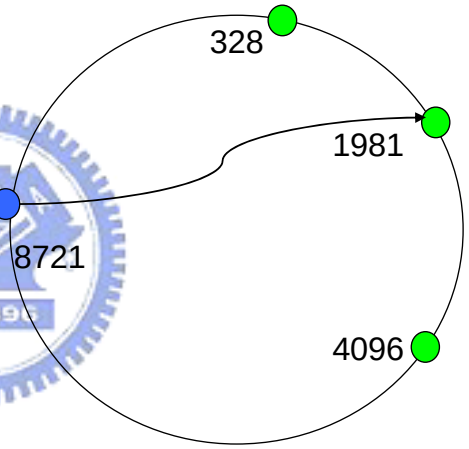


Figure 3.7.c The successfully matching example.

However, the caller could not specify range attributes as SNAS. The SNAS of caller must belong to the RAS of callee, but a range of query usually covers the number of single user. If a range attribute is set as a SNAS, no RAS could contain the SNAS and no user could match this query. To solve this problem, we divide the range into several levels and insert one level into the Bloom filter at once. Then we query several times and each of them including one level, and the joint set of those result is the result of setting range attributes as RNAS. But there is a big overhead if multiple range attributes are contained in a query, so we leave the improvement method in future works.

3.6 Call Handling for Off-line Users

Recall the call flow described in Section 3.1, a call contains 5 states to process, which are publish, query, callyou, callback, and answer respectively. MFPGC system still provides off-line user handling mechanism to handle caller or callee off-line in each calling state, that is, if a caller or callee disappears during one state, the call will preserve until next time the caller or callee joins.

There are many identifier used in MFPGC system become Chord, SIP, and IP are all the under layer protocols of MFPGC system and the five steps are implemented in different layer. Publish and query is the functionality provided by Chord, so the destinations are the responding nodes of the attributes and use Chord ID as identifier. The callback and answer is simply a SIP call, so the destinations are SIP URI. The responding nodes in Chord always exist by consistent hashing, so we won't discuss the situation of responding nodes off-line. And further MFPGC system is on the top of SIP layer, so whether the SIP URI is valid is also not our main point. We focus on two situations in MFPGC system, one is the users publish their information after the caller has queried, and the other is the callee becomes off-line after the responding node has matched and sent callyou message. MFPGC system provides off-line handling mechanism to make the call process smoothly.

3.6.1 Delayed query

The delay query mechanism simply means publish after query. The delay query flow is displayed in Figure 3.9. In order to achieve delay query we must save to query in the first forwarded node, and do the comparison process while receiving each publish message.

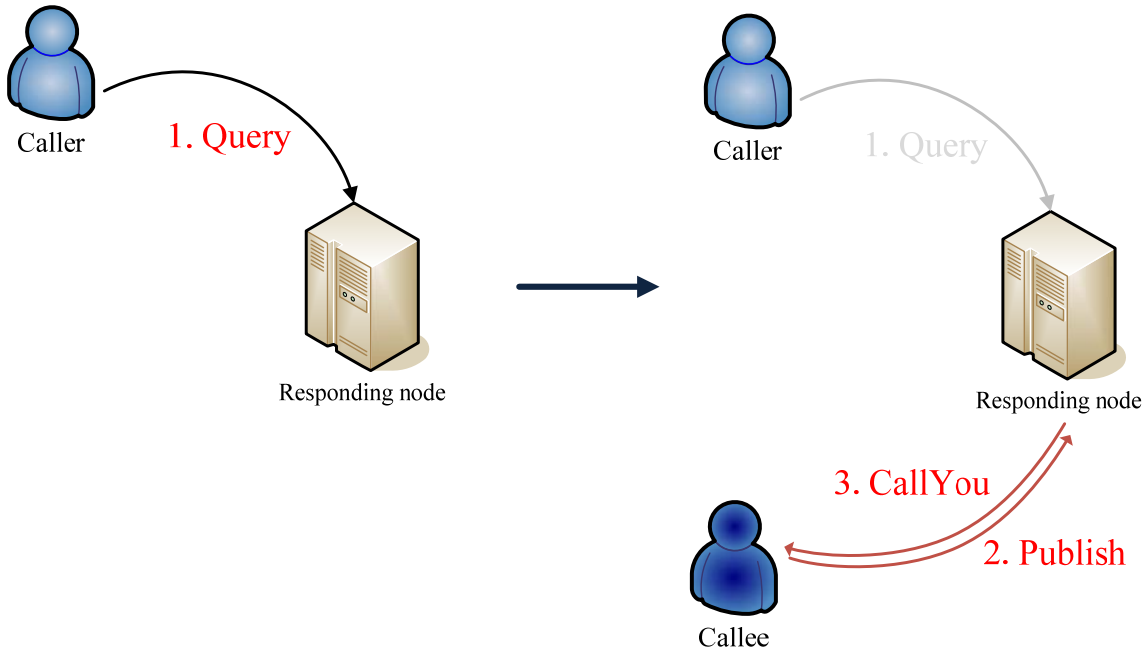


Figure 3.8 The delayed query flow

In order to identify a user, we add a profile ID to each publish message. A profile ID is an integer number generated by hashing all attributes of that user. We assume each user would fill enough specific attributes and could be identified by those attributes, so a profile ID could roughly identify a user in MFPGC system.

After a node has received a query message and compared its own record, it saves the query and a matching list contains the profile ID that has matched this query for a while. If a publish message comes during this time, the node compares the target ID and Bloom filters for the new user. If those conditions are all matched, the node further checks whether the profile ID has ever matched this query. A user may re-publish his information due to restarting MFPGC program or using MFPGC in another computer. By saving and comparing the profile ID we could recognize those situations and the user won't receive the duplicate callyou messages.

3.6.2 Delayed CallYou

The delay callyou is the callee becomes off-line before the “CallYou” message has been sent to it. As the callee returns on-line, the call will be resumed immediately. We mark the profile on-line or off-line in the responding node and add a “CallYouReply” message to acknowledge the “CallYou” message. If a node does not receive the reply after sending a callyou message, the responding node would mark the profile off-line. When the profile is off-line, every matched query is recorded in the responding node. Next time the user of this profile registers the profile will become on-line and the callyou messages will be forward to the user according to the matched queries.

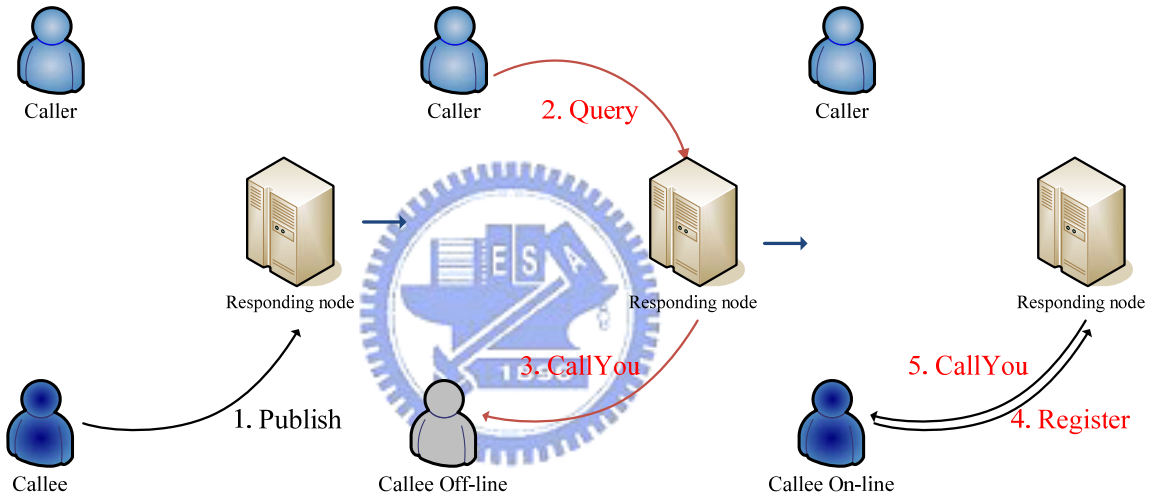


Figure 3.9 The delayed CallYou flow

3.6.3 Delayed CallBack

As we mentioned in Section 3.1, the callback and answer are implemented by SIP. The callback and answer process can not be handled by MFPGC, but we add a machine ID to the query message and keep each query for a TTL in local machine. When a caller is off-line, a callee that matches the query of the caller could not send the callback request to the caller. In this case, the callee can inform the responding node that the callyou messages received by the callee are cancelled. Next time when the caller becomes on-line, the valid queries are sent to the responding nodes with machine ID and the cancelled

callyou messages will be forwarded to the callee again. The delayed callback process could repeat until the caller and caller are on-line at the same time.

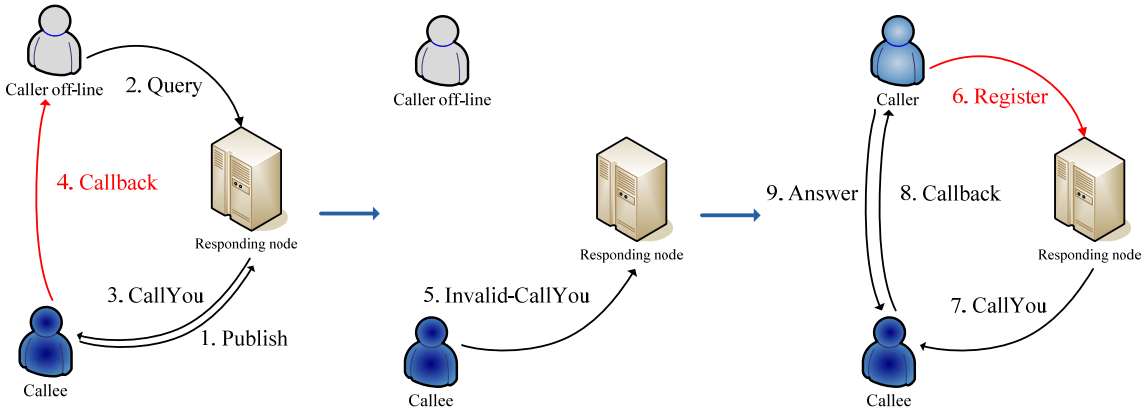


Figure 3.10 The delayed CallBack flow

Chapter 4 System Implementation

In this chapter we present our system implementation in detail. Our system is written in C++ and based on CCLSIP UA, a sip communication system. Figure 4.1 shows that MFPGC takes advantage of SIP, Chord DHT and IP to operate.

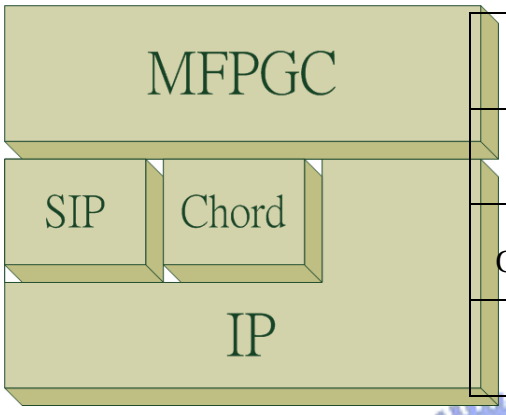


Figure 4.1 Layers of MFPGC system

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

Table 4.1 The MFPGC messages

4.1 System components

Our system has three types of node with different functionality. Figure 4.2 depicts the communication between those components and table 4.2 shows those difference and we will describe all in next section.

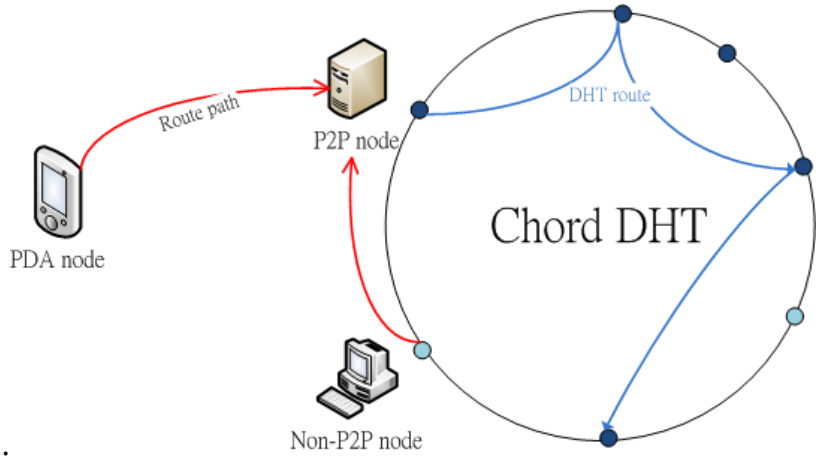


Figure 4.2 The three types of components in MFPGC system

components	DHT Routing	Store others information	Publish & Query	Location information
P2P node	Y	Y	Y	N
Non-P2P node	N	N	Y	N
PDA node	N	N	Y	Y

Table 4.2 The difference between the three components

4.1.1 P2P node

P2P nodes are the most important part of our system and implemented all of the peer-to-peer functionality such as routing, client, server and communication functionality like publishing, querying, communicating.

Due to the properties of Chord, a P2P node should maintain a finger table and a backup successor list for routing purpose. P2P nodes use a thread to periodically send a keep-alive message to each node in the finger table. If any node ungracefully left, other nodes which have saved it as finger will rebuild their finger tables.

Another thread listens the network socket and handles MFPGC messages we listed in Figure 4.1. And we deploy a database to maintain user attributes and the publish information of other users in two tables respectively. 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 node without peer-to-peer functionality, and provide a lightweight scheme for weaker computing, storage, or network users. A

non-P2P node would use a P2P node as the gateway to send MFPGC messages. The gateway P2P node could be a famous node or gotten from famous nodes. Because non-P2P nodes have no node ID, P2P nodes will not add non-P2P nodes to their finger table, and thus no DHT message will be forwarded to non-P2P nodes. The only messages sent by non-P2P nodes are the publish and query which contain the IP address and port. By using IP information other nodes could directly connect to non-P2P nodes without node ID. The only and most important functionality of non-P2P nodes is query and publish. Communication is another foundation functionality provided by under-layered SIP UA through IP network.

4.1.3 PDA node

PDA nodes are non-peer-to-peer nodes but executed in PDA side with GPS functionality. The MFPGC system in PDA side is based on mini UA, which is a mobile version of CCLUA implemented in Windows Mobile platform and provides SIP-based communication functionality through wireless network. The location attribute could be directly obtained using GPS in PDA node. The work flow and other usage of PDA nodes are the same as non-P2P nodes.

4.2 System defined attributes and user defined attributes

For convenience in implementation, we define some attributes in advance for users, and reserve extensions by user defined attributes. The system defined attributes in MFPGC system includes name, nick name, age, income, location, university, professional specialty, and hobby. The name, nick name, professional specialty, and hobby are string type attributes and the age and income are numerical type attributes. The location attribute includes two numerical attributes longitude and latitude respectively. The

university attribute is a hybrid attribute contains university name and the duration in that school.

4.3 Bloom filter implementation

As we mentioned in Chapter 3, MFPGC system uses Bloom filter to store multiple attributes. A Bloom filters is represented by a 512 bits array, and MD5 hash function is used to hash an attribute to an index. In order to generate *k* indices, we used the same hash function but different input text to hash the concatenation of the original text and "0" to generate a new index. Repeat the concatenating until we get *k* indices. For example, consider a system defined attribute "Club:Bridge" and *k* is 4. We hash "Club:Bridge", "Club:Bridge0", "Club:Bridge00" ,"Club:Bridge000" to four indices and set those position of the bit array to 1.

4.4 Message encryption

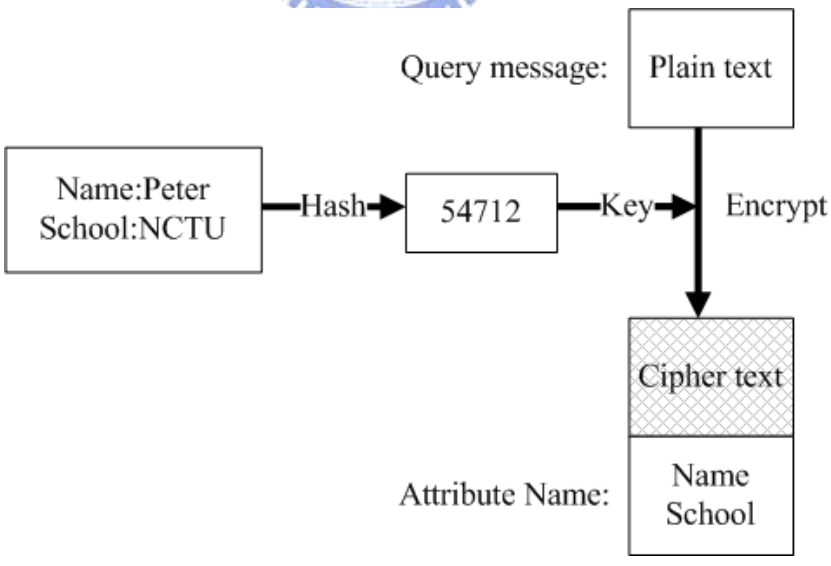


Figure 4.3 The encrypt process in query

In MFPGC system a caller will put a brief message in a query in order to invite the callee to callback. The Callee could read this message and decide whether he would call

back. The query message will be first sent to the responding node in DHT, and then forwarded to the callees. So if some private information is included in the message, the responding node could collect that information. Thus we use Advanced Encryption Standard (AES) algorithm to encrypt it. The key used in AES is the hash value of the joint of each necessary attribute name and value, and we also record the attribute names in the query. Therefore a callee receiving the query could use the attribute names to find the key of AES because each necessary attribute in query must belong to the attributes of the callee by the matching definition. Furthermore, the responding node is unable to know the attribute value so the privacy is perfectly kept in the query process.

Chapter 5 Performance Evaluation

In this chapter we list several DHT-based systems with multiple attributes query or range query functionality and compare their performance with MFPGC system.

5.1 Metrics and Comparison

At first we introduce some metrics to evaluate our system.

Hops: The most general metric to evaluate the routing overhead in overlay network, it implies the distance from source to destination. The fewer hops in a route result in lower overhead and less response time. If there are multiple destinations, the maximum hop is adopted in the process. Because a message transmitted between two nodes means the increasing of hops, the transmitted messages number is direct proportion of hops.

Number of nodes for a query: Number of destinations which do matching in a query. In multiple attribute query system, a query is usually forwarded to several destinations for searching because multiple attributes are stored in different nodes.

Number of nodes for a publish: Number of destinations which record the publish information. Different algorithm to handle multiple attributes query will result in difference of number of nodes for publish and query.

Storage in a node for a publish: A user publishes his attributes and one of the responding nodes should cost the storage to maintain the published information. The storage overhead counts on the information in a publish message.

Hops with respect to selectivity: The selectivity is a variable in range query and implies the percent of the numerical 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 systems we mentioned in Chapter 2 with MFPGC system in table 5.1 and 5.2. Table 5.1 show the comparison in multiple attributes aspect, and the variable *n* is the number of nodes in the system, n_{attr} is the number of attributes in a query or publish, *M* is the bit map size of Bloom filter. The *misc.* implies constant information such as IP address, node ID, and port. The P_r and P_l in MKey means the position of most right and left 1 in Bloom filter, and P_0 means the probability that a bit set to 0. $1-P_0(n_{attr})$ means the probability of a bit set to 1 after inserting N_{attr} attributes.

Table 5.1 The comparison between the five systems

class	**Multiple attributes string 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}	$M + misc.$
MKey	$P_r - P_l$	$(P_r - P_l)\times\log(n)$	$\left\lceil \frac{m(1-P_0(n_{attr}))}{2} \right\rceil$	$misc.$
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 5.1 display the performance of range query. We only compare MFPGC and MAAN because MKey does not support range query. The performance of SCARP and MARK varies with different range, and hardly evaluate in direct way. The variable *s* means the selectivity of a range attribute, and the $s_{\min}$ means the minimum selectivity range of the ranges in a query. The variables $R_{\max}$ and $R_{\min}$ mean the lower bound and upper bound of a numerical attributes, and *level* means the range to divide in MFPGC system. The last column shows the computation when the responding node compare a query with its maintained data, the n_t means the number of records which have the

same target ID as the query.

Table 5.2 The comparison between the five systems

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	$\frac{s\times(R_{max}-R_{min})}{level}\times\log(n)$	$\frac{s\times(R_{max}-R_{min})}{level}\times\log(n)$	$\frac{s\times(R_{max}-R_{min})}{level}$	$n_t \times M$
MKey				n_t
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 specific ID to specify the callee. In this thesis we design an innovative communication system using a set of 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 query user attributes. By using Bloom filter to represent user attributes and encrypting the 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 complete off-line user handling mechanism to focus on off-line users during each query, callyou, and callback. The call is kept even if the caller or callee is off-line and processes when they become on-line. The functionality will improve the flexibility of search and communication.

For the responding nodes of hot keywords, the loading in networking, storage, and computation all dramatically increase as the users increase. An efficient load balancing mechanism for our system is critical to promoting MFPGC system. The mechanism should consider hot keyword attributes and improve the level dividing method for range query. Another problem is a sender-specified necessary range attribute was not supported by our system and it 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, June 16–20, 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, June 1718, 2004,Paris, France,2004.