DESIGN AND SIMULATION

OF

DISTRIBUTED HASH TABLE IN CELLULAR MOBILE NETWORK

A dissertation submitted in partial fulfillment of the requirement for the Masters Degree in Computer Science and Information Technology

Submitted to

Central Department of Computer Science and Information Technology

University Campus

Tribhuvan University

Kirtipur, Kathmandu

Nepal

Submitted by

Lalita Sthapit

July, 2008

Central Department of Computer Science and Information Technology
University Campus
Tribhuvan University
Kirtipur, Kathmandu, Nepal

LETTER OF RECOMMENDATION

This is to certify that Ms. Lalita Sthapit has completed this dissertation entitled "**DESIGN AND SIMULATION OF DISTRIBUTED HASH TABLE IN CELLULAR MOBILE NETWORK**" under my guidance. This is her independent work for the fulfillment of the Masters Degree in Computer Science and Information Technology. I recommend this dissertation for final evaluation.

_____

Prof. Dr. Srinath Srinivasa
International Institute of Information Technology
Banglore, India

Tribhuvan University

Central Department of Computer Science and Information Technology

University Campus

Kirtipur, Nepal

LETTER OF APPROVAL

This is to certify that this dissertation entitled "**Design and Simulation of Distributed Hash Table in Cellular Mobile Network**", submitted by Ms. Lalita Sthapit, has been accepted for partial fulfillment of the requirement for Masters Degree in Computer Science and Information Technology.

Evaluation Committee:

Head, Central Department of Computer Science and Information Technology,	Supervisor, Prof. Dr. Srinath Srinivasa International Institute of Information
Internal Examiner	External Examiner
Date:	

ACKNOWLEDGEMENT

It is my pleasure to acknowledge the contributions of all the people who helped and guided me throughout the conduction of this work.

I would like to express my gratitude towards **Prof. Dr. Srinath Srinivasa** (IIIT-b) for his supervision and guidance despite his tight work schedules. I cannot remain without thanking **Mr. Sanket Patil** (IIIT-b) for his assistance and suggestions throughout the duration of this work.

I am thankful to **Prof. Dr. Devi Dutta Paudyal** (Former Head, Central Department of Computer Science and Information Technology) for his inspiration and encouragement during two years study of Masters Degree. Thanks to all the **honorable teachers** of CDCSIT who helped enhance our knowledge by sharing their knowledge and experiences with us.

I am indebted to my **parents and family members**, specially my aunt **Dr.Sabitri Sthapit**, for their continual inspiration and support to complete this dissertation. Without their encouragement, this work would not have become a reality.

Special thanks to my friends **Rashmi, Achyut** and **Ritesh** for their helping hands and fruitful discussions. I would also like to thank all the **friends** and my **seniors** at CDCSIT for their encouragement.

Lalita Sthapit

ABSTRACT

DHT is a decentralized system which acts like a hash table. It locates to a value in the decentralized system when provided with a key. DHT was first used by peer to peer system for the purpose of searching data within a group of nodes in a system.

In this work, the DHT concepts have been deployed in the base stations of cellular mobile network. The nodes in this DHT are, hence, immobile unlike other DHTs. The DHT has employed a data centric approach for searching of records. Thus the search is based on the location of the data rather than the location of nodes containing data. It is a system that binds the resources at different locations in a cellular mobile network by providing a method to approach those resources. It maps record keys to corresponding records using distributed index. This index is used to register as well as retrieve new records in the system.

A model of the system has been developed and simulation has been done as how the concept of DHT could be carried out over base stations. Simulation is a useful tool to study the dynamic nature system. A cellular network which is a discrete system, methods of discrete-event simulation is suitable to model and simulate such system. Hence an event based discrete event simulation has been used for the purpose.

LIST OF FIGURE

TABLE OF CONTENT

CHAPTER 1
INTRODUCTION

1.1 Background

Portable or mobile devices are extensively used today. They are growing more powerful in terms of computational and storage capabilities. These devices supports and provides variety of facilities like – browsing internet; managing calendar, date and time; storing of multimedia files like songs, videos, games, pictures etc. As a consequence, it is desirable to have applications and services that support not only communication but also data sharing among mobile users. Group of users may need to communicate for different purposes. For example – a small group of employee, at a large organization, may form a group to share official documents while they are on move. A group of students may form a collaborative group to share lessons, notes, and piece of music or pictures or maintain a global address book of all their friends. Mobile teamwork or collaboration has become an emerging requirement in the daily life nowadays. Many collaborative tools and system have been developed, and many are being proposed.

In distributed applications / systems, a node that wants data from another node needs to know the address of the node on which the data resides. If it does not know, it needs to find out. The requesting node needs to form a point-to-point connection with the destination node. Only then, communication and data sharing is possible between them. This approach of communication in which there is a point to point connection between two communicating nodes, is called host centric [10] approach. In this approach, the host is in focus rather than the interested data. This approach is suitable for networks where the topology is explicitly managed and the data are routed based on static routing tables [15]. In host centric approach, the network layer does routing of data with the help of host IP addresses. Routing is based completely on the host addresses, hence the term – host centric.

However, it is observable that the requests are interested in the required data which possesses the required data, and not the location of the node. So, applications should be able to obtain the required data regardless of where the data reside. Hence, it would be preferable to have an interface through which applications can issue queries and get to the source of the required data no matter what the address of source of the data. This approach is called data centric [13], since it focuses on the required data only, regardless of the hosting-node.

In a cellular mobile network, the two types of topologies can be considered. One is the network of the mobile devices, and another is the network of base stations. The topology of the base station is constant and does not change frequently. On the other hand, the topology of mobile devices changes constantly due to the continual movement of the mobile devices. A host centric approach to distributed applications over a cellular mobile network becomes very tedious, time consuming and complicated. This is because the hosts of the data/ records are the mobile nodes and it becomes highly inefficient to maintain dynamic host centric routing tables that need to be updated with every change in the topology of mobile device. No guarantee can be given about the validity of these routing tables. Moreover, it will be time consuming and expensive – in the sense that - point – to – point connection is established to search and find the required data until it is found in one particular hosting node. Therefore, a host centric networking approach is avoidable over this network.

This work is based on the research proposal [1]. Its objective is to build a *data centric middleware abstraction* on top of an existing collaborative network of mobile devices; the abstraction being the network of mobile nodes acting like a database. The dissertation has tried to look into one of the problems mentioned in the above research proposal. The details of the above research proposal have been discussed in the following section 1.1.3. The whole work has adopted the data centric networking approach, though the underlying routing model may use the concept of host centric networking. Applications interact with this data centric abstract layer and are not concerned about the underlying routing schemes.

According to the above mentioned research proposal, applications can query the network like querying a database. The network should take care of routing the query to the source of the data, and the data back to the source of the query, based just on the data and the query. Rather than having a host centric network and deploying a data centric overlay on top of it, data centric networking approach handles both network and database services at the same layer. This layer sits between the data link layer and the application layer. This layer would be called as a data centric abstraction middleware [1].

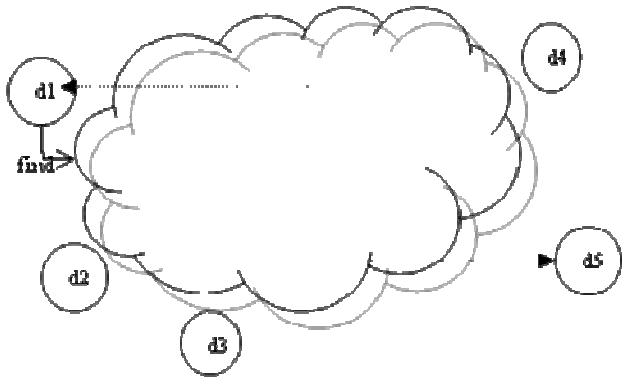

Figure 1.1: d1 indirectly communicates with d5 through an abstract layer

However, there are a number of challenges that had been identified by the above proposal writers while building the middleware [1]:

- Uniform Schemata
- Distributed Indexes
- Query Processing
- Consistency

Before discussing these challenges, let's consider some application examples of the abstract middleware.

1.1.1 Some Application Examples

1.1.1.1 An address book application

Let's consider a scenario, which comprises of a group of mobile users. One of the users wants to contact one of the other users but does not have the contact number of the later one. In general, this person would try to find out his number by contacting one or more of the other users by calling them or by sending those messages. Point-to-point connections are established between the first user and the other users from whom he is trying to get the number. This process would continue until he finds required number or until he gives up.

Now let's suppose a group of such users who share a common address book. A common address book is a collection of the records from all the users in that group. Again, one of the users is searching for another user's number. He would first search in his list of records. If not found, the abstract middleware application would search it in the common address book and finds it out from there if it exists there. The required number is returned to him as if it was present in his own record list.

1.1.1.2 Managing a calendar

Let's suppose there is a group of people who are working on certain task. For this purpose, they may have plannings and scheduled the subtasks according to the plan as to what part of the work has to be done on/within which date. This sort of schedule may need to be changed and updated several times. Hence a mobile calendar can be maintained and shared among the group. Any changes in the calendar/schedule would be known by the group members since the schedule is shared. Such an application for managing calendar can also have a capability to alert the members about the changes made.

1.1.1.3 Sharing multimedia files

As in the case of address book, a mobile user may have stored a list of multimedia files in her mobile device, which she may want to share with other interested friends. An

application can be built for managing and sharing these multimedia files. Such application may also have capabilities to search for a file among the group users and return it to the requesting user.

1.1.2 The Bigger Problem: Data Centric Middleware Abstraction

Now let's get back to the main problems that have been discussed in the proposal.

- Uniform Schemata

  Records are stored in each mobile device in the form of a table. If applications are made to communicate through queries, the query interface should be uniform across all the communicating applications. That is, these applications should be familiar with other applications database schemata and share the similar schemata. These schemas cannot be guaranteed to be same in all the communicating mobile devices. Therefore, one of the major challenges was found out to be the way to find a technique to provide uniform schemata of the data to be shared across the communicating applications.

- Distributed Indexes

  "According to the proposal, every mobile has one or more tables. Users create indexes on some of the fields that are regularly searched for. A large number of queries are based on the values of the indexed fields. Queries are based on the values of the indexed fields. Queries are data centric. Therefore, an application that queries the network need not be concerned about the location of the data. It should simply get the data as though it were locally available. The database indexes should enable this by acting as data centric routing tables." [1]

  The focus of this dissertation is to contribute on this particular problem.

- Query Processing

  The processing of query has been discussed to be another major problem in the above mentioned proposal. It specifies that the queries can be complex and its

answer may not be available in a single node. Queries may need to be broken down and distributed across the network or it may need to be replicated to find its complete answer. Above all, the query needs to be processed in the abstract layer/ network and not in a particular mobile device.

- Consistency

  Another major problem identified by the researcher is to maintain consistency across the multiple copies of the same data that exist in the network. It is undeniable that there may exist two replicas of same data. Also, given two different copies, then how can a correct one be found?

The above are the four major problems that were found challenging in building an abstract middleware. Among them, second problem – that is – to build a distributed index has been kept into focus on this dissertation.

1.2 The Problem Definition

1.2.1 The distributed index

Distributed index –as the name suggests- is a collection of indexes that are partitioned and distributed across various nodes in a network. The required data requested by a query is located using this distributed index. This index forms on of the major part of the abstract middleware. The major goal of this dissertation has been the designing this distributed index and simulate a case.

1.2.2 An approach in brief

Every mobile node has table of records, see figure 1.3. These mobile devices are under a range of certain base station. These mobiles may also have their own indexes in their tables. If a mobile wants to participate in the group of sharing data, she has to register. Every base station has a bucket, which is the collection of indexes from group of mobile nodes under its range, see figure 1.3. The set of these buckets from all the base station forms the distributed index.

Now, each bucket acts like a Hash Table, see figure 1.4. Hence, the name – distributed hash table. It consists of tuples, which map the data-keys into the node that has the records corresponding to the key. When a BS receives a key, it is hashed. This hash value helps to locate that particular node and the mobile in which the data resides. Once this node id is found, the required record can be returned to the requesting node. The detailed description is on chapter 3.

The above approach of creating a distributed hash table has been simulated using the concepts of discrete event simulation. The detailed description is on chapter 4.

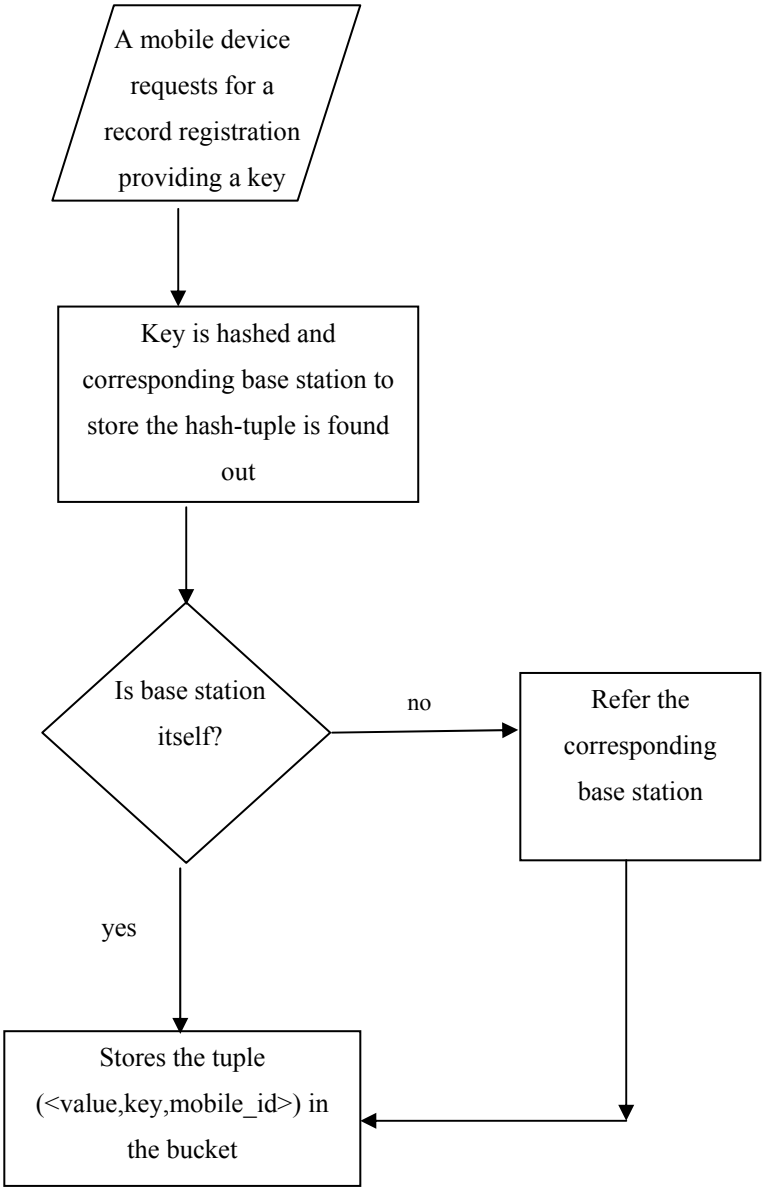

Figure 1.2: A simple flowchart of the registering approach

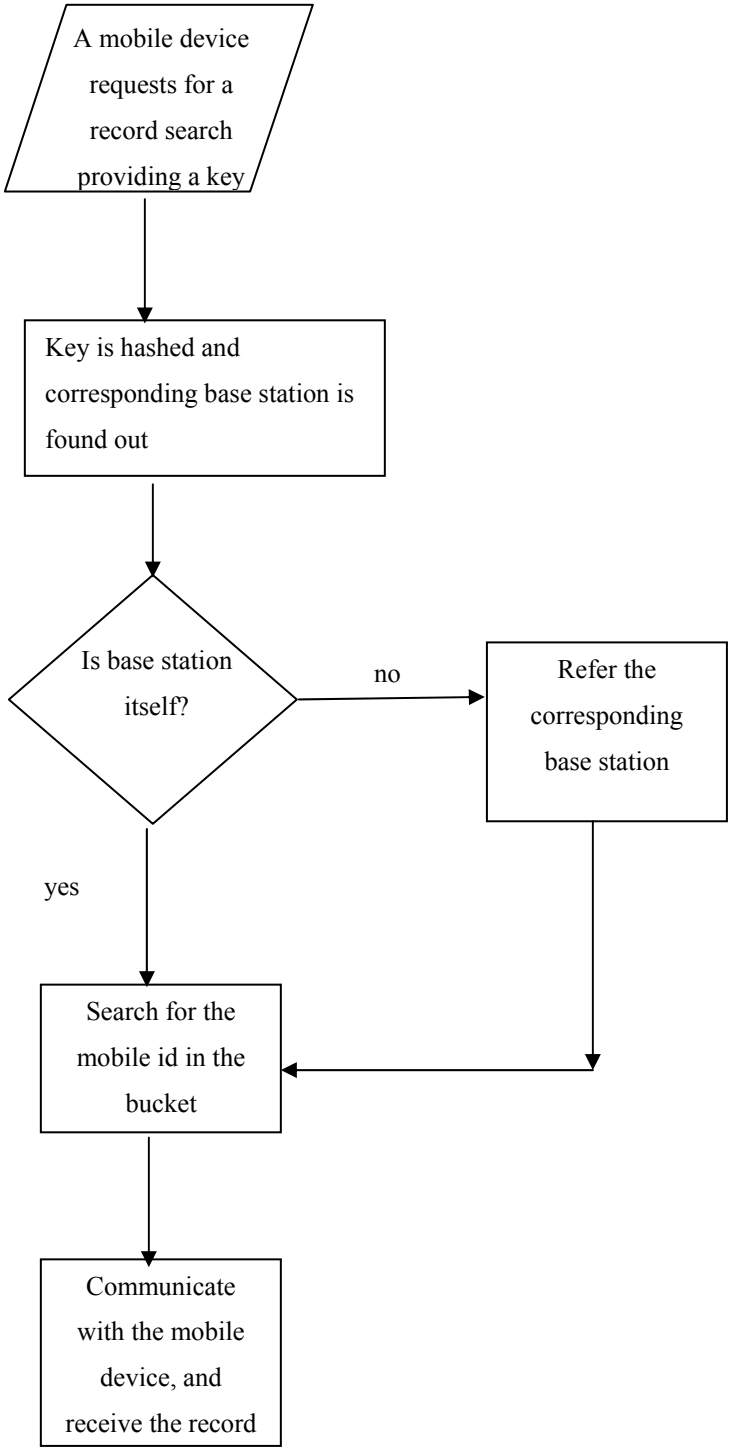

Figure 1.3: A simple flowchart of the retrieving approach

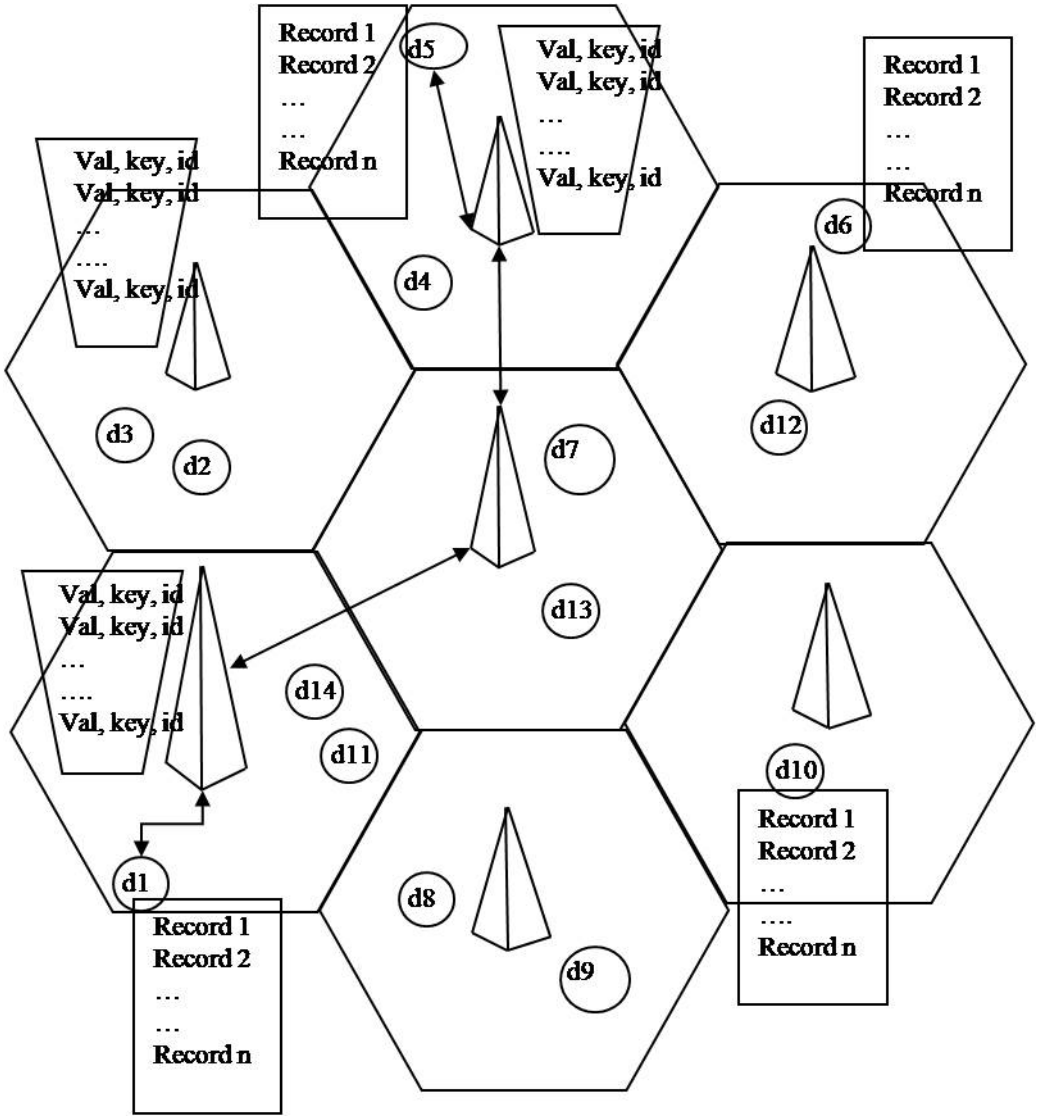

Figure 1.4: d1 searching for record. Base stations acting inside an abstraction. Record found in d5.

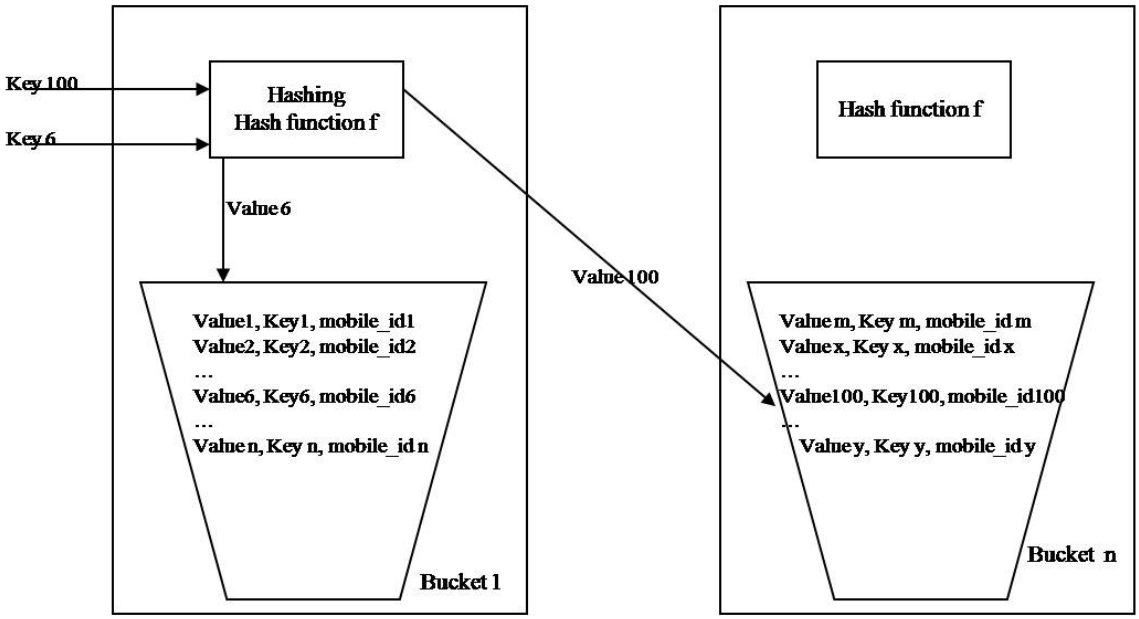

Figure 1.5: Hashing – Given a key, finding out the mobile_id where the record with the given key resides

CHAPTER 2
CELLULAR NETWORK

2.1 An Introduction

Cellular Network is an advanced communication technique. It is the foundation of mobile wireless communication like mobile telephones, personal communication system, and wireless internet and web applications. It supports users in locations that are not easily served by wired network. A cellular network is a radio network made up of a number of low power transmitters. The range of such low power transmitter is small; hence, the area is divided into number of hexagonal radio cells, each served by a base station consisting of a transmitter, receiver and control unit. Each of these cells is allocated by a band of frequencies. Adjacent cells are assigned different frequencies to avoid interference. Cellular networks offer a number of advantages such as, increased capacity, reduced power usage, better coverage.

2.1.1 Characteristics

2.1.1.1 Frequency Reuse

As has been stated earlier, each cell has a base transceiver, and each cell is allocated a band of 10 to 50 frequencies depending upon the traffic expected. The increased capacity in a cellular network is due to the use of same radio frequency band in a different area for a completely different transmission. This reuse of same frequency in other nearby cells allows the frequency to be used of multiple simultaneous conversations. The transmission power must be controlled carefully to allow communication within the cell using the allocated frequency while limiting the power at that frequency that escapes the cell into adjacent ones. It is essential to determine the number of cells that can intervene between two cells using the same frequency so that the two cells do not interfere with each other. The frequency reuse factor is the rate at which the same frequency can be used in the network. It is 1/K where K is the number of cells, which cannot use the same frequencies

for transmission. Common values for the frequency reuse factor are 1/3, 1/4, 1/7, 1/9 and 1/12.

2.1.1.2 Increasing Capacity

As more customers use the system, traffic may increase so that there are not enough frequencies assigned to a cell to handle its calls. This may be handled as follows:

- Adding new channels:

  When a system is set up in a region, not all of the channels are used, and growth and expansion cab be managed in an orderly fashion by adding new channels.

- Frequency borrowing:

  Frequencies may be taken from adjacent cells by congested cells. Frequencies can also be assigned to cells dynamically.

- Cell splitting:

  Cells in areas of high usage can be split into smaller cells. To use a smaller cell, the power level used must be reduced to keep the signal within the cell. Also, as the mobile units move, they pass from cell to cell, which require transferring of the call from one base transceiver to another. As the cells become smaller, this process becomes frequent. A radius reduction factor F reduces the coverage area and increases the required number of base stations by a factor of F^2.

- Cell sectoring

  A cell is divided into number of wedge shaped sectors (3-6 per cell), each with its own set of channels. Each sector is assigned a separate subset of the cell's channels, and directional antennas at the base station are used to focus on each sector.

2.2 Operation of Cellular System

A base station (BS) is placed approximately near the center of each cell. Each BS has several transmitters and receivers, which simultaneously handle full duplex

communication and also have towers, which support several transmitting, and receiving antennas. Each of these BSs has number of forward channels to transmit to its mobile users and an equal number of reverse channels to receive from mobile users. BSs are connected by a wire-line transmission link or by point-to-point microwave radio to a

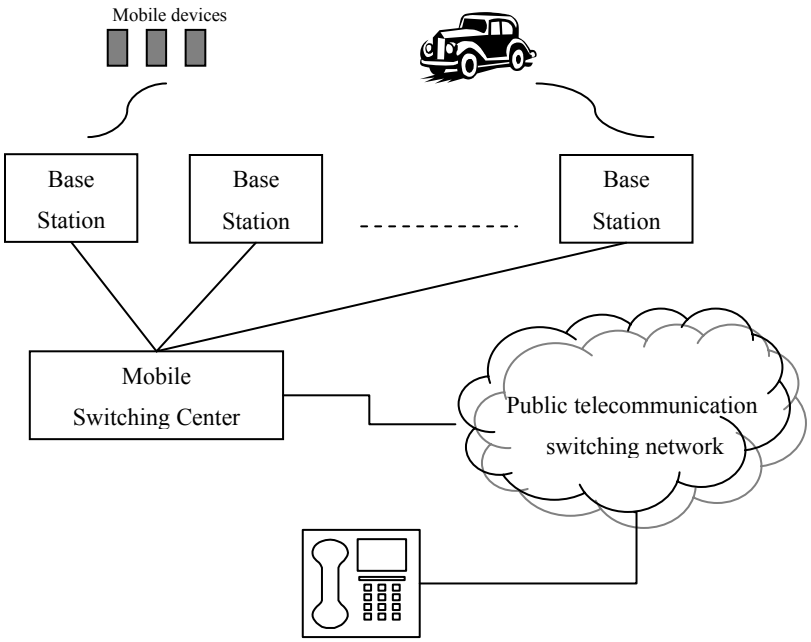

Figure 2.1: Overview of cellular system

telephone switch called mobile switching center (MSC). One MSC serves multiple base stations. It also handles connections between BS and public switched telephone network. The MSC assigns voice channels to each call, performs handoffs and also monitors calls for billing information. To use the cellular system, users only needs to place or answer a call, the remaining are automatic. Two types of channels are available for communication between a mobile device and BS. Control channels exchange information about setting up, establishing connection of mobile with a BS and maintaining call, while traffic channels transmits voice or data across connected mobile units.

Different cells/BS with different frequency bands frequently broadcasts on different setup channels. When a mobile device is turned on, it selects the appropriate BS by selecting the strongest setup channel in the system. Then a handshaking takes place between this mobile unit and the MSC that controls this BS. This handshake identifies the user and registers its location. This course of action is repeated until the mobile unit is on, to keep track of its location and movement. If the mobile unit enters in a new cell, then a new BS is selected.

To set up a connection with another mobile unit, a mobile unit sends a number of the receiving mobile unit on the preselected setup channel. The receiver at the mobile unit first checks if the setup channel is idle. If yes, it transmits the request. The BS sends the request to the MSC.

The MSC sends a paging message to BSs. Each BS transmits the paging signal on its own assigned setup channel.

The receiving mobile unit recognizes its number on the set up channel being monitored and responds to that BS. This BS, in turn, responds the MSC. The MSC establishes a circuit between a sending and receiving BSs by assigning an available traffic channel within each BS's cell and notify these BSs. The BSs, in turn, notifies their corresponding mobile units. The two mobile units communicate with the assigned channels. The mobile units can exchange voice and data signals through their respective BSs and MSC till the connection ends.

2.3 Handoff

The mobility of the mobile units is one of the important requirements in a wireless communication system. In cellular system, continuous coverage is achieved by executing **handoff**, when the mobile units cross cell boundaries. Handoff is a procedure of changing the assignment of a mobile unit from one base station to another as the mobile moves from one cell to another. It is an important issue because in a cellular network,

neighboring cells use disjoint subset of frequency bands and there should be a negotiation between the mobile unit and the serving base station.

As the call proceeds, the base station monitors the signal level. If the signal level falls below a specified threshold the MSC is notified and the mobile station is instructed to transmit on the setup channel. All base stations in the vicinity are instructed to monitor the strength of the signal level prescribed setup channel. The MSC uses this information to determine the best cell to which the call should be handed off. The current base station and mobile station are instructed to prepare for a handoff. The MSC then releases its connection to the first base station and establishes a connection to the new base station. The mobile station changes its channels to those selected in the new cell. The connection is interrupted for the brief period that is required to execute the handoff.

2.4 The term - Data Centric

2.4.1 The Host Centric Approach

In any communication model, distributed system or resource sharing system, the applications are designed in such a way that, users have immediate access to computing resource and those resources have access to application data. When an application instance communicates with another, it needs to know the address of the host (i.e. the node in which the later instance is running). In order to transfer resources, a point-to-point connection between the two communicating nodes should be established first. This approach is termed as host centric since the host of the resource is in focus rather than the data itself. Once the relationship between the two end points is established, resources are transferred between them.

In this approach, synchronization between servers, and between servers and clients, is complex. When a system is upgraded, there may be a need for modifying the interface between endpoints, upgrading and testing new codes and configurations. The absence of incompatibilities should also be ensured during such modifications.

Another complexity is regarding the dynamics in the network architecture. Nodes in the network may fail to stay in the network due to some reasons. Nodes may join and leave recurrently. The application architect cannot assume that the network and application architecture is static and unchanging. It may be difficult to expand the application to include a larger network with more endpoints.

2.4.2 The Data Centric Approach

The data-centric approach gives emphasis to the data and not the data owner. This is because, in the data sharing process, the data is more important than the host which hold that data. Hence, the application requesting the data must receive the data regardless of whether it knows or does not know the location of the data.

Various approaches have been adapted to achieve data centric property. Among them, one is the publish-subscribe [13] (or "pub-sub"). In this model, data sources, or producers, publish data to a common location on the network. This could be a memory-to-memory transfer, or it could be a database or other persistent storage. When published data arrives at the shared location, a message goes out to the subscribers. Processes that need that data can subscribe to it through a messaging service. Subscribers can then go to the shared location to obtain the data, and use it in their own processing. A publish-subscribe model for data distribution enables the implementation of such a data-centric architecture across a large-scale network.

Many of the approaches achieve the data centric property by deploying an overlay that abstract the underlying layer which may still be host centric.

The host centric routing is suitable for networks where the topology is managed explicitly and the data are routed based on static routing tables, since the routing is totally based on the host addresses. In cellular mobile system, the hosts of the data are the mobile nodes and it becomes highly inefficient to maintain dynamic host centric routing tables that need to be updated with every change in the topology of mobile device. The information

in the routing table may become invalid due to the continual change in the positions of mobile nodes. Point – to – point connection is established to search and find the required data until it is found in one particular hosting node. Therefore, data centric approach is more uncomplicated and comfortable in cellular mobile systems rather than host centric approach.

CHAPTER 3
DHT DESIGN

3.1. Distributed Hash Table (DHT)

3.1.1 Hashing

Hashing is a searching scheme to, directly, find out the location of a record no matter where the record is in the record storage space. The fundamental principle behind hashing is to determine a number from the given key information and use this number to access information related to the key. The transformation of the key into a corresponding location is done using a **hash function**. A hash function takes a key as an input and transforms it into a value, called **hash value**, which indicates the location of the required information.

A fundamental property of all hash functions is that if two hash values are different, then the two input keys are also different in some way. But this may not be the case always. Sometimes, different keys may be transformed to the same location. When this occurs, it is called **collision**. So, a good hash function is the one in which there is minimum possibility of collision. Hash table is a major application for hash functions. A **hash table** is a data structure that associates keys with values. A hash table is, generally, built up using an array where the data to be searched is stored. A given key is transformed into an array index (the associated value for the given key) using a hash function. The hash function is a way for assigning numbers to the input data such that the data can then be stored at the array index corresponding to the assigned number. A hash function for a hash table should be fast relative to the cost of retrieving a record in the table, besides minimizing collision such that the time required to retrieve a desired record is less.

3.1.2 A Distributed Hash Table (DHT)

DHT is a hash table, consisting of a list of (key, value) pairs. Using a hash function, values can be mapped to a given key. Besides this, storage, lookups and retrieval are distributed among multiple machines, called nodes, in DHT. Each node is analogous to an array slot in a hash table. DHTs are typically designed to scale to large numbers of nodes. These nodes can join and leave the network as and when they want.

DHT are decentralized – in the sense that the nodes, collectively, form the system without any central coordination. DHT are scalable. The system functions efficiently even with thousands of nodes. Further, the system should also be reliable even with nodes continuously joining, leaving, or failing. Hence, distributed hash tables (DHTs) are decentralized distributed systems

- in which a set of keys are distributed among multiple nodes participating in the system
- which can efficiently route messages to the unique owner of any given key and
- which can handle continual node arrivals and failures.

A key technique used to retain the above properties is that any one node needs to coordinate with only a few other nodes in the system, so that only a limited amount of work needs to be done for each change in membership. DHTs should also make provable guarantees about issues such as load balance, data integrity, and performance.

3.1.3 Structure of DHT

A DHT is built around a domain of keys (an example of keys may be set of 160-bit strings). The keys are distributed among the participating nodes according to a **key space partitioning scheme**. The **overlay network** connects the nodes, allowing them to find the owner of any given key in the domain of the key. Storage in and retrieval from the DHT may proceed as follows. Suppose the key space is the set of 160-bit strings. To store a file with given filename and data in the DHT, the SHA1 hash of filename is found, producing a 160-bit key k. Then, a message may be sent to any node participating in the

DHT to store the key k and its corresponding data. The message is forwarded from node to node through the overlay network until it reaches the single node responsible for key k as specified by the key space partitioning, where the pair (k, data) is stored. Any other client can then retrieve the contents of the file by again hashing filename to produce k and asking any DHT node to find the data associated with k by message passing. The message will again be routed through the overlay to the node responsible for k, which will reply with the stored data.

3.1.3.1 Key space Partitioning Scheme

Most DHTs use some variant of consistent hashing to map keys to nodes. This technique employs a function $\delta(k_1,k_2)$ which defines an abstract notion of the distance from key k_1 to key k_2. Each node is assigned a single key called its identifier (ID). A node with ID i owns all the keys for which i is the closest ID, measured according to δ. Consistent hashing has the essential property that removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs, and leaves all other nodes unaffected.

3.1.3.2 Overlay Network

Each node maintains a set of links to other nodes (its neighbors or routing table). Together these form the overlay network, and are picked in a structured way, called the network's topology. All DHT topologies share some variant of the most essential property: for any key k, the node either owns k or has a link to a node that is closer to k in terms of the key space distance defined above. To route a message to the owner of any key k, forward the message to the neighbor whose ID is closest to k. When there is no such neighbor, then we must have arrived at the closest node, which is the owner of k. It is better if the maximum number of hops in any route (route length) is low, so that requests complete quickly; and that the maximum number of neighbors of any node (maximum node degree) is low, so that maintenance overhead is not excessive. In the world of decentralization, distributed hash tables (DHTs) recently have had a

revolutionary effect. Knowledge of DHT algorithms is going to be a key ingredient in future developments of distributed applications.

3.2 Literature

There has been a lot of research work dealing with DHT. Following are some of the recent DHTs that have been proposed as a platform for building a variety of scalable and robust distributed applications.

File sharing is to grant access of files and information to others through computers and networks. Napster pioneered the concept of peer-to-peer sharing system [12]. It introduced a technique for peer-to-peer file sharing in decentralized systems. It was an online music file sharing service created by Shawn Fanning and operating between June 1999 [2] and July 2001 while he was attending Northeastern University in Boston. It allowed music fans to share MP3 format song files with each other. The system used a central database system to process and store musical resources. Napster was accused by the music industries of massive copyright violations. The original service was shut down by court order.

Gnutella [12], is a popular file sharing system feasibly established today as a third largest peer-to-peer file sharing system. The main feature of Gnutella is that, users put the files they want to share on their hard disks and make them available to others in the network. There is no central database that possesses all the resources available on the network. Instead, the machines on the network tell each other about available files using a distributed query approach. To participate in the Gnutella network, users run a piece of Gnutella software. Also, there are many different client applications available to access the Gnutella network. Gnutella network is an unscalable distributed system [3], and hence inspires the development of distributed hash tables, which are much more scalable but support only exact-match, rather than keyword, search.

Chord [4] is a distributed lookup protocol that efficiently locates the node that stores a particular data item. Chord uses a one-dimensional circular keyspace. The node responsible for the key is the node whose identifier most closely follows the key; that node is called the key's successor. Chord maintains two sets of neighbors. Each node has a successor list of nodes that immediately follow it in the keyspace. Routing correctness is achieved with these lists. Routing efficiency is achieved with the finger list of nodes spaced exponentially around the key space. Routing consists of forwarding to the node closest to the key; path lengths are hops.

In Pastry [5], nodes are responsible for keys that are the closest with the keyspace considered as a circle. The neighbors consist of a Leaf Set, which is the set of closest nodes. Correct routing can be achieved with this leaf set. Routing consists of forwarding the query to the neighboring node that has the longest shared prefix with the key (and, in the case of ties, to the node with identifier closest numerically to the key). Pastry has neighbors and routes within hops.

CAN (Content Addressable Network) [7] choose its keys from a d–dimensional toroidal space. Each node is associated with a hyper-cubal region of this key space, and its neighbors are the nodes that "own" the contiguous hypercubes. Routing consists of forwarding to a neighbor that is closer to the key. CAN has a different performance profile than the other algorithms; nodes have neighbors and path-lengths are hops. Note, however, that when, CAN has neighbors and path-lengths like the other algorithms.

Tapestry [11], to address the problems of the first generation of peer-to-peer applications, a second generation of p2p applications was developed. This includes Tapestry. It also implements a basic key-based routing mechanism. This allows for deterministic routing of messages and adaptation to node failures in the overlay network. Tapestry is an extensible infrastructure that provides decentralized object location and routing focusing on efficiency and minimizing message latency. This is achieved since Tapestry constructs locally optimal routing tables from initialization and maintains them in order to reduce routing stretch. Furthermore, Tapestry allows object distribution determination according

to the needs of a given application. Tapestry also allows applications to implement multicasting in the overlay network.

However, all the above approaches have been designed for the Internet, Ad Hoc networks, and peer-to-peer networks. These networks have either a static topologies (like in peer-to-peer network) or have totally dynamic network (ad hoc networks). DHTs like Chord, Pastry, and CAN were developed with the idea of providing data centric overlays. In this dissertation, a concept of DHT has been tried to be deployed over a cellular mobile network in which base stations are fixed whereas mobile nodes continually change their position.

Some additional literature on mobile collaboration goes as follows.
MoCA [16], mobile collaboration architecture is a middleware for developing and deploying context aware mobile collaborative applications for mobile users. It addresses the problem of capturing and processing users' queries. It comprises of client server model as in mobile databases with an addition of collaborative layer.

A MOTION [8] service architecture is architecture for mobile teamwork [9]. In this architecture, peers can interact with each other either by passing message or using pub-sub system. It also supports distributed searches.

3.3 DHT Design – Our Approach

A cellular network can be visualized as a distributed system of mobile peers with some special static nodes that are connected to a high-speed reliable network [6]. In this design of DHT, the base stations are considered to be the nodes, which acquire the distributed indices like structure. Each cell in a cellular network has its controlling base station and each base station controls a set of mobile devices under its range. Each of these mobile devices has a list of records of interest. The owner may create an index on some of the fields. Each node (i.e. base station) has a bucket containing a list of tuples of the form

<value, key, mobile_id>. Here, the *key* is the value of a field of a record to be searched. The *value* is the hash value of the key, the mobile_id being the identifier of the mobile which consists of the record with the given key. Using this tuple, the desired record could be reached. The collection of these buckets form the distributed index. As has been discussed earlier, in a cellular mobile network, there are two types of topologies. One is the network of the mobile devices, and another is the network of base stations. The topology of the base station is constant and seldom changes. While the topology of mobile devices changes constantly due to the continual movement of the mobile devices. Hence, a host centric approach to distributed applications over a cellular mobile network becomes very aggravating, time consuming and complicated. This is because the hosts of the data/ records are the mobile nodes and it becomes highly inefficient to maintain dynamic host centric routing tables that need to be updated with every change in the topology of mobile device. Guarantee cannot be given about the validity of these routing tables. Moreover, it will be time consuming and expensive – in the sense that - point – to – point connection becomes necessary to search and find the required data until it is found in one particular hosting node. Therefore, we consider a data centric networking approach over this network.

The DHT is comprised of various components. They have been discussed in the following sections.

3.3.1 Keyspace Partitioning Scheme

The key space consists of two sets of keys; one of them is the set of keys of nodes and the other is the set of keys of records. Each node is assigned a single unique key called its identifier (or node id). This key is hashed to produce a value. This value is the part of the key space and used to uniquely identify the node in the key space. In this DHT, SHA-1 [14] has been used to produce a 160-bit identifier space represented by a 40 digit hex key. Similarly, the values of the key field of all records are also hashed into a 160-bit value and identifies the record tuples, such node-hashes (node identifiers) and the record-key-

hashes (record identifiers), combinely, form the key space. See Figure: 3.1. These identifiers are evenly distributed in the overlay network with each node storing several different identifiers.

The key space partitioning scheme splits the ownership of this key space among the participating nodes in such a way that a node with an id x has all the keys for which x is the closest id. The node ids are considered to be points on a circle. The circular key space is split into contiguous segments whose end points are the node ids. If i1 and i2 are the two adjacent ids, then the node with ID i2 owns all the keys that fall between i1 and i2. See Figure 3.2.

Here, since the base stations are the nodes, it is assumed that the frequency of addition and removal of the nodes is null. So, the node addition, removal and their effects would not be dealt with.

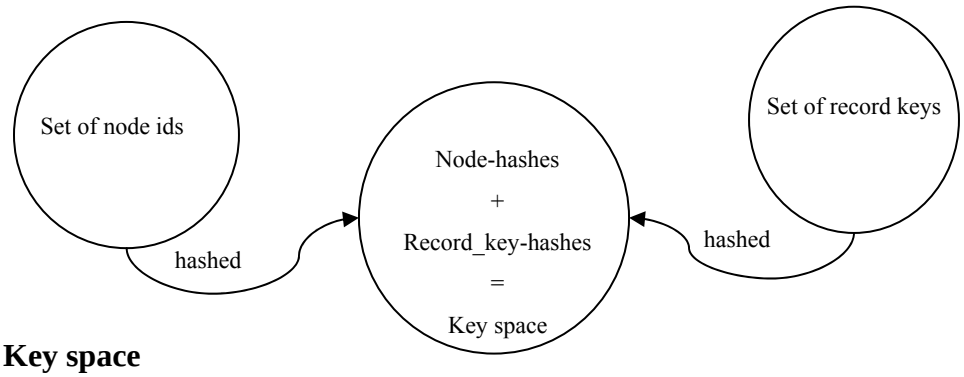

Figure 3.1: Key space

Once, the keys are partitioned, the overlay network connects the nodes. The owner of any given key in the key space can then be found out.

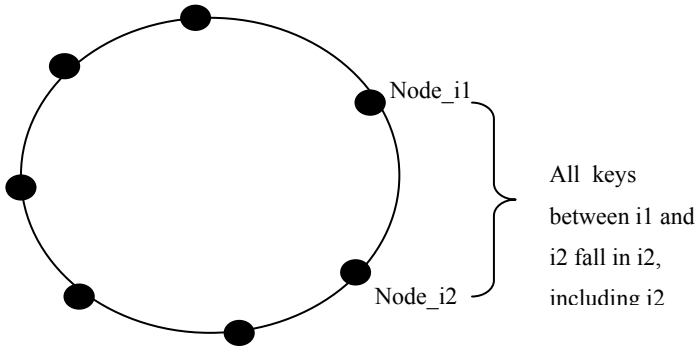

Figure 3.2: Key space Partitioning

3.3.2 Overlay Network

The overlay network allows mapping keys onto corresponding nodes. It has been assumed that the nodes (or base station) of the cellular network are arranged in a ring. That is, each base station is connected to the other two base stations, one being a previous node and the other being the following node.

The node identifier is arranged in certain order (eg. ascending order clockwise). If a node n is the current node and there is a destination node id, it is decided whether to go to the following node or the proceeding node to the reach the destination node.

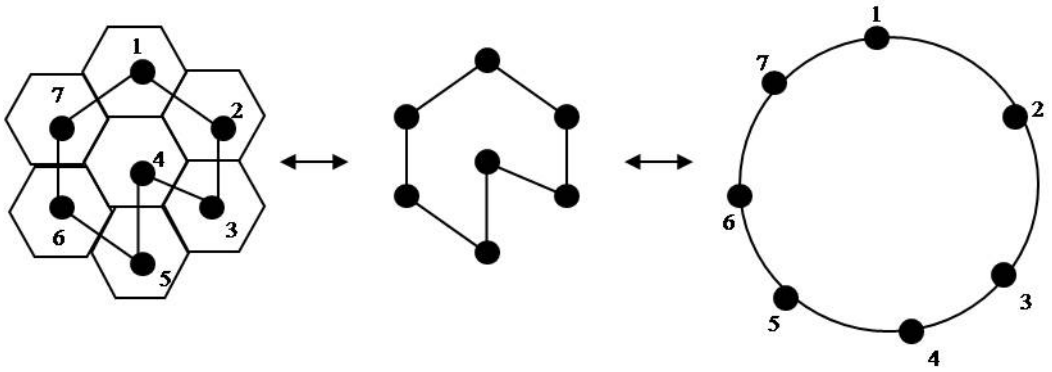

Figure 3.3: Arrangement of Nodes in a ring. Each node is connected to the two other nodes.

For example, if the destination node id 5, and the current node is 2 (which is less than 5), then the proceeding direction is forward clockwise rather than backward anticlockwise. It is possible to reach the destination node from backwards too, since the nodes are arranged in a ring but the distance is long. The direction of the move should be carefully decided accordingly such that the cost is minimum and less time is spent to reach the destination.

In this scheme, the maximum number of hops, in the worst case, to reach the destination node is n/2. Here, n is the number of nodes. The time complexity is linear i.e. O(n). This seems to be a somewhat inefficient. However, the efficiency can be improved by modifying the network topology itself, by increasing the number of neighbors, i.e.

increasing the degree of nodes. Again, if the degree is larger, the maintenance overhead may be excessive.

3.3.3 Bucket Management

All cells in the cellular network have a base station each. Each base station communicates with and controls all the mobile nodes under its range. Each base station consists of a bucket in it, see figure 1.3 and 1.4. The bucket consists of a list of tuples in the form <value, key, mobile_id>. Here, key is the key of the record to be searched. The value is the corresponding hash value of the key, which would map us to the mobile_id where the actual record of the key resides.

3.3.3.1 Addition of record

When a mobile phone wants to share its records with other nodes in the network, it registers the key of the records to its base station. During the registration it sends the keys to the base station. The base station hashes the key using hash function to produce a corresponding hash value, and finds where the tuple for the particular records should be stored. This is done by comparing the hash value (hash_rec) with the hash value of the nodes (hash_id). If h1 is the hash of the record key, a node with its node_id's hash h2 should be found such that h2>=h1.

A list of base station identifiers and their corresponding hash value is maintained. The list is sorted in ascending order on the basis of hash value. This is because, in the network overlay, the order of the base stations in the ring is arranged according to the order of the hash value. The hash values are considered as the identifiers of the base stations in this network. To compare and find out the corresponding base station for a given key, a search similar to a binary search is be used. The searching process work as follows.

- If the hash value (hk) of key (k) is less than or equal to the hash (identifier) of the first base station, then hk lies in that base station.

- Similarly, if the hash value (hk) of key (k) is greater than the last base station in the list, then hk lies in this base station.

- Else, the hash (hk) of key (k) is compared to check if it lies between the two base stations (m1 and m2) that lie near the middle position among the base stations in the list.

    - If the hk > m1 and also hk<=m2 , the hk belongs to the base station m2.

    - If hk <=m1, search is carried out in the sublist consisting of the base stations starting from low + 1 upto mid1.

    - If hk> m2, search is carried out in the sublist consisting of the base stations starting from mid2+1 upto high. Here, low is the position of the first base station in the list/ sublist of interest; while high is the position of the last base station.

After finding out the node, which should own the tuple for the record with the given key, the tuple <hash value, key, id of the requesting mobile> can be routed to that node (base station), using an efficient routing method. When the tuple is routed and dropped in the corresponding node, it is stored in its bucket. The arrangement of the stored tuple may be ordered or unordered. Whatever the arrangement method might be, the tuples maps the hash value to the mobile node_id. In the simulation, the undordered list of records has been used. The tuples would be appended at the end. In this case, when one searches for a particular tuple, sequential search is performed. But sequential search has searching complexity of O(n).

For improved efficiency, searching algorithm and efficient data structures could be used. One method could be sorting the tuples in ascending or descending order of the hash values, and applying binary search. This way, the searching complexity would reduce to O(log n).

Another method could be indexing using B Trees. B-Tree could be used if the number of record-tuple is too large and wish to store and retrieve them in a disk file rather than internal memory. B-Tree helps to minimize the number of disk accesses. The upper bound searching time complexity or the number of disk accesses in B-tree of order M with N keys is given by $(1 + \log_{[M/2]}(N/2))$.

3.3.3.2 Retrieval of record

When any mobile device user wants to search for information, she sends a query to her base station. The query includes a record key. When the base station receives this request, the key is hashed into a hash value. If the value matches its own id's hash value, it will search its own bucket. Else, the hash value is compared to find out the base station where the corresponding mapping tuple for the given key resides. This is again done using the above searching scheme. It will pass on the request to this base station. Once a base station finds out that the key's hash is in its bucket, it searches its bucket and finds the corresponding mobile device id from the list of <value, key, mobile_id>. When the mobile_id is found, it returns this id to the requesting base station. The base station then obtains the required record from this mobile id, by sending a request to the base station that is controlling this mobile. On the other hand, the later base station communicates with this mobile to get the record. A series of communication between intermediate base stations may be necessary to return the required record to the requesting base station. The routing of the message and the record is done, again, using the routing method. When the requesting base station gets the record, it is sent to the requesting node, which is under its range.

3.3.4 Hashing

SHA 1, acronym of Secure Hash Algorithm, is a cryptographic hash function. It generates an almost-unique 160-bit (20-byte) signature for a text. It is a 'one-way' cryptographic function, and is a fixed size for any size of source text. This makes it suitable when it is

appropriate to compare ‘hashed’ versions of texts. SHA-1 is one of the most secure hash algorithms, and has been used in many security applications and protocols.

CHAPTER 4

DESIGN AND IMPLEMENTATION OF SIMULATOR

4.1 Discrete Event Simulation

4.1.1 Simulation

It is a technique to imitate the behaviors or operations of various kinds of real world facilities or process. The process or facility of interest is generally called a **system**. Simulation is a fast and relatively inexpensive method of doing an experiment on a system to produce the basis for making some decision regarding the system's alteration - the decision being based on the results of the simulation. Certain parameters and components of the system to be studied are gathered. Then a **model** of the system is built in such a way that the model satisfies the required specifications of the system. The system's performance is predicted from the knowledge of the model's behavior. If the predicted performance compares favorably with the desired performance, the designed model is accepted. Else the system is redesigned and the process is repeated. Hence, simulation is done and used before an existing system is altered or a new system is built, to reduce

- the chances of failure to meet specification
- to eliminate unforeseen bottlenecks
- to prevent under or over utilization of resources
- to optimize system

4.1.2 Basic System-Concepts

A system exists and operated in time and space. It is bounded inside a **system boundary**. A system may often be affected by changes occurring in the **system environment**. Alternatively, some changes inside the system may also affect the system environment and not the system itself. In a system, there are certain distinct **entities** (or objects), each of which possesses **attributes** (or properties) of interest. The description of all the entities and their corresponding attributes comprises the **state of the system**. The entities interact

with each other causing some changes in the description of the entities, attributes. Such processes or interaction that occurs at a point in time, which may change the system state, are called activities or events. The approach of simulation depends on the type of the system to be studied. A system is called a **discrete system** if the state variables in the system changes instantaneously at separate points of time. That is, the changes in the system are predominantly discontinuous. To simulate such systems, a technique called **Discrete Event Simulation** may be used.

4.1.3 Components of Discrete Event Simulation

In discrete event simulation, the system activities are represented as a chronological sequence of **events**. Each event occurs at an instant of time. The event causes change in the system state. In addition to the representation of system state variables and the logic of what happens when system events occur, discrete event simulations include the following components:

Clock

A number referred to as a clock time tracks the passage of time during simulation. The clock time is zero at the beginning of the simulation. The value of the clock time skips or hops to the next event start time as the simulation proceeds. Hence, the clock indicates the units of elapsed simulation time. The measurement units may be specified in the way that is suitable for the system being modeled. The discrete event simulation's purpose is to study a complex system by computing the times that would be associated with real events. There are basically two ways to update the time.

- In a method called **event-oriented** method, clock is advanced to the time at which the next event is due to occur. The system does not wait for the delays between events to occur in the real time.
- A second method is called an **interval-oriented** method. Here, the clock is advanced by small intervals of time and at each interval it is determined whether an event is due to occur at that time interval.

Event Notice and Event List

It is a record of an event to occur at the current or some future time, along with any associated data necessary to execute the event. Event list is a list of event notices for future events, ordered by the time of occurrence. The simulation maintains at least one list of simulation events. An event must have a start time, some kind of code that constitutes the performance of the event itself, and possibly an end time. In some approaches, there are separate lists for current and future events. Events in their lists are sorted by event start time. Typically, events are **bootstrapped** – that is, they are scheduled dynamically as the simulation proceeds. In other words, bootstrapping is the process in which an ongoing event schedules another event to occur in some future simulation time.

Ending Condition

A discrete event simulation can be said to run forever since the events are bootstrapped. Therefore, the end of the simulation must be decided in advance. Common method to identify an end condition are - “at time t” or “after processing n number of events” or, more generally, “when statistical measure X reaches the value x”.

Random Number Generators

The simulation needs to generate random variables of various kinds, depending on the system model. A **random variable** is a quantity that is uncertain or which cannot be predicted. The generation of such quantity is accomplished by one or more pseudorandom number generators.

Statistics

During the course of simulation, it is necessary to keep track of the system's statistics. The statistics quantify the aspects of interest. The exact statistics required for a model depends upon the analysis being performed. Some of the common statistics are counts (a quantity giving the number of entities of some type or a number of times some event occurred etc), Utilization (a quantity which defines the fraction of the time some entity is

engaged), occupancy (a quantity which defines the fraction of a group of entities in use on the average) etcetera.

4.1.4 Execution Mechanism of discrete event simulation

A discrete-event simulation executes as follows :

Start

- Initialize Ending Condition to FALSE.
- Initialize system state variables.
- Initialize Clock (usually starts at simulation time zero).
- Schedule an initial event (i.e., put some initial event into the Events List).

Loop until ending condition is FALSE, do the following

- Set clock to next event time.
- Do next event and remove from the Events List.
- Update statistics.

End the simulation

- Generate statistical report.

4.2 Simulation of DHT

A cellular network simulator has been implemented to simulate the above discussed distributed hash table. The simulation is based on the concepts of discrete event simulation. With regards to the searching algorithms within each bucket, the above-mentioned search algorithms have not been applied; instead a sequential search has been use. Much concept has been borrowed from a simulator called ASSET, Ad hoc System Simulator and Enhancement Testbed, a result of a project at iiit-b, India. ASSET was designed with an aim to provide a Simulator for ad hoc systems where one can simulate a network and test new protocols. ASSET simulates the creation of nodes in an ad hoc network, movement of these nodes and positioning of the nodes. The system calculates the position of the nodes based on real and virtual co-ordinates and verifies the error

factor. Such properties of ASSET have been incorporated in this DHT simulation to simulate the creation, movement and location of mobile nodes.

The code for the simulator has been written in an objected oriented programming language JAVA.

The class diagram along with the descriptions of their major functions (methods) that has been built is as follows.

4.2.1 Class Description

1. Simulator

   The simulation execution begins here. It consists of an object of Configurator class. It initiates the enqueing of events in the event queue.

   Important attributes:

   - nodespace : object of type Space
   - ConfObj : object of type Configurator
   - dhtObj : object of type DHT
   - EventQ : a linked list of the events occuring in the cellular network
   - Localtime: the timestamp

   Important method details

   - Simulator (Space s, Configurator co) : it sets nodespace = s and ConfObj = co
   - Run ( ) : it sets the localtime = timestamp for the next event in the Event queue and then executes that event by calling the happen( ) method
   - addEvent(Event e): Adds an event to EventQ in its place iff its time stamp is below timelimit
   - main(String args[]) does the following :

creates objects for configurator , space , simulator . The simulation starts with the command ‘start’. The node creation event is added to the event queue. If number of nodes is specified then the node creation events will be enqued to the event queue .The number of events depends on the number of nodes specified or the udensity factor. The simulation stops at the time = timelimit (parameter set in the Configurator class). To stop the simulation before this time issue the command 'quit' may be used.

2. Configurator

   It configures various parameters (such as number of nodes, time limit, radius, udensity, space size etcetera) for the simulation. The configurations decide the behaviour of the system. The parameters are fed in the form of a file - The Configurator file where the values for each parameter are stored beforehand.

3. Space

   This is the cellular network space. This is the entire bounded space where the cellular network system is simulated. The space has collection of mobile devices (represented by **Node** class). The space is divided into cells or **Chunk**s. Each chunk has a fixed **BaseStation**.

   Important attributes are:

   - xmin , ymin , xmax , ymax : The coordinates which bound the space under consideration.
   - Nodelist : a linked list of all the nodes in the space
   - Chunklist : a linked list of the chunks in the space
   - ConfObj : an object of type Configurator
   - bsList : a linked list of base stations
   - TotalBS : integer variable that stores the number of basestation

Important methods

- Space(Configurator co) : It makes a linked list of chunks.Their ids vary from 0 till n-1 , where n depends on the following parameters. The boundaries of the space (xmin,ymin) and (xmax,ymax) and the chunksize. All these parameters are obtained with the help of the the object co.

- int determinechunk(int x, int y) : For the node located at position (x,y) , the method returns the id of the chunk to which the node belongs to. This calculation depends on the node's position, the chunksize and the boundary of the space.

4. Chunk

These are the partitions made in the Space. They are square regions of size = chunksize * chunksize, chunksize = radius. Each chunk consists of a Base Station. To ease the implementation, the base station id and the chunk id have been made same.

Its attributes:

- num : Gives the chunk number.
- LinkedList Nodelist : It is a linked list of all the nodes included in the chunk.

An important method:

- int getnum( ) : returns the chunk number

5. Node

The node class represents mobile device. Node has position, unique id, network id. Each Node knows about its immediate neighbors and even knows their location. Each Node maintains a list of its neighbor nodes.

Some method details :

- int getx( ) : It returns the x coordinate of the node

- int gety( ) : It returns the y coordinate of the node
- int getid( ) : It returns the Node id.
- addConfigurator(Configurator co) : It sets the ConfObj = co
- int getchunk( ) : It returns the chunk
- int getnetid( ) : It returns the Node's network id 'nid'
- int getBS() : It returns the base station id of the node

6. Event

   It stands for any event happening in the space. Node creation, Node movement, Record addition, Record retrieval is all events.

   Its attributes

   - timestamp : the time at which an event occurs since the starting of the simulator.

   Some method details:

   - Event( ) : It instantiates the Event class
   - Event(int ts) : sets the timestamp = ts
   - int gettimestamp( ) : It returns the timestamp of the event
   - settimestamp(int ts) : It sets the timestamp = ts
   - abstract void happen( ) : an abstact method .It will be overridden in all the other classes that inherit the Event class.

7. NodeCreation

   This is an event that creates a new node in the space.

   Its attributes are as follows:

   - SimObj : an object of type Simulator
   - nodespace : an object of type Space
   - n : an object of type node
   - dh : an object of type DHT
   - ConfObj : an object of type Configurator
   - x , y : coordinates of the node which has to be created

Some important method details

- happen( ) : It will create a node by calling an appropriate constructor of the node class. It then finds out the neighbour list of the node. It then adds node movement event at timestamp = timestamp + mf. Similarly, the AddRecord event and the RetrieveRecord event are also added at timestamp= timestamp + constant. All of these events are added in the event queue of the Simulator by calling the method SimObj.addEvent(event).

- createFile( ): A file is created which stores the records of interest belonging to that particular node.

8. NodeMove

   This is an event that relocates a node from one position to another.

   Its attributes:

   - SimObj : an object of type Simulator
   - nodespace : an object of type space
   - n : an object of type node
   - ConfObj : an object of type Configurator

   Method

   - happen( ) : It relocates the node by calling the method relocate( ) with the help of the object nodespace. It recomputes node's neighbours and displays them. It then creates a new object of the type NodeMoveEvent and adds it to the event queue using the object SimObj.

9. AddRecordEvent

   This event registers a new record in the network such that it is accessible to the other members in the network. A node with a new record initiates this event to add the tuple <hash_value of record key, record_key, node_id> in the distributed hash table. A text file called “CommonText.txt” has been used to pick an arbitrary record as a key, since we don’t have facility for users to give an input key.

Important attributes included are as follows:

- ndespace : object of type space
- node : object of type node
- buck : object of type bucket
- dht : object of type dht
- SimObj : object of type Simulator

Methods included are as follows:

- addRecordEvent : Initializes the objects nodespace , node, dht and SimObj.

- happen : simulates the addition of new record in a mobile device, its registration in the basestation. It then calls the function called register() which registers new key in base station. Finally, it adds a new AddRecordEvent in the event queue.

- register: It takes a key and record as an input. The key is hashed to find the corresponding base station to store the tuple <value, key, node id>. Once base station is found, it stores the tuple in the bucket of that base station.

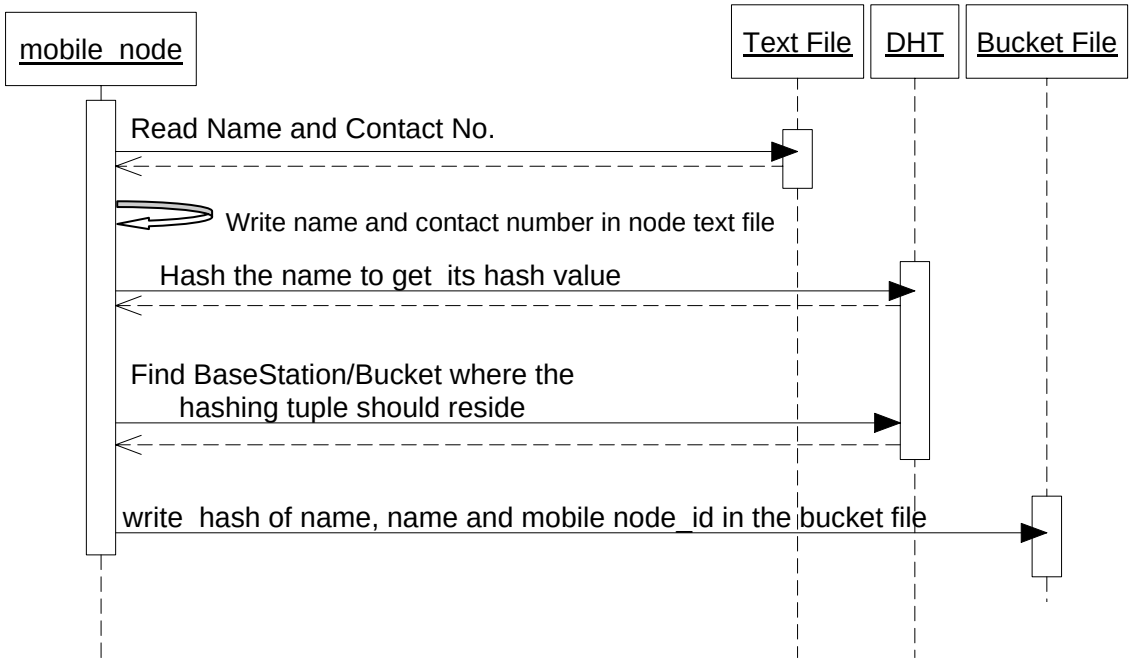

Figure 4.1: Data addition

10. Retrieve Record Event: It retrieves the existing record from a node (mobile device) and returns it to the mobile device requesting it.

    Included important attributes are as follows:

    - node: Object of type node
    - s : Object of type Space
    - dht : Object of type DHT
    - SimObj : Object of type Simulator

    Some major methods:

    - retrieveRecordEvent : initializes s, node, dht and SimObj

    - routeString: It takes destination basestation and string to be routed (i.e. the requested record) as an argument, then routes the string to the destination base station using the routing algorithm discussed above. It then calls the writeDestNode function to write the record in the destination node.
    - writeInDestNode: It takes the string (requested record) as an argument and writes it to the destination mobile node (the requesting mobile).

    - readString: finds, reads and returns the required record

    - getSrcNodeId(int bs,String recdName): It searches and returns the id of the mobile node where the requested record resides.

    - find_buck: It takes a key of a record as an input, hashes it, finds the corresponding base station that stores the tuple <value, key, node id> for a given record key. It then returns the hash_id of that basestation.

- happen : The happen function directly or indirectly calls all the above function in order to retrieve a record from a source node and return it to the requesting node. The sequence diagram of its flow is as follows:

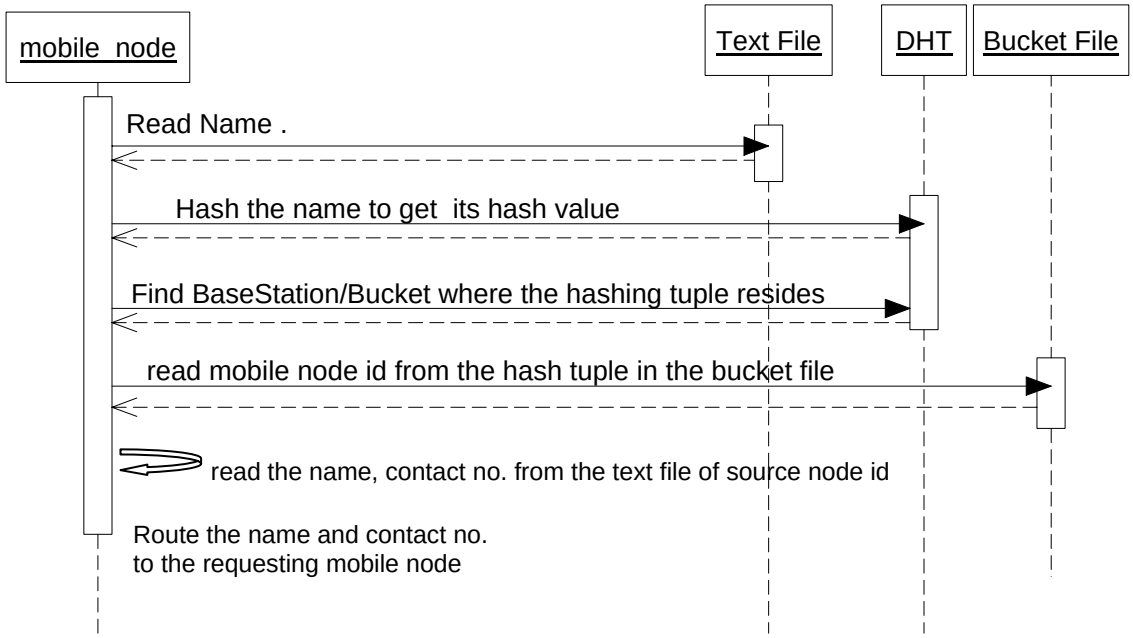

Figure 4.2: Data retrieval

11. Bucket Class

A bucket class keeps the list of tuples <value, key, node_id> where value is the hash value of a record key, key is the record key and node_id is the identifier of the mobile device which consists of the actual record of the key. The size of the bucket is not fixed. The newly registered record-tuple is appended at the end of the tuple-list. Each bucket has a file to store this list of tuples.

Some important attributes:

- bucket_id : an integer that uniquely identifies a bucket. This value equals to the base station to which the bucket belongs.

Important methods included in this class are as follows:

- createFile : This takes basestation identifier, say i , as an argument. Then it creates a file for the bucket belonging to base station with id i.
- addtoBucket : It takes record string and bucket id, say i as an argument. It, then, appends the record string at the end of the file belonging to the bucket with id i.

12. DHT Class

It consists of a functions, which, simply, takes a key and hashes it to return the value of the key. These functions would be used for both registering new records as well as retrieving records. For hashing purpose, SHA-1 hashing algorithm has been used. So the output value will be a 16-bit string.

Some attributes included:

- nodespace : Object of type Space
- name : a string to be hashed
- bucketid : an integer type data to store bucket identifier
- bslist[][] : an array of base station id along with their hash_id
- totalbs : an integer to store number of base stations

Some important methods included in this class are as follows:

- DHT : It's a constructor where member variables are initialized. It initializes the bsList [][] by reading the ids of all the basestation and converting them to their corresponding hash_id. The pair (hash_id , id) are stored in this list and the list is sorted in the ascending order of hash_id.

- hashKey : It takes a string as an argument and hashes it to produce a 16-bit string hash value. This value returned after converting into an integer.

- getHashValue: It again takes a string as an argument and hashes it to produce a 16 bit string hash value. It returns this value without converting it to an integer.

- hashBS: It takes an integer (base station identifier) as an argument and hashes it to produce a 16-bit string hash value. This value returned after converting into an integer.

- find_successor: It takes a "key" as an input and calls bsrch function to get a successor's id of the input "key".

- bsrch: It takes key, lower bound, upper bound of bsList array in argument. It finds the successor of the key and returns it using the algorithm discussed in section 3.3.3.1.

- getPos: It takes base station hash id as an input and returns its position (index) in bsList[ ][ ] if it exist, -1 otherwise.

- getId : It takes base station hash id as an input and returns its corresponding id if it exist, -1 otherwise.

13. The Base Station(BS) Class

Its data members includes

- bs_id: identifier which is equal to its chunk id
- bs_range: its range which is equal to the chunksize
- buck: Object of type bucket
- ConfObj: Objects of Configurator classs

Various functions of this class includes

- BaseStation: initializes bs_id to n, bs_range to the chunksize and creates file for this particular basestation's bucket by calling createFile() function of the bucket class..

4.3 The class diagram

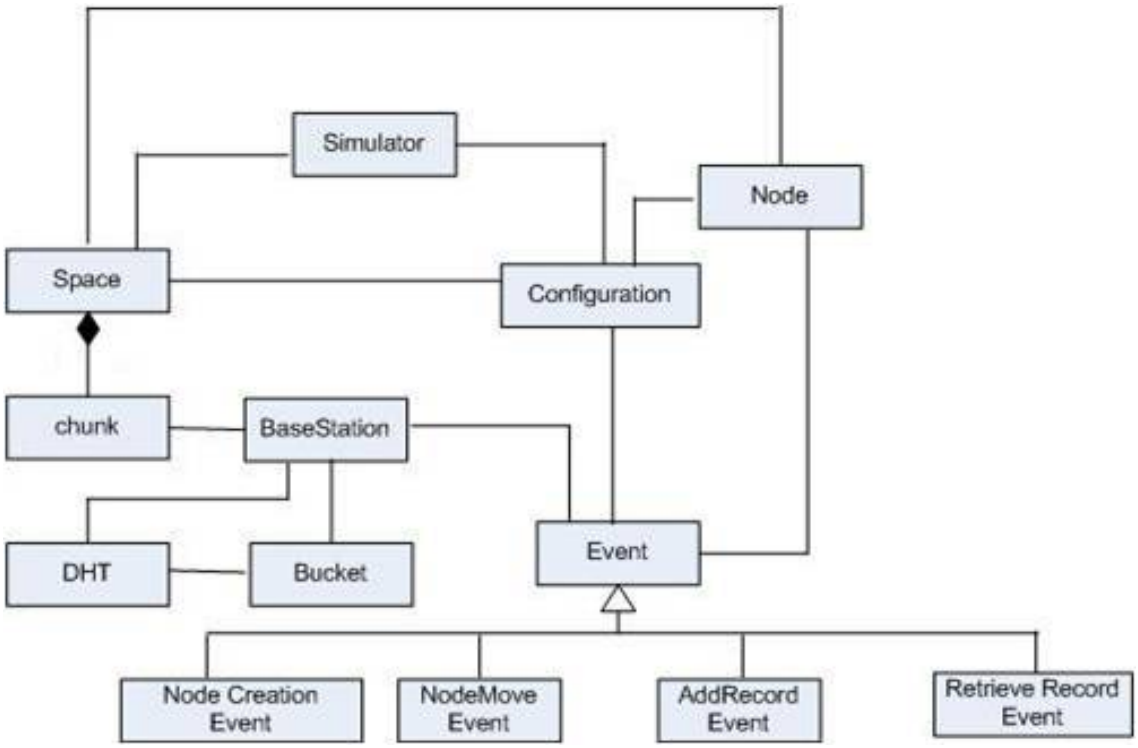

Figure 4.3: Class Diagram

CHAPTER 5
CASE-STUDY

5.1 Simulation Run

The simulation is done for one of the application mentioned above, i.e. the address book application.

The simulation starts with the creation of 2D node space of the specified dimension. The space is partitioned into number of cells with a specified radius and basestation in each cell. A record file is also created along with the base stations to store hashing-tuples. The simulation runs until the user quits from the simulation.

Once the base stations are created, the DHT is created which consists of a record of all the base stations' identifier and their corresponding hash identifiers too.

Users can specify number of mobile nodes using a parameter called *numnodes*, which is the number of mobile nodes that have to be created with a normal distribution around the center of the node space.

The simulation is a discrete event simulation, hence it is event driven. The first event added in the event queue is a node creation event which creates new mobile node. This event bootstraps other events like node movement event, add record event and retrieve record event which again adds new events in the event queue. The events occur periodically.

5.2 Output

Some of the views of output of this simulation are as follows:

Starting output : Creation of BaseStation (Nodes) and DHT

```
Welcome to the prototype version of SIMULATION of DHT

BS created with an id : 0
BS Range : 5
Bucket Created - filename: Bucket0.txt
BS created with an id : 1
BS Range : 5
Bucket Created - filename: Bucket1.txt

BS created with an id : 2
BS Range : 5
Bucket Created - filename: Bucket2.txt

BS created with an id : 3
BS Range : 5
Bucket Created - filename: Bucket3.txt

BS created with an id : 4
BS Range : 5
Bucket Created - filename: Bucket4.txt

BS created with an id : 5
BS Range : 5
Bucket Created - filename: Bucket5.txt

BS created with an id : 6
BS Range : 5
Bucket Created - filename: Bucket6.txt

BS created with an id : 7
BS Range : 5
Bucket Created - filename: Bucket7.txt

BS created with an id : 15
BS Range : 5
Bucket Created - filename: Bucket15.txt

Sorted List of basestaion hash_ids with corresponding ids created
Hash_id           Id

12          0
18          15
31          8
59          14
75          11
84          12
120         6
122         4
151         9
```

```
171         1
176         2
187         3
196         5
218         7
229         10
244         13
node#> start with numnodes
```

Creation of Mobile Nodes

```
Base station for node 8 is 10
0: Created a new node with id = 8and cordinates = (0, 0) and chunk = 10
0: Neighbours(8) = {0, 1, 2, 3, 4, 5, 6, 7}
Record File created: NodeFile8.txt
AddRecord Event added in the event queue.
RetrieveRecord Event added in event queue.

Base station for node 9 is 10
0: Created a new node with id = 9and cordinates = (0, 0) and chunk = 10
0: Neighbours(9) = {0, 1, 2, 3, 4, 5, 6, 7, 8}
Record File created: NodeFile9.txt
AddRecord Event added in the event queue.
RetrieveRecord Event added in event queue.

Base station for node 10 is 5
0: Created a new node with id = 10and cordinates = (-3, -3) and chunk =
5
0: Neighbours(10) = {}
Record File created: NodeFile10.txt
AddRecord Event added in the event queue.
RetrieveRecord Event added in event queue.

Base station for node 11 is 5
0: Created a new node with id = 11and cordinates = (-2, -3) and chunk =
5
0: Neighbours(11) = {10}
Record File created: NodeFile11.txt
AddRecord Event added in the event queue.
RetrieveRecord Event added in event queue.
```

Registering new record

```
_____________________________________________

Event to add new record in a DHT
The hash value for key "Anil" is 181
Successor Found!! Successor of key  181 is the bs 187
Record has been registered.
New AddRecord Event added in the event queue.

_____________________________________________

Event to add new record in a DHT
The hash value for key "Achyut" is 218
Successor Found!! Successor of key  218 is the bs 218
Record has been registered.
New AddRecord Event added in the event queue.
```

Retrieving Existing Record

```
_____________________________________________

This event retrieves record
The hash value for key Anil is 181
Successor Found!! Successor of key  181 is the bs 187
String to retrieve Anil is in base station with hash_id 187, its id is
3
The retrieve event has been generated by node 9 whose basestation is
10, basestation hash_id is 229
****Record of Anil found in the node 35.****
Destination BS reached.
Anil 5544259 added in destination NodeFile 9.txt
New RetrieveRecord Event added in event queue.

_____________________________________________

This event retrieves record
The hash value for key Achyut is 218
Successor Found!! Successor of key 218 is the bs 218
String to retrieve Achyut is in base station with hash_id 218, its id
is 7
The retrieve event has been generated by node 2 whose basestation is
10, basestation hash_id is 229
****Record of Achyut found in the node 1.****
```

```
Destination BS reached.
Achyut 9841253534 added in destination NodeFile 2.txt
New RetrieveRecord Event added in event queue.
```

```
This event retrieves record
The hash value for key Kabindra is 173
Successor Found!! Successor of key  173 is the bs 176
String to retrieve Kabindra is in base station with hash_id 176, its id
is 2
The retrieve event has been generated by node 59 whose basestation is
1, basestation hash_id is 171
No record available
New RetrieveRecord Event added in event queue.
```

Base Station Bucket File

```
Mukunda 206 22
Achyut 218 29
Pushpa 210 35
Rajiv 232 31
Kamal 253 42
```

Mobile Node File

```
Bhaskar 9841342145
Anil 5544259
Rashmi 4427793
Dinesh 9841355340
Achyut 9841253534
Mukunda 4222222
```

CHAPTER 6
CONCLUSIONS AND FUTURE WORKS

In this dissertation, a distributed indexing mechanism for a cellular network has been presented. The indexing method being hash table, and the distribution being the division of the hash table into parts and placing them at different nodes that are the members of cellular network. The work was focused on building a data centric search of records. A mobile finds the needed records stored in one of the devices within a group/network, without getting connected to them one by one. Thus, the concept of data centric search was achieved.

However, many issues are there which are still remained to be addressed. Some of them are briefly discussed below.

The storage structure, record structures or schema of the actual records stored in the mobile devices have not been dealt with. If the data are supposed to be shared, it is important to have common schema, or else proper interface becomes necessary. Similarly, there must be a common method/syntax of querying. Also different field in a record should be able to act like a key rather than a single field.

Different copies of same record may be available in the network. There must be a mechanism to maintain the consistency of such redundant data.

Security issue has not been worked out. Also the practical issues regarding the system have not been worked out and are still left to be studied.

REFERENCES

[1] Sanket Patil, Srinath Srinivasa. A Data centric Abstraction Middleware for Mobile Networks. International Institute of Information Technology – Banglore, India, 2006.

[2] Napster's High and Low Notes - Businessweek - August 14, 2000

[3] http:// www.gnutella.com

[4] Ion Stoica, Robert Morris, David Karger, M.Frans Kaashoek, Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Services for Internet Applications. August 2001.

[5] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. November 2001.

[6] Sanket Patil, Srinath Srinivasa, Shrisha Rao. Distributed Garbage Detection of Replicated Objects in Mobile Networks. A Technical Report. International Institute of Information Technology – Banglore, India. May 29, 2006.

[7] Sylvia Ratnasamy, Paul Trancis, Mark Handley, Richard Karp, Scott Shenker. A Scalable Content Addressable Network. In proceedings of ACM SIGCOMM. 2001.

[8] Egin Kirda, Pascal Renkam, Gerald Reif, Herald Gall. A Service Architecture for Mobile Teamwork. In Proceedings of DEKE, 2002.

[9] G. Reif, E. Kirda, H. Gall, G.P. Picco, G. Cugola, and P. Fenkam. A Web-based peer-to-peer architecture for collaborative nomadic working. In Proceedings of the 10th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), Boston, MA, USA. IEEE Computer Society Press, June 2001.

[10] Scott Shenkar. The Data-Centric Revolution in Networking. Proceedings of the International Conference on Very Large Database, 2003.

[11] http://pdos.csail.mit.edu/~strib/docs/tapestry/tapestry_jsac03.pdf

[12] http:// computer.howstuffworks.com/ file-sharing1.htm

[13] http://www.devx.com/architect/Article/34158/0/page/2

[14] http://www.itl.nist.gov/fipspubs/fip180-1.htm

[15] Sanket Patil. Localization and Garbage Collection in Ad Hoc Networks. Master's Dissertation, International Institute of Information Technology – Banglore, India. June 2005.

[16] Vagner Sacramento, Markus Endler, Hana K, Rubinsztejn, Luciana S. Lima, Kleder Goncalves, Fernando N. Nascimento, Giulliano A. Dueno. MoCA: A middleware for developinig Collaborative Application for Mobile Users. IEEE Distributed System. Online October 2004 Vol.5 No.10.

BIBLIOGRAPHY

1. William Stallings. Principles of Cellular Network in Wireless

Communication and Networking,. Isbn: 81-203-2386-6, Eastern Economy Edition, Prentice-Hall Pvt.Ltd India.

2. Theodore S. Rappaport. Wireless Communication Principles and Practice. Second Edition. Prentice-Hall Pvt.Ltd India.

3 Narsingh Deo. System Simulation with Digital Computer. Isbn: 81-203-0028-9, Prentice-Hall Pvt.Ltd India.

4 Goeffrey Gordan. Discrete System Simulation in System Simulation. Isbn: 81-203-0140-4, Second Edition, Prentice-Hall Pvt.Ltd India.

5 Remez Elmasri, Shamkant B. Navathe. Fundamentals of Database Systems, Fourth Edition.

6 http:// www.gnutella.com

7 http:// www.napster.com

8 Himabindu Pucha, Saumitra M. Das and Y. Charlie Hu. Poster. How to Implement DHT in Mobile Ad Hoc Networks? August 2004.

9 Sylvia Ratnasamy, Scott Shenker, Ion Stoica. Routing Algorithms for DHTs: Some Open Questions. 2002.

10 Frank Dabek. A Distributed Hash Table. Massachusetts Institute of Technology. September 2005.