P-2565

SHARING LOCATION DEPENDENT EXPERIENCES IN MANET

By

P. SHARAFNA

Reg. No. : 0720108018

of

KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641 006

(An Autonomous Institution Affiliated to Anna University Coimbatore)

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements
for the award of the degree
of

MASTER OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

MAY 2009

BONAFIDE CERTIFICATE

Certified that this project report titled "**Sharing location dependent experiences in MANET**" is the bonafide work of **Miss.P.Sharafna (0720108018)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]

SIGNATURE OF THE GUIDE

Mrs.Amutha Venkatessh M.E.,

Senior Lecturer,

Department of Computer

Science and Engineering

[illegible]

HEAD OF THE DEPARTMENT

Dr.S.Thangasamy Ph.D.,

Professor and Dean,

Department of Computer

Science and Engineering

The candidate with **University Register No. 0720108018** was examined by us in Project Viva-Voce examination held on 05-05-09

[illegible] 5/5/09

Internal Examiner

[illegible] 5/5/09

External Examiner

SNS COLLEGE OF TECHNOLOGY

Approved by AICTE and Affiliated to Anna University
(An ISO 9001:2000 Certified Institution)

Department of Information Technology
&
Society for Information Sciences and Computing Technology

Third National Conference on

NETWORKS,INTELLIGENCE AND COMPUTING SYSTEMS

This is to certify that Dr./Prof./Mr/Ms........ P.Sharafna M.E. CSE ....

of Kumaraguru College of Technology .... has Presented a technical paper

titled .... Location Dependent Experiences in MANET ....

in the National Conference NCNICS held on 16th February 2009

Convenor — [illegible] — Chief Secretary

Prof. [illegible] — [illegible] — S. N. Subbramaniam

ACKNOWLEDGEMENT

I express my sincere thanks to our Chairman **Padmabhu.han Arutselvar Dr. N.Mahalingam B.Sc, F.I.E** and Correspondent **Shri. Balasubramanian** for all their support

I would like to begin by thanking to **Dr. Joseph V. Thanikal, Ph.D.,** Principal and **Prof.R.Annamalai** , Vice-Principal for providing the necessary facilities to complete my project work.

I also thank **Dr.Thangasamy, Ph.D.,** Professor and Dean, Head of the department of Computer Science and Engineering for his valuable advice and useful suggestions throughout this project.

I express my heartiest thanks to my Project Guide **Mrs.Amutha Venkatessh M.E.,** Senior Lecturer, Department of Computer Science and Engineering, who rendered her valuable guidance and support to perform my project work extremely well.

I would express my heartiest thanks to our Project Coordinator as well as my Project Guide **Mrs.V.Vanitha M.E.,** Assistant Professor, Department of Computer Science and Engineering, who rendered her valuable guidance and support to perform my project work extremely well. I also thank the teaching and non-teaching staffs of our Department for providing us the technical support in the duration of my project.

I also thank my parents, sister and friends who have supported me and helped me to complete the project work.

Last but not least, I express my heartfelt thanks to **The Almighty** for the blessings. Without his permissions and blessings it would not have been possible to complete this project work.

ABSTRACT

MANET consists of a number of mobile hosts that communicate with each other through wireless packet relaying. MANET is excellent for sharing information in areas lacking of communication infrastructures such as disaster areas, combat zones, or unexplored territories (e.g., deep space or deep sea). In these scenarios, mobile hosts scatter around to observe various parts of a large unfamiliar territory and record their experience in their local storage.

This projectwork investigates a new problem of sharing location dependent experiences among mobile hosts in mobile ad hoc networks. An experience is location and observer dependent. In other words, experiences of different people witnessing the same event may be quite different. The ability to retrieve prior experiences observed in a given area in advance is very useful for newcomers wishing to enter the same vicinity. Solving this problem is vital for important applications such as hurricane rescue missions, combat missions, and deep space or deep sea exploration. The user of the mobile host reviews these experiences to determine the course of actions, depending on a specific application. In this projectwork, we propose a distributed solution that lets mobile hosts share their experiences efficiently.

ஆய்வுச் சுருக்கம்

நடமாடும் தன்மைக்கான வலையமைப்பில் எண்ணற்ற நடமாடும் கணுக்கல் ஒன்றோடு ஒன்று கம்பியில்லா தகவல் பெட்டக தொடர் ஓட்டம் மூலம் தகவல்கள் அனுப்புகின்றன. பேரழிவுப் பகுதிகள், போர்வட்டார வளையங்கள்,ஆராயாத எல்லைகள்(எ.கா., ஆழ்கடல் (அ) ஆழ்விண்வெளி) போன்ற தொடர்பு உள்கட்டமைப்பில்லா பகுதிகளில் தகவல் பகிர்ந்துகொள்ள நடமாடும் தன்மைக்கான வலையமைப்பு மிகச்சிறந்தவையே. இப்படிப்பட்ட சூழ்நிலைகளில், நடமாடும் புரவலர்கள் மிகப் பெரிய பழக்கமில்லாத எல்லையின் பல பகுதிகளை கவனிக்கவும், தங்களுடைய உள்ளூர் தேக்கத்தில் தங்கள் அனுபவங்களை பதிவு செய்யவும் சுழ்ந்து பரவுகின்றன.

இந்த ஆய்வு நடமடும் தன்மைக்கான வலையமைப்பில் நடமாடும் கணுக்களிடையே இடத்தை பகிர்ந்து கொள்ளும் அனுபவத்தை சார்ந்துள்ள பிரச்சனைகளை ஆராய்வது ஆகும். அந்த அனுபவம் இடத்தையும் கவனிப்பாளரையும் சார்ந்தது. இந்த அனுபவம் ஒவ்வொருவருக்கும் மாறுபடும். இதில் முன்னதாக ஏற்பட்டுள்ள அனுபவத்தை வைத்து புதிதாக வருபவகர்கள் பக்கத்து வலையமைப்பில் நுழைய உதவுகிறது.

இந்த பிரச்சனையை தீர்ப்பதன் மூலம் சூறவளி மீட்ப்பு பணி, போர்ப்படை மீட்புபணி, ஆழ்கடல் ஆராய்ச்சிகளில் பயன்ப்படுத்தபடுகிறது.

இதைப்பயன்படுத்துபவர்கள் இந்த அனுபவத்தை மதிப்பீடு செய்து தங்களது அடுத்த கட்ட நடவடிக்கையை மேற்க்கொள்ளலாம்.நடமாடும் கணுக்களின் அனுபவத்தை சிறப்பாக பகிர்ந்து கொள்ளக்கூடிய விநியோ கி க்கும் தீர்வை இந்த ஆய்வில் மேற்க்கொள்ளப்பட்டுள்ளது.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
MANET	Mobile Ad hoc Network
QoS	Quality of Service
RIP	Routing Information Protocol
OSPF	Open Shortest Path First
GCLP	Geography-based Content Location Protocol
TTL	Time-To-Live
NS2	Network Simulator 2
NAM	Network Animator
TCL	Tool Command Language
LODE	Location Dependent Experience

CHAPTER 1

INTRODUCTION

1.1 GENERAL BACKGROUND

Today, many people carry numerous portable devices, such as laptops, mobile phones, PDAs and mp3 players, for use in their professional and private lives. For the most part, these devices are used separately that is, their applications do not interact. Imagine, however, if they could interact directly: participants at a meeting could share documents or presentations; business cards would automatically find their way into the address register on a laptop and the number register on a mobile phone; as commuters exit a train, their laptops could remain online; likewise, incoming email could now be diverted to their PDAs; finally, as they enter the office, all communication could automatically be routed through the wireless corporate campus network.

These examples of spontaneous, *ad hoc* wireless communication between devices might be loosely defined as a scheme, often referred to as *ad hoc* networking, which allows devices to establish communication, anytime and anywhere without the aid of a central infrastructure. Actually, *ad hoc* networking as such is not new, but the setting, usage and players are. In the past, the notion of *ad hoc* networks was often associated with communication on combat fields and at the site of a disaster area; now, as novel technologies such as Bluetooth materialize, the scenario of *ad hoc* networking is likely to change, as is its importance. The most widespread notion of a mobile *ad hoc* network is a network formed without any central administration which consists of mobile nodes that use a wireless interface to send packet data. Since the nodes in a network of this kind can serve as routers and hosts, they can forward packets on behalf of other nodes and run user applications.

In MANETs communication between nodes is done through the wireless medium. Because nodes are mobile and may join or leave the network, MANETs have a dynamic topology. Nodes that are in transmission range of each other are called neighbors. Neighbors can send directly to each other. However, when a node needs to send data to another non-neighboring node, the data is routed through a sequence of multiple hops, with intermediate nodes acting as routers. An example ad hoc network is depicted in Figure (1.1 a).

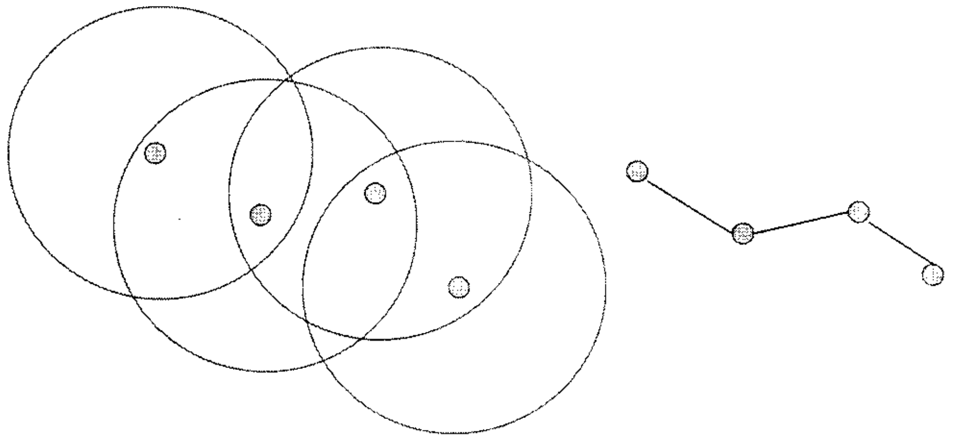


Figure 1.1 An example ad hoc network

1.2 MANET-CHARACTERISTICS AND ISSUES:

In contrast to traditional wireline or wireless networks, an *ad hoc* network could be expected to operate in a network environment in which some or all the nodes are mobile. In this dynamic environment, the network functions must run in a distributed fashion, since nodes might suddenly disappear from, or show up in, the network. In general, however, the same basic user requirements for connectivity and traffic delivery that apply to traditional networks will apply to ad hoc networks. Below, we discuss some typical operational characteristics and how they affect the requirements for related networking functions.

1. Distributed operation:

A node in an ad hoc network cannot rely on a network in the background to support security and routing functions. Instead these functions must be designed so that they can operate efficiently under distributed conditions.

2. Dynamic network topology:

In general, the nodes will be mobile, which sooner or later will result in a varying network topology. Nonetheless, connectivity in the network should be maintained to allow applications and services to operate undisrupted. In particular, this will influence the design of routing protocols. Moreover, a user in the *ad hoc* network will also require access to a fixed network (such as the Internet) even if nodes are moving around. This calls for mobility management functions that allow network access for devices located several radio hops away from a network access point.

3. Fluctuating link capacity:

The effects of high bit-error rates might be more profound in a multihop *ad hoc* network, since the aggregate of all link errors is what affects a multihop path. In addition, more than one end-to-end path can use a given link, which if the link were to break, could disrupt several sessions during periods of high bit-error transmission rates. Here, too, the routing function is affected, but efficient functions for link layer protection (such as forward error correction, FEC, and automatic repeat request, ARQ) can substantially improve the link quality.

4. Low-power devices:

In many cases, the network nodes will be battery-driven, which will make the power budget tight for all the power-consuming components in a device. This will affect, for instance, CPU processing, memory size/usage, signal processing, and transceiver output/input power. The communication- related functions (basically the entire protocol stack below the applications) directly burden the application and services running in the device. Thus, the algorithms and mechanisms that implement the networking functions should be optimized for lean power consumption, so as to save capacity for the applications while still providing good communication performance. Besides achieving reasonable network connectivity, the introduction of multiple radio hops might also improve overall performance, given a constrained power budget. Today, however, this can only be realized at the price of more complex routing.

5. Quality of Service:

The support of multimedia services will most likely be required within and throughout the ad hoc network. As an example, the following four quality-of-service (QoS) classes would facilitate the use of multimedia applications including

1) Conversational (voice);
2) Streaming (video/audio);
3) Interactive (Web); and
4) Background (FTP, etc.).

These service classes have been identified for QoS support in the UMTS network and should also be supported in the PAN environment. However, the inherent stochastic communications quality in a wireless ad hoc network makes it difficult to offer fixed guarantees on the services offered to a device. In networks of this kind, fixed guarantees would result in requirements for how nodes move, as well as requirements for node density, which would inherently inhibit the notion of ad hoc operation. To further improve user perception of the service, user applications that run over an ad hoc network could be made to adapt to sudden changes in transmission quality. QoS support in an ad hoc network will affect most of the networking functions such as security, routing, and mobility. In addition, local buffer management and priority mechanisms must be deployed in the devices in order to handle differentiated traffic streams.

Multihop characteristics:

In dealing with an unreliable wireless broadcast medium, special radio considerations should be addressed in the communication system of an *ad hoc* network, to ensure reliable and efficient operation. One way of doing this is to employ multi-hopping, which facilitates the reuse of resources in both the spatial and temporal domains, provided that the nodes which participate in the network are reasonably well distributed in space. In contrast, single hop networks mainly share the channel resources in the temporal domain.

In a multihop scenario, it makes sense not to waste more energy than what each hop requires. In essence, the key to conserving energy is to control the transmit power, in order to compensate for path losses that occur when a message is sent between adjacent nodes.

MANET-ISSUES:

There are numerous issues to consider when deploying MANETs. The following are some of the main issues.

1. Unpredictability of environment: Ad hoc networks may be deployed in unknown terrains, hazardous conditions, and even hostile environments where tampering or the actual destruction of a node may be imminent. Depending on the environment, node failures may occur frequently.

2. Unreliability of wireless medium: Communication through the wireless medium is unreliable and subject to errors. Also, due to varying environmental conditions such as high levels of electro-magnetic interference (EMI) or inclement weather, the quality of the wireless link may be unpredictable.

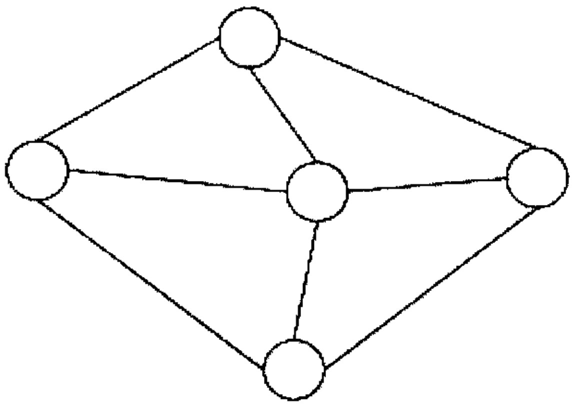


Figure 1.2 An Ad hoc network with nodes in transmission ranges

An example ad hoc network, with circles representing nodes is shown. Two nodes that are in transmission range of each other are connected by a line.
Furthermore, in some applications, nodes may be resource-constrained and thus would not be able to support transport protocols necessary to ensure reliable communication on a lossy link. Thus, link quality may fluctuate in a MANET.

3. Resource-constrained nodes: Nodes in a MANET are typically battery powered as well as limited in storage and processing capabilities. Moreover, they may be situated in areas where it is not possible to re-charge and thus have limited lifetimes. Because of these limitations, they must have algorithms which are energy-efficient as well as operating with limited processing and memory resources. The available bandwidth of the wireless medium may also be limited because nodes may not be able to sacrifice the energy consumed by operating at full link speed.

4. Dynamic topology: The topology in an ad hoc network may change constantly due to the mobility of nodes. As nodes move in and out of range of each other, some links break while new links between nodes are created. As a result of these issues, MANETs are prone to numerous types of faults including:

- **Transmission errors**: The unreliability of the wireless medium and the unpredictability of the environment may lead to transmitted packets being garbled and thus received in error.
- **Node failures**: Nodes may fail at any time due to different types of hazardous conditions in the environment. They may also drop out of the network either voluntarily or when their energy supply is depleted.
- **Link failures**: Node failures as well as changing environmental conditions may cause links between nodes to break.
- **Route breakages**: When the network topology changes due to node/link failures and/or node/link additions to the network, routes become out-of date and thus incorrect. Depending upon the network transport protocol, packets forwarded through stale routes may either eventually be dropped or be delayed; packets may take a circuitous route before eventually arriving at the destination node.
- **Congested nodes or links**: Due to the topology of the network and the nature of the routing protocol, certain nodes or links may become over utilized, i.e., congested. This will lead to either larger delays or packet loss. Routing protocols for MANETs must deal with these issues to be effective.

1.3 TYPICAL APPLICATIONS:

Mobile ad hoc networks have been the focus of many recent research and development efforts. So far, *ad hoc* packet-radio networks have mainly been considered for military applications, where a decentralized network configuration is an operative advantage or even a necessity. In the commercial sector, equipment for wireless, mobile computing has not been available at a price attractive to large markets.

However, as the capacity of mobile computers increases steadily, the need for unlimited networking is also expected to rise. Commercial *ad hoc* networks could be used in situations where no infrastructure (fixed or cellular) is available. Examples include rescue operations in remote areas, or when local coverage must be deployed quickly at a remote construction site.

Ad hoc networking could also serve as wireless public access in urban areas, providing quick deployment and extended coverage. The access points in networks of this kind could serve as stationary radio relay stations that perform *ad hoc* routing among themselves and between user nodes. Some of the access points would also provide gateways via which users might connect to a fixed backbone network.

At the local level, ad hoc networks that link notebook or palmtop computers could be used to spread and share information among participants at a conference. They might also be appropriate for application in home networks where devices can communicate directly to exchange information, such as audio/video, alarms, and configuration updates.

Perhaps the most far-reaching applications in this context are more or less autonomous networks of interconnected home robots that clean, do dishes, mow the lawn, perform security surveillance, and soon.

Some people have even proposed *ad hoc* multihop networks (denoted sensor networks) for example, for environmental monitoring, where the networks could be used to forecast water pollution or to provide early warning of an approaching tsunami.

Short-range *ad hoc* networks can simplify intercommunication between various mobile devices (such as a cellular phone and a PDA) by forming a PAN, and thereby eliminate the tedious need for cables. This could also extend the mobility provided by the fixed network (that is, mobile IP) to nodes further out in an ad hoc network domain. The Bluetooth system is perhaps the most promising technology in the context of personal area networking.

CHAPTER 2

LITERATURE REVIEW

2.1. EFFICIENT CONTENT LOCATION IN MOBILE AD HOC NETWORKS

The rise of computer networks has led to a new paradigm of computing. The standalone, isolated computer has given way to the computer as part of an infrastructure working together to provide users with increased benefits. This has led to an abundance of information and services made available to users by remote servers. A fundamental issue in this environment is efficiently locating needed content. Such content may be in the form of files, services, or any other kind of data. A number of strategies have been proposed to tackle the problem; however, most of these strategies assume a resource-rich network. This assumption does not hold in the case of mobile ad hoc networks which have different characteristics and requirements. In this thesis, an algorithm for efficient content location in location-aware ad hoc networks is described. The Geography-based Content Location Protocol (*GCLP*) makes use of location information to lower proactive traffic while minimizing query cost. This protocol increases the amount of proactive location advertisements while limiting the cost of location queries. The protocol is based on geographical information coupled with simple geometric rules. The results of the analysis show that *GCLP* performs favorably in terms of overhead, latency, and scalability.

2.2. CROSS-LAYER DESIGN FOR DATA ACCESSIBILITY IN MOBILE AD HOC NETWORKS

Mobile ad hoc networks (MANET) are becoming an integral part of the ubiquitous computing and communication environment, providing new infrastructure for multimedia applications such as video phone, multimedia-on-demand, and others. In order to access multimedia information in MANET, Quality of Service (QoS) needs to be considered, such as high success rate to access multimedia data, bounded end-to-end delay, and others. A data accessibility service is presented for a group of mobile users to access desired data with high success rate. This accessibility service is only possible if we utilize advanced data advertising, lookup and replication services, as well as a novel predictive location-based QoS routing protocol in an integrated fashion. Using cross-layer design, it is illustrated how the QoS routing protocol assists data advertising, lookup and replication services to achieve high data access success rate.

2.3. A SURVEY ON ROUTING PROTOCOLS IN MANETS

A Mobile Ad-hoc Network (MANET) is a temporary wireless network composed of mobile nodes, in which an infrastructure is absent. There are no dedicated routers, servers, access points and cables. Because of its speedy and convenient deployment, robustness, and low cost, a MANET can find its applications in the following areas:

(1) Military use (e.g. a network in the battlefield)

(2) Search and rescue

(3) Vehicle-to-vehicle communication in intelligent transportation

(4) Temporary networks in meeting rooms, airports, etc.

(5) Personal Area Networks connecting cell phones, laptops, smart watches, and other wearable computers

If two mobile nodes are within each other's transmission range, they can communicate with each other directly; otherwise, the nodes in between have to forward the packets for them. In such a case, every mobile node has to function as a router to forward the packets for others. Traditional routing protocols used in hardwired networks, such as distance vector protocols (e.g. RIP) and link state protocols (e.g., OSPF) cannot be applied in the MANET directly for the following reasons:

(1) There may be uni-directional links between nodes;

(2) There is more than one eligible path between two nodes;

(3) The consumption of bandwidth and power supply incurred by periodic routing information updates is considerable;

(4) The routing fabrics converge slowly in contrast to rapid topology change.

Most research effort has been put in the routing protocols since the advent of the MANET. They can be divided into the following categories:

(1) Unicast routing protocols

a. Topology-based routing protocols

- Proactive routing protocols
- Reactive routing protocols
- Hybrid routing protocols

b. Geographical-based routing protocols

(2) Multicast routing protocols

(3) Broadcast algorithms

Although there are many kinds of routing protocols competing for unicast, multicast and broadcast communication for the MANET, it seems that one protocol cannot fit all the different scenarios and traffic patterns of MANET applications. For example, proactive routing protocols are well suited for a small-scale, broad-band MANET with high mobility, while reactive routing protocols are well suited for a large-scale, narrow-band MANET with moderate or low mobility. If the mobile nodes in the MANET move too quickly, they have to resort to broadcast to achieve peer-to-peer communication. In a summary, every routing protocol has its strengths and drawbacks, and aims at a specific application. As a result, the prospective standard for routing protocols in the MANET is very likely to combine some of the most competitive schemes.

2.4. PERFORMANCE OF MOBILE AD HOC NETWORK IN CONSTRAINED MOBILITY PATTERN

Mobile Ad Hoc Networks (MANET) performance is affected by a number of factors, e.g. speed, node density. The effect of constrained mobility on the performance of the MANET protocol is focused . Experiments were carried out to measure the performance of MANET protocols in two mobility patterns: random waypoint and Metropolitan Grid (M-Grid). The latter reflects the mobility pattern that would be encountered in inter-vehicle communication system. The results show that MANET performance in the M-Grid mobility pattern is reduced considerable as compared to random waypoint. A parameter called coverage density index was proposed to normalize mobility patterns coverage and thus its connectivity. The probability of connectivity and route length provides an indication of the performance of MANET protocols given the mobility pattern.

2.5. EVALUATION OF GEOCAST ROUTING PROTOCOLS

A mobile ad hoc network (MANET) is a network consisting of a set of mobile nodes capable of communicating with each other without the assistance of base stations. The goal of a geocasting protocol is to deliver data packets to a group of nodes that are located within a specified geographical area, i.e., the geocast region. Evaluation of the performance of three geocast routing protocols for a MANET is performed via simulation: Location-Based Multicast, GeoGRID, and Geocast Adaptive Mesh Environment for Routing.

2.6. FACILITATING MATCH-MAKING SERVICE IN AD HOC AND SENSOR NETWORKS USING PSEUDO QUORUM

In a match-making system, sources (producers) advertise generated data without any particular destination in mind. Destinations (consumers) are determined based on their interests (via subscriptions) in receiving the produced data. Advertisements and subscriptions are *matched* by the underlying network service. They proposed to facilitate match-making capability in ad hoc and sensor networks by adapting the idea of quorum system. A quorum system is formed by organizing nodes into subsets called quorums, where every two quorums intersect and no quorum includes another quorum. To accommodate node mobility and network scale, we propose that producers and consumers systematically forward their advertisement and subscription messages to form 'pseudo' quorums, where they are matched at intersecting nodes. Simulation results shows that pseudo quorum based matchmaking system achieves a very high matching rate with much less messaging overhead as compared to that of event and query flooding.

2.7. PROCESSING RANGE-MONITORING QUERIES ON HETEROGENEOUS MOBILE OBJECTS

The various ways of leveraging heterogeneous mobile computing capability for efficient processing of real-time range-monitoring queries is considered. In our environment, each mobile object is associated with a resident domain and when an object moves, it monitors its spatial relationship with its resident domain and the monitoring areas inside it. An object reports its location to server whenever its movement affects any query results (i.e., crossing any query boundaries) or it moves out of its resident domain. In the first case, the server updates the affected query results accordingly while in the second case, the server determines a new resident domain for the object. This distributive approach is able to provide accurate query results and real-time monitoring updates with minimal location update and server processing costs. In addition, the new scheme allows a mobile object to negotiate a resident domain based on its computing capability. Thus, a more capable object can have a larger resident domain reducing its chance of having to request a new resident domain because of moving out of it. This feature makes the new approach highly adaptive to the heterogeneity of mobile objects. In our performance study, we compare it with an existing approach using simulation. The study shows that the new technique is many times better in reducing mobile communication and server processing costs.

CHAPTER 3

METHODOLOGIES

A large number of research efforts in various aspects of MANET have been made in recent years, ranging from flooding and routing , data lookup and delivery , and service discovery , just to name a few. However, there are no prior works proposed to solve the problem of sharing location dependent experiences in MANET. Efficient sharing of location-dependent experiences among mobile hosts in MANET has not been investigated in the literature and is the subject of the investigation in this projectwork.

3.1 Centralized Approach with single stationary server

A simple solution is a centralized approach that lets each mobile host report its location-dependent experience to a central server at a fixed location via multi-hop relays. The server keeps past experiences of all the mobile hosts. Before moving to a new location, a mobile host asks the server for other hosts' experiences in the close vicinity of the new location. The centralized approach is not scalable since it requires frequent interactions between the server and each mobile host, especially when a large number of mobile hosts create many experiences. Furthermore, the mobile hosts exhaust their battery power quickly as they need to frequently relay experiences to/from the central server. Last, the server is a single point of failure.

3.2 Centralized Approach with Four stationary servers

To address these problems, an adhoc network of a number of stationary servers can be used. These servers have a long transmission range to communicate among themselves and are more powerful than a regular mobile host. Each mobile host reports its experiences to its nearest stationary server. These stationary servers communicate among themselves to relay relevant experiences to a querying host. This approach, however, requires additional servers, which may not be feasible or too costly for some applications.

3.3 Distributed Approach

A simple distributed approach is to treat a location dependent experience as general data. Each host maintains its own experience. Before moving to a new location, a host broadcasts a request with the intended location information to its neighbors that relay the query to all the hosts through flooding. Only the hosts who have created experiences in the desired vicinity reply to the querying host. This approach is very expensive since a network-wide broadcast is needed every time a host moves. In this paper, we propose a new, efficient distributed solution, taking advantage of location information and the following ideas to reduce communication overhead and provide fast response time.

First, a host's experience is separated into an experience summary and a data item. The summary provides information such as the location and the time the experience is observed. The experience summary is typically very small. The corresponding data item can be large as it may be a video recording of an event. This separation enables us to employ different replication policies for data items and experience summaries. We store only one copy of a data item in the aggregated space of all mobile hosts to effectively utilize the storage space. We keep one or more copies of an experience summary in different mobile hosts to provide a fast response time.

Second and more importantly, the experience summary and the corresponding data item are kept in a mobile host close to the location where the experience was recorded. This host can provide any other host that wants to move into that location the relevant experiences quickly and efficiently. This strategy takes into account of the different replication strategies for the summary and the data item.

CHAPTER 4

PROPOSED TECHNIQUE

4.1 PROBLEM DEFINITION

MANET consists of a number of mobile hosts that communicate with each other through wireless packet relaying. MANET is excellent for sharing information in areas lacking of communication infrastructures such as disaster areas, combat zones, or unexplored territories (e.g., deep space or deep sea). In these scenarios, mobile hosts scatter around to observe various parts of a large unfamiliar territory and record their experience in their local storage. Before moving to any new location, a mobile host queries for experiences observed by other mobile hosts in the close vicinity of the new location. The user of the mobile host reviews these experiences to determine her course of actions, depending on a specific application. For example, for space exploration, an astronaut may decide to move forward to examine the new location that has not been explored by others.

4.2 LODE PROTOCOL:

The proposed technique for sharing location dependent experiences is called *LODE*, an acronym for LOcation Dependent Experience Sharing. An experience is location and observer dependent data that may be in the form of text, images, audio, or videos. Each experience is associated with a recording time, location, and information about the host that creates the experience. It is assumed that the boundary of the intended exploration area is known in advance and the mobile hosts only move within the area. Prior to an exploration, each host is configured with the width and height of the exploration area and the virtual grid size smaller than the exploration area. Hence, a host sees the exploration area as a number of virtual grids. Each mobile host is able to determine its own position using some positioning system such as GPS. Each host has a memory and a local disk. The host's memory is small but enough to run our protocol and keep one data item. The local disk is much larger and can store a number of experiences (summaries and data items). All hosts have the same communication bandwidth, transmission range, and memory and disk capacity.

4.2.1 CORE IDEAS OF LODE

LODE uses an underlying broadcast protocol provided by the network layer to broadcast a message within a number of hops limited by the time-to-live (TTL) of the message. LODE aims to provide low query response time and low communication overhead.

The core ideas of LODE are as follows. Since a data item can be large, only one copy of a data item is kept in the aggregated storage space of all the mobile hosts. One or more copies of an experience summary are maintained. All the mobile hosts that are currently in or about to move into a virtual grid are allowed to store the summaries of all experiences that have been created in the grid. The last host that would leave a virtual grid empty is required to keep the summaries of this grid with it.

Since a summary query specifies a query rectangle, any mobile hosts in the virtual grid enclosing or intersecting the query rectangle reply to the querying host. If the querying host does not receive a reply within a pre-defined timeout period because either no experiences were created in the query rectangle or the network is partitioned, the host records EMPTY as the summary corresponding to the query rectangle.

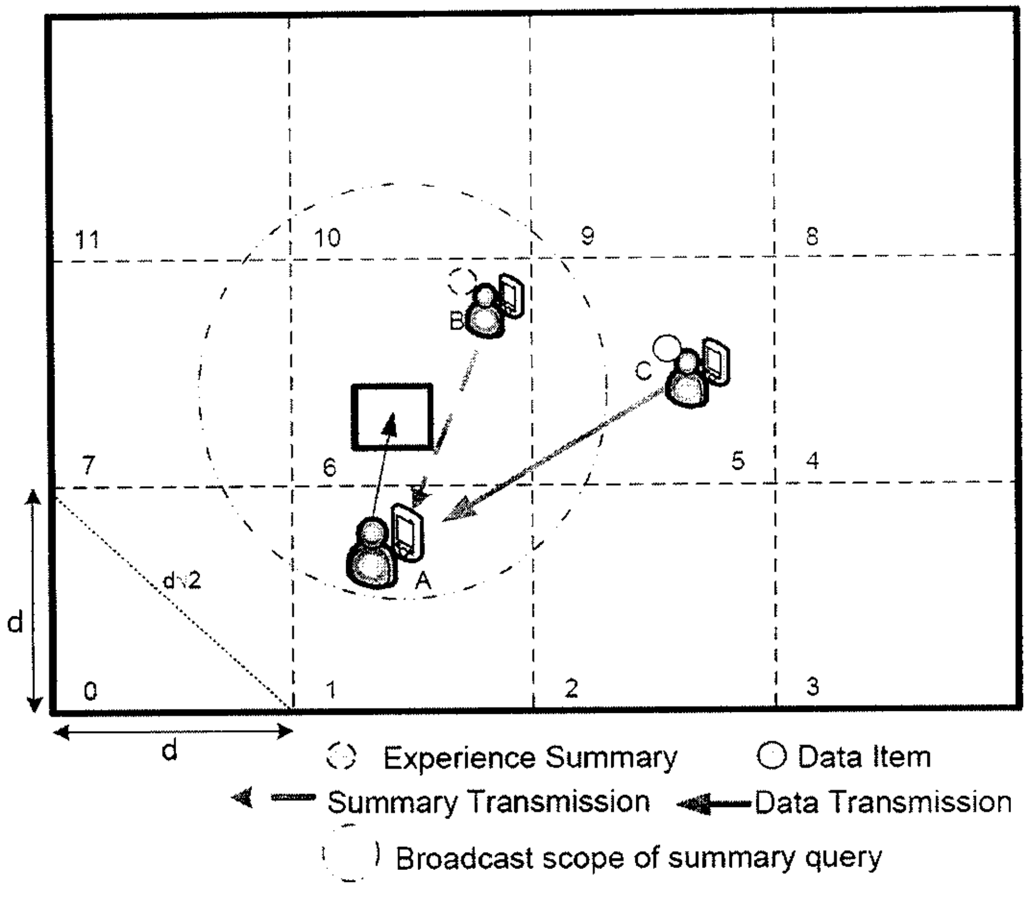


Figure4.1 LODE Using Virtual Grids

The above Figure illustrates these ideas. Suppose that the user of host A wants to move to a new location. The user issues a summary query with the query rectangle indicating the vicinity of the new location (i.e., the rectangle in Grid No. 6 in Figure). Note that the user is unaware of the virtual grids, but the application running on the mobile host of the user is. Host A knows that the query rectangle is not in its current virtual grid. The host broadcasts a summary query for the user's specified query rectangle with a limited TTL and waits for a reply for a predefined timeout period. Host B currently in the virtual grid of the query rectangle (Grid No. 6) sends the summaries of all experiences in the query rectangle to host A. Host B has the summaries since we make all hosts in a virtual grid maintains all the summaries of the experiences created in the virtual grid they are currently in.

Next, the user of host A chooses to view one or more data items corresponding to the received summaries. Host A broadcasts a data query in a limited scope to get the desired data item. Note that the location of the owner host of the experience (the one storing the data item) is not part of the summary since hosts are mobile and we do not want to keep updating the summary each time the owner host moves. Host A waits for a reply for a pre-defined timeout period. Host C, the owner host replies to the querying host, host A.

If host A is closer to the virtual grid of the query rectangle than host C, host A stores the received data item on its disk and host C deletes the data item from its disk. Host A lets other hosts in the destination virtual grid knows that host A is the new owner of the data item. At most one host replies for a given data query since we ensure that only one host stores the data item in the entire network to effectively utilize the aggregate storage space. The querying host may not receive any reply because of network partitioning. In this case, after the timeout period, the host records EMPTY DATA.

Note that prior to host A issuing the query, host C has moved out of Grid No. 6 after it created the experience in that grid. Before leaving the grid, host A found that at least one host (host B in the example) was in the grid and can maintain the summaries of all experiences in the grid. Host C then deleted all the summaries of Grid No. 6 from its disk but it still keeps the data item it created in that grid. If host A actually moves into Grid No. 6, this host broadcasts a summary query for the entire Grid No. 6 and stores the returned summaries on its local disk. Subsequent query rectangles in this grid issued by this user can be answered using the on-disk summaries.

With summaries and data items within the limited broadcast scope about the size of the virtual grid in most cases, LODE can shorten the response time and reduce communication overhead.

4.2.2 PRELIMINARIES

It is assumed that the boundary of the intended exploration area is known in advance and the mobile hosts only move within the area. Prior to an exploration, each host is configured with the width and height of the exploration area and the virtual grid size smaller than the exploration area. Hence, a host sees the exploration area as a number of virtual grids. Each mobile host is able to determine its own position using some positioning system such as GPS. Each host has a memory and a local disk. The host's memory is small but enough to run our protocol and keep one data item. The local disk is much larger and can store a number of experiences (summaries and data items). All hosts have the same communication bandwidth, transmission range, and memory and disk capacity.

We separate an experience into two parts:

1. A data item and
2. An experience summary.

An **experience summary** has

1. A location where the experience takes place;
2. Time when the experience is recorded;
3. A unique experience ID;
4. Owner ID; and
5. An optional brief description of the data item.

A **data item** can be text, images, or videos about an interesting event occurring at a location. Data items need not be of the same size.

The experience summary and the corresponding data item of an experience have the same experience ID. The experience ID can be generated from the IP or MAC address of the host and the time the experience is recorded. The owner ID uniquely indicates the host that stores the data item. Mobile hosts can issue two types of queries:

1. Summary query
2. Data query

Summary query consists of

a) The top-left and bottom right coordinates of the query rectangle indicating the vicinity of the host's next intended destination and

b) The amount of available disk space on the querying host. Note that shapes other than rectangles can also be used for a summary query. We chose a rectangle for ease of explanation. The available disk space is used by the host receiving the summary query to determine whether to send additional information.

Data query specifies the unique experience ID of the data item to be retrieved. The ID is obtained from the summary returned as a result of the summary query.

4.2.3 MODULES

At any one time, each host is located in one virtual grid of size dxd meters2. Let tr be the transmission range (in meters) of a host. Let DH be the minimum number of hops covering the diagonal distance of a virtual grid. Each mobile host maintains a FullFlag variable initialized to false. The host sets this variable to true when it is not able to store a summary broadcast by other hosts in its current grid since its local disk is full. The host performs one of the following actions.

1. Create a new experience and save the Experience
2. Receive a new experience summary
3. Enter a neighboring grid
4. Leave the current grid
5. Process a user query

6. Receive a summary Query
7. Receive a data query

1. Create a new experience and save the Experience:

After creating an experience in memory, the user tells her mobile host to save an experience on disk.

```
Save Experience ()
If available space is enough for the summary and the data item
    Store both the data item and the summary on disk
    Broadcast the summary with the message TTL set to DH
Else Discard the experience
```

With this TTL setting, all hosts in the broadcasting host's grid receive the summary. Each of these hosts stores a copy of the summary in its local disk if its disk space is available in order to answer summary queries from other hosts that wish to enter this grid. We do not broadcast the data item since it may be large. If the broadcasting host is near the grid border, the summary will reach hosts in a neighboring grid as well, but these hosts will not store the summary outside of their current grid.

2. Receive a new experience summary:

When a host receives a new summary, it executes the following.

```
Receive New Summary ()
If this host is in the same grid as the host originating the broadcast of
the summary
    If enough disk space is available to store the new summary
        Store the summary on disk
    Else
        Set FullFlag to true
        Discard all summaries of the host's current grid
    End If
Else Discard the summary
```

Only a host within the broadcast scope of the experience summary and in the same grid as the host originating the summary broadcast will try to keep the summary. When the host cannot store all summaries of its current grid, it sets FullFlag to true and deletes all on-disk summaries of the current grid. This is to utilize its available disk space for storing its own future experiences.

3. Enter a neighboring grid:

When the user decides to move to her next location, the mobile host checks whether the new location is in a neighboring grid. If so , the mobile host executes the following code that lets the entering host keep summaries of all experiences recorded in the destination grid if the host has enough available disk space.

```
Enter Neighbor Grid ()
1) GTTL = 2 * DH
2) Repeat
        2. a) Broadcast a summary query with the message TTL set to GTTL, the query rectangle
set to the grid, and the available space set to the current available disk space
        2. b) If not getting a query response within a timeout period
                GTTL = GTTL + DH
             End If
  Until getting a response or GTTL exceeding a TTL threshold
3) If not getting a response
        3. a) Set the summary for this grid to EMPTY
4) Else
  If the response is negative
  /* this host's available disk is too small for the summaries */
        4. a) Set FullFlag to true
5) Else
  /* the response is positive */
        5. a) Set FullFlag to false
        5. b) Store the summaries in the response on disk
        5. c) If the responding host (sender) is not in the destination grid of this host
                5. C.1) Receive and store summaries of additional grids on disk.
```

```
            5. C.2) Ask the sender to delete from its disk all the summaries it sent to this
            host
        End If
End If
```

Only one broadcast is needed if there exists a mobile host in the destination grid. Otherwise, more broadcasts (limited by the TTL threshold) are issued to find a host (in a neighboring grid of the destination grid) that has all summaries of experiences of the destination grid. Since several hosts in the destination grid may be responding, the querying host picks the one that responses first and ignores subsequent responses.

When the chosen responding host (sender) is not in the destination grid, the querying host asks the sender to delete the summaries of the destination grid if the querying host has enough space. This is because the querying host will soon be in the destination grid and will answer summary queries related to the destination grid instead. Depending on the amount of the available disk space indicated in the summary query, the sending host may send additional summaries. These summaries are from experiences created in the grids closer to the destination grid than the sending host's current grid. The sender deletes these summaries from its disk only if the querying host can store these summaries .The idea is to keep the summaries near their original locations.

4. Leave the current grid:

When a host wishes to leave its current grid, it decides whether to carry summaries related to this grid with it or not. The host will not carry the summaries of its current grid with it if there exists some other host in its current grid. The purpose is to try to keep experience summaries near the locations they were created.

```
Leave Current Grid ()
1) Broadcast a LEAVE message with the message TTL set to DH
2) If this host gets a response within a timeout period
        /* some other host in this grid has the same summaries */
        Delete all summaries related to its current grid
    End If
```

Any host receiving the LEAVE message executes the following code.

```
Receive LEAV E Message ()
If the host is in the same grid as the sender and its FullFlag is false
        Reply to the sender
End If
```

Note that hosts in a different grid do not reply and hosts that are in the same grid, but do not have all the summaries of this grid (i.e., FullFlag is true) do not reply.

5. Process a user query:

Before the user moves to a new location, the user sets the desired query rectangle indicating the vicinity of the new location and the mobile host performs the following.

```
Issue Summary Query ()

1) If the query rectangle is entirely in the current grid and FullFlag is false
        1.a) Return only relevant summaries using the on-disk summaries
        of the current grid
2) Else
   If the query rectangle is entirely in a neighboring grid of the current grid
   /* and FullFlag is true */
        2.a) GTTL = 2 ▯ DH
        2.b) Broadcast a summary query with the message TTL set to GTTL, the query rectangle
set to the user's query rectangle, and the available disk space set to negative
        2.c) If receiving a response within a timeout period
                Return the user the summaries in the response
        End If
3) Else
        3.a) Find all candidate grids that intersect the query rectangle
        3.b) Let MAXD be the largest distance between the host and the farthest corner of each
    of the candidate grids
        3.c) Broadcast a summary query with the message TTL set
        3.d) If receiving a response within a timeout period
                Return all the summaries in the user's query rectangle
            Else
                Return EMPTY
            End If
    End If
```

If the query rectangle is entirely within the current grid, the summary query can be answered right away since the host stored all summaries of the grid already on its disk if FullFlag is false when it entered the grid. Otherwise, the host broadcasts a summary query to grids enclosing or intersecting the user's query rectangle. Based on the receiving summaries, the user selects an experience to review. The mobile host issues a data query for the user's selected experience as follows.

```
    Request Data()
    1) Set GTTL to the number of hops covering the diagonal distance between the querying host and
the farthest corner of the exploration area.
    2) Broadcast a data query for the data item ID with the TTL set to GTTL
    3) If receiving the data item within a timeout period
            3.1) If the host has enough disk space to store the requested data item and the
        sender's grid is not in the same grid as the query rectangle
            /* move the data item closer to its original location */
                3.1.a) Store the data item on its disk.
                3.1.b) Ask the sender to delete the data item from its local disk.
                3.1.c) Broadcast the updated summary with the message TTL set to DH
                to announce that this host is the new owner of the data item.
            End If
    Else
            Return EMPTY DATA
```

The data owner sends the data item to the querying host using some existing routing protocol such as AODV. If the querying host has enough space to store the data item, the querying host asks the current data owner to delete the item from its local disk. The querying host becomes the new owner of this data item. The idea is to maintain a data item in a mobile host closer to the location that the data item was created. A querying host issues one data query to request one data item. The overhead can be reduced further by packing several data item IDs in one data query.

6. Receive a summary query:

The host receiving the summary query executes the following code. The term destination grid refers to the grid of the query rectangle.

```
Receive Summary Query ()
1) If the host does not have summaries related to the query rectangle or FullFlag is true
        Discard the summary query
Else
2) If this host is in the same grid as the query rectangle
        Send the querying host all summaries of experiences in the query rectangle
3) Else
        If this host and the query rectangle are in two different grids
        3.a) Mark a response as negative if the querying host's available space is smaller
than the total size of the experiences in the destination grid
        3.b) Put all summaries of experiences in the destination grid in the response if
the response is not marked negative
        3.c) Send the response to the querying host
        3.d) When asked by the querying host to delete the summaries of a particular
grid, this host deletes all on-disk summaries related to the specified grid
        3.e) Send all summaries related to each of the neighboring grids of the
        destination grid if the distance between the neighboring grid of the destination
grid is closer to the destination grid than this host's current grid and the querying
host has enough available disk space to store all these summaries
End If
```

The rationale behind Step 3) is to keep the summaries of experiences near the location where the experiences were created.

7.Receive a data query:

When receiving a data query, the host executes the following code.

```
Receive Data Query ()
1) If the host has the requested data item
        Send the requested data item
        If this host is not in the same grid as the querying host
                Delete this data item when asked by the querying host
        End If
   End If
```

The host with the data item deletes its data item only when the querying host can store the data item.

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Simulation Environment:

The simulations were implemented on NS2 (Network Simulator), a Discrete Event Simulator modeled for wireless network systems. The Network Simulator components consist of Network Simulation Engine to execute tcl scripts containing simulation setup and events and NAM (Network Animator) to visualize the output.

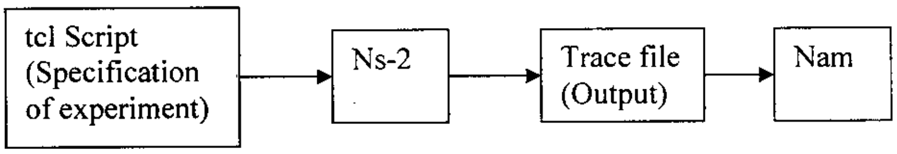


Figure 5.1: NS2 Components

5.2 Simulation Parameters:

Parameter	**Default value**	**Unit**
Exploration area	160 X160	meter^2
Virtual grid area	10 X10	meter^2
M-GRID	2 X2	meter^2
Host density	5	Neighbors/host
Event density	0.5	Event/meter^2
Transmission radius	$10\sqrt{2}$	meter
Query rectangle	1 X 1	meter
Host's moving speed	0-10	Meter/s
Avg size of a data item	5	MB

The performance of LODE is compared with those of 1 stationary server (1-SS) and 4stationary servers (4-SS). Each trace file records the host locations and summary queries issued by these hosts. The three techniques are compared using the following performance metrics.

Summary query latency:

Summary query latency measured as smallest number of hops needed to reach the first host having the answer for a summary query. The summary query indicates the area (query rectangle) where the user wants to know any prior experience.

Data query latency:

Data query latency measured as smallest number of hops needed to reach the first host having the requested data item for a data query.

Experience success rate:

Experience success rate measured as a fraction of the total number of users' queries that are satisfied. A user's query is satisfied if the users can successfully retrieve the data items of all experiences in a query rectangle. The user may not be able to get some experience either because the summary is not found or the data item is not found due to network partitioning.

Drop rate:

Drop rate measured as a fraction of times a new experience cannot be stored in a mobile host's disk due to limited disk space at the host. For the 1-SSand 4-SS schemes, the drop rate is always zero since stationary servers have enough disk space to keep all the experiences.

Effect of Host Density

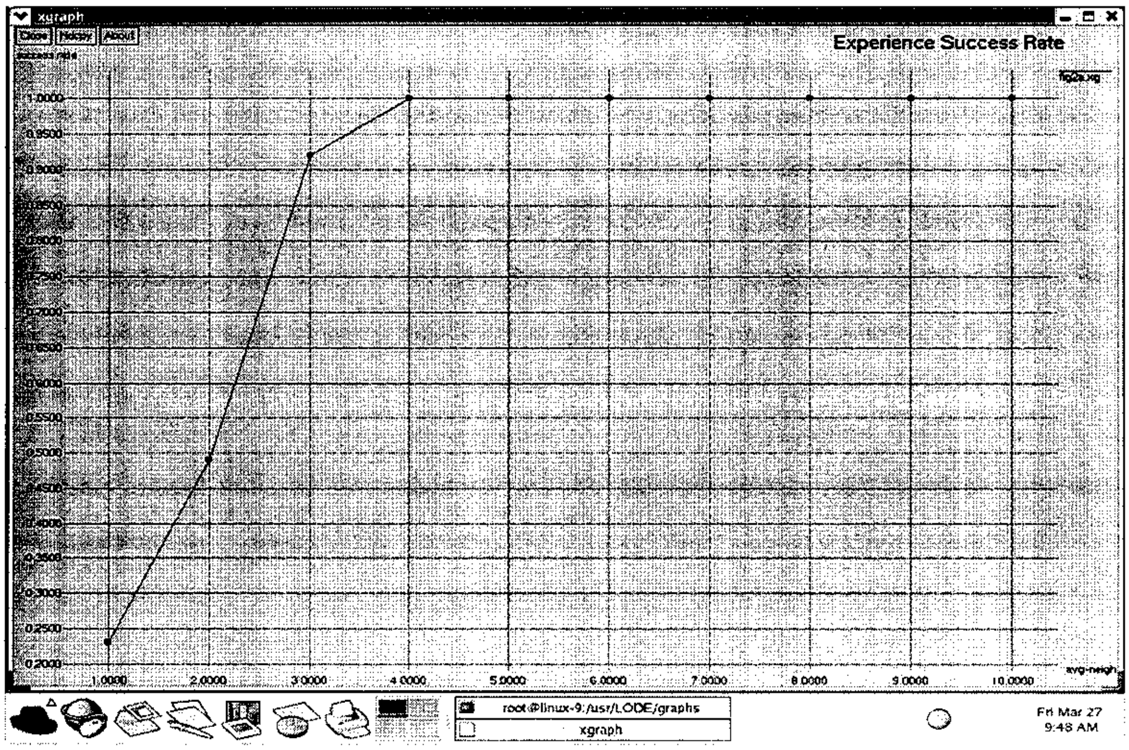
The effect of various host densities on the success rate is investigated for retrieving experiences and average latencies to get responses for summary queries and data queries. The host density is varied between 1 and 10 average neighbors per host and the other parameters are fixed at their default value.

Effect of Event Density

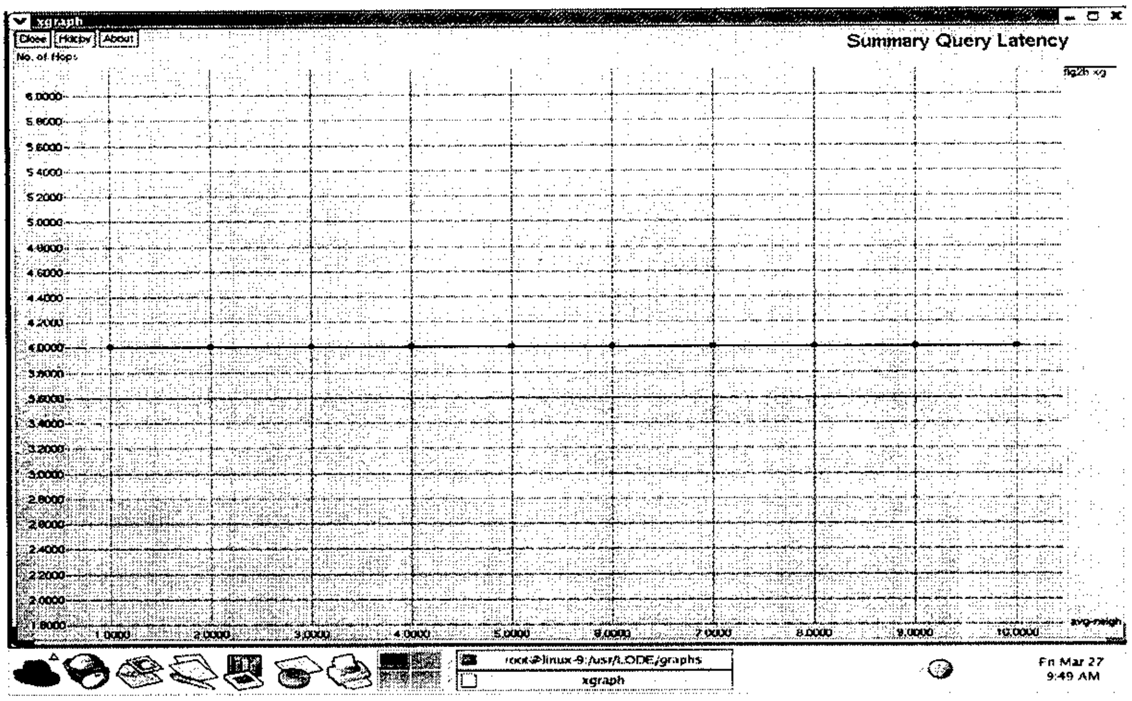
A number of events, ranging from 0.1 to1 event=meter2 are generated and other parameters are set at the default values.

5.3 Simulation Results:

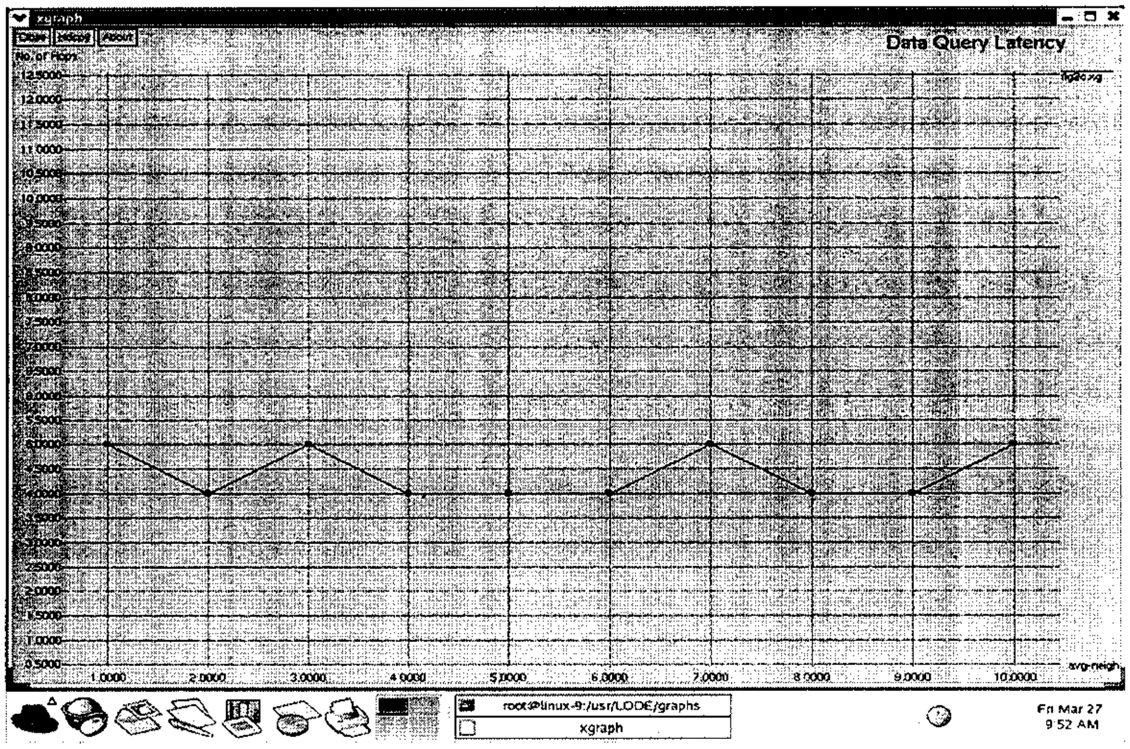
Experience success rate:



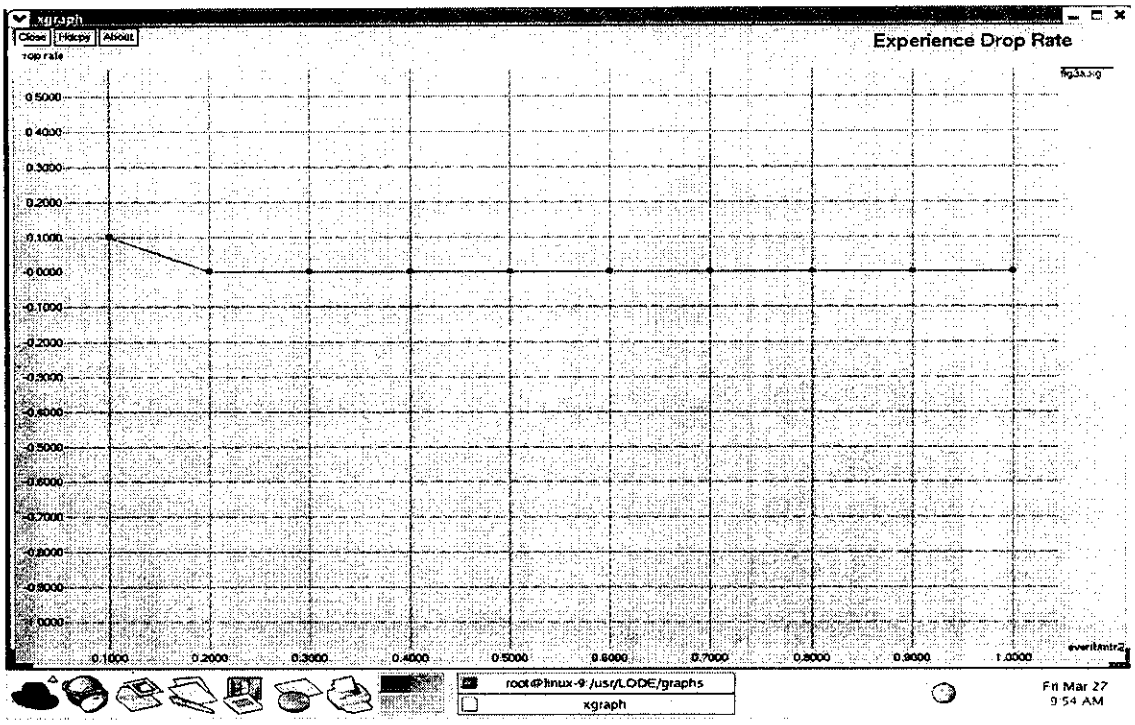
Summary query latency:



Data query latency:



Drop rate:


xgraph
Close
Hdcpy
About
Experience Drop Rate
rop rate
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
-0.1000
-0.2000
-0.3000
-0.4000
-0.5000
-0.6000
-0.7000
-0.8000
-0.9000
-1.0000
0.1000
0.2000
0.3000
0.4000
0.5000
0.6000
0.7000
0.8000
0.9000
1.0000
root@linux-9:/usr/LODE/graphs
xgraph
Fri Mar 27
9:54 AM

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENTS

This projectwork presented a new problem of sharing location dependent experiences for applications such as hurricane response missions, combat zones, or deep sea or deep space exploration. A new distributed solution called LODE is proposed and it is shown that it cuts the latency for accessing desired experiences by more than half compared with the technique in which all mobile hosts report to a single stationary server for any new experiences they encounter.

The future works include the detailed study of the impact of the virtual grid size, a further reduction of broadcast overhead for data items by predicting the location of a host after some time steps via analysis given a movement pattern. This will enable us to determine the appropriate TTL value for a data query. Evaluation of the performance of LODE can be done for other hosts' movement patterns such as the random way points model.

7. APPENDIX

7.1 Source code

Slp_ua.h:

```
#include "slp/slp-ua.h"

u_int16_t G_Xid = 0;
int rx_count = 0;
int rx_advertcount = 0;
RNG G_arrival;

// SLPua OTcl linkage class
static class SLPuaClass : public TclClass {
 public:
  SLPuaClass() : TclClass("Application/SLPua") {}
  TclObject* create(int, const char*const*) {
    return (new SLPua);
  }
} class_slp_ua;

void reReqTimer::expire(Event *)
{
  s_->send_srvrqst_pkt(r_, a_);
  if (m_==REG)
    resched(i_);
  else
    resched(G_arrival.exponential(i_));
}

void urlCacheTimer::expire(Event *)
{
  s_->remove_cached_url(n_);
}

void retxTimer::expire(Event *)
{
  if (n_->retxok || !n_->gotreply)
    h_->update_resched_resend(n_);
  else //no further responses caused by the PR_LIST
    h_->remove_from_table (n_->xid);
}

// Constructor (also initialize instances of timers)
SLPua::SLPua() : running_(0) , rqstTable_(this)
```

```
{
 bind("pktsize_",&pktsize_);
 bind("dstport_",&dstport_);
 bind_bool("maxSearch_",&maxSearch_);
 number_of_scopes_ = 0;
 url_head = NULL;
}

// OTcl command interpreter
int SLPua::command(int argc, const char*const* argv)
{
 Tcl& tcl = Tcl::instance();

 if (argc == 3)
 {
  if (strcmp(argv[1], "attach-agent") == 0) {
   agent_ = (Agent*) TclObject::lookup(argv[2]);
   if (agent_ == 0) {
    tcl.resultf("no such agent %s", argv[2]);
       return(TCL_ERROR);
   }
   // Make sure the underlying agent support MM
   if(agent_->supportSLP()) {
       agent_->enableSLP();
   }
   else {
       tcl.resultf("agent \"%s\" does not support SLP Application", argv[2]);
       return(TCL_ERROR);
   }
   agent_->attachApp(this);
   return(TCL_OK);
  }
  else if (strcmp(argv[1],"add-scope") == 0)
  {
   add_scope(argv[2]);
   return(TCL_OK);
  }
  else if (strcmp(argv[1],"remove-scope") == 0)
  {
   remove_scope(argv[2]);
   return(TCL_OK);
  }
 }
 else if (((argc==4)||(argc==6))&&(strcmp(argv[1],"rqst-service") == 0))
 {
  if (argc==6)
  {
   if (strcmp(argv[4],"-reg")==0)
    reReqTimer *resend=new reReqTimer(this, atof(argv[5]),
              (nsaddr_t)strtol(argv[3], (char **)NULL, 16), argv[2], REG);
   if (strcmp(argv[4],"-exp")==0)
    reReqTimer *resend=new reReqTimer(this, atof(argv[5]),
              (nsaddr_t)strtol(argv[3], (char **)NULL, 16), argv[2], EXP);

  }
  send_srvrqst_pkt(argv[2],(nsaddr_t)strtol(argv[3], (char **)NULL, 16));

  return(TCL_OK);
 }
 return (Application::command(argc, argv));
}

void SLPua::init()
{
 //add_scope("DEFAULT");
 xid_seed();

}

void SLPua::add_scope(const char* newscope) {
 char *scope = small_string(newscope);
 for (int i=0; i<number_of_scopes_; i++)
 {
  if (strcmp(local_scopes_[i],scope)==0)
  { // already in list
   return;
  }
 }
 if (number_of_scopes_==SLP_MAX_SCOPE)
 {
  printf("Can't add anymore scopes, MAX=%d scopes per SA\n",SLP_MAX_SCOPE);
  return;
 }
 local_scopes_[number_of_scopes_]=scope;
#ifdef DEBUGUA
 printf("UA:Added %s, total=%d Scope \n",local_scopes_[number_of_scopes_],
number_of_scopes_+1);
#endif
 number_of_scopes_++;

}
```

```
void SLPua::remove_scope(const char *removescope) {
 int i,j;
 char *scope = small_string(removescope);
 for (i=0; i<number_of_scopes_; i++)
 {
  if (strcmp(local_scopes_[i],scope)==0)
    break;
 }
 number_of_scopes_--;
 for (j=i; j<number_of_scopes_; j++)
 {
  local_scopes_[j]=strdup(local_scopes_[j+1]);
 }
}

void SLPua::cache_url(u_int16_t url_entry_count, const char* url) {

 urlNode *c;
 hdr_slp_url *addurl;
 int index=0;
 for (int i=0; i<url_entry_count; i++)
 {
       addurl=(hdr_slp_url *)(url + index);
       if (addurl->lifetime != 0)
       {
        #ifdef DEBUGUA
        printf("%f: Expiring after %f\n",CURRENT_TIME(),(double)(addurl-
>lifetime)-CURRENT_TIME());
      #endif
        c = new urlNode(this, (double)(addurl->lifetime));
        (c->data).lifetime=addurl->lifetime;
        (c->data).url=strdup(addurl->url);
        (c->data).url_length=addurl->url_length;
        (c->data).auth_no=0;
        c->next = url_head;
        url_head = c;

        #ifdef DEBUGUA
        printf("UA:Cached %s %d\n",(url_head->data).url, (url_head->data).auth_no);
        #endif
     }
     index+=sizeof(hdr_slp_url);
 }
}

static bool match(const char *url, const char *service)
```

```
{
 int position;
 char *token;
 char *temp1;
 char *temp2;
 temp1=strdup(url);
 token=strstr(temp1,"://");
 position = token -temp1;
 temp2=strndup(temp1,position);
 delete temp1;

 if (strstr(temp2,service) != NULL)
 {
  delete temp2;
  return true;
 }
 else
 {
  delete temp2;
  return false;
 }
}

bool SLPua::search_cached_urls(const char* service)
{
 bool matched=false;
 urlNode *c = url_head;
 int offset;
 Tcl& tcl = Tcl::instance();
 char wrk[5000];

 while (c!=NULL)
 {
  if (match((c->data).url,service)==true)
  {
   matched=true;
   tcl.evalf("puts $tracefd \"C n%d: %f****Cached service urls matches %s\"",agent_-
>addr(),CURRENT_TIME(),(c->data).url);
   return matched;
  }
  c=c->next;
 }
 return matched;
}

void SLPua::remove_cached_url(urlNode *removeurl) {
  urlNode *p=NULL, *c=url_head;
  while ((c!=NULL) && (c != removeurl)) {
   p=c; c=c->next;
  }
  if (c==NULL)
   return;
  if (p==NULL)
   url_head = c->next;
  else
   p->next = c->next;
  delete c;

  //we assume that a cached service doesn't timeout before MC_CONFIG_MAX, so we
do
  //not care to remove it from the prlist if the request retransmission are still active
}

void SLPua::start()
{
  init();
  running_ = 1;
}

void SLPua::stop()
{
  running_ = 0;
}

char *  SLPua::get_local_scopes() const
{
  char buf[21];
  char str[SLP_MAX_SCOPE*20]="";
  for (int i=0; i<number_of_scopes_; i++)
  {
   if (i==0) {
    sprintf(buf,"%s",local_scopes_[i]);
    strcat(str,buf);
   }
   else {
    sprintf(buf,",%s",local_scopes_[i]);
    strcat(str,buf);
   }
  }
  return strdup(str);
}
// Send application data packet
```

```
void SLPua::send_srvrqst_pkt(const char* service_type, nsaddr_t destaddr)
{
 //first check if request service is cached
 if (search_cached_urls(service_type)==true)
   return;

 #ifdef DEBUG
   printf("No cached match...sending request\n");
 #endif

 //Otherwise, send a new service request
 ns_addr_t dest;
 dest.addr_=destaddr;
 dest.port_=dstport_;
 char * scopes = get_local_scopes();
 //printf("Dest is %d:%d\n",dest.addr_,dest.port_);
 hdr_slp_srvrqst srvrqst_buf;
 if (running_) {
   // the below info is passed to UDPslp agent, which will write it
   // to SLP SRVRQST header after packet creation.
   srvrqst_buf.version = SLP_VERSION;
   srvrqst_buf.function_id = SLPTYPE_SRVRQST;
   srvrqst_buf.flags = SLP_REQUEST_MCAST;
   srvrqst_buf.next_ext_offset=0;
   srvrqst_buf.xid = xid_generate();
   srvrqst_buf.language_tag_length=strlen(SLP_LANGUAGE);
   srvrqst_buf.language_tag=SLP_LANGUAGE;
   srvrqst_buf.length_prlist=0;
   srvrqst_buf.length_service_type=strlen(service_type);
   srvrqst_buf.string_service_type=strdup(service_type);
   srvrqst_buf.length_scope_list=strlen(scopes);
   srvrqst_buf.string_scope_list=scopes;
   srvrqst_buf.length_predicate=0;
   srvrqst_buf.length_spi=0;
   srvrqst_buf.length=srvrqst_buf.size();
#ifdef DEBUG
  printf("n%d: UA Sending Service Request (xid=%d) to %x:%d\n",agent_->addr(),
srvrqst_buf.xid,dest.addr_,dest.port_);
#endif
   rqstTable_.add_resched_send(srvrqst_buf.xid, srvrqst_buf.length, (char*)
&srvrqst_buf, dest);
  }
}

// Receive message from underlying agent
void SLPua::recv_slpmsg(ns_addr_t src, int nbytes, const char *msg)
```

```
{
#ifdef DEBUGUA
 printf("n%d: UA receiving message from %x:%d\n",agent_->addr(),src.addr_,
src.port_);
#endif

 if(_ sg) {
   hdr_slp* slp_buf = (hdr_slp*) _ sg;

   if (slp_buf->function_id == SLPTYPE_SRVRPLY) {
    // If received packet is a request packet
    hdr_slp_srvrply* slp_srvrply_buf = (hdr_slp_srvrply*) _ sg;
    //print service reply here
    #ifdef DEBUG
    printReply(slp_srvrply_buf);
    #endif
    cache_url(slp_srvrply_buf->url_entry_count,slp_srvrply_buf->url);
    if (_ axSearch_)
      rqstTable_.add_pr(slp_srvrply_buf->xid, src.addr_);
    else
      rqstTable_.cancel_retx(slp_srvrply_buf->xid);
    #ifdef DEBUGUA
    printf("n%d:String is %s\n",agent_->addr(),rqstTable_.get_pr(slp_srvrply_buf->xid));
    #endif
   }
   else if (slp_buf->function_id == SLPTYPE_SAADVERT) {
    hdr_slp_saadvert* slp_saadvert_buf = (hdr_slp_saadvert*) _ sg;
    printf("n%d: ****Received a service advert xid=%d scopes=%s url=%s (adv
count=%d)\n",
        agent_->addr(), slp_saadvert_buf->xid, slp_saadvert_buf-
>string_scope_list,slp_saadvert_buf->url,++rx_advertcount);
    if (_ axSearch_)
      rqstTable_.add_pr(slp_saadvert_buf->xid, src.addr_);
    else
      rqstTable_.cancel_retx(slp_saadvert_buf->xid);

    //configure SA with scopes fro_  the SAAdvert
    discover_scopes(slp_saadvert_buf->string_scope_list,  slp_saadvert_buf-
>length_scope_list);
   }
  }

}
void SLPua::discover_scopes(const char* scope_list, u_int16_t length_scope)
{
 if (length_scope==0)
```

```
  return;

 char *delimiters = ",";
 char *token;
 char scope[length_scope];
 strcpy(scope,scope_list);
 token=strtok(scope, delimiters);
 while (token!=NULL)
 {
  add_scope(token);
  token=strtok(NULL, delimiters);
 }
}

void SLPua::printReply (hdr_slp_srvrply* rply)
{
  printf("n%d: ****Received a service reply xid=%d url=%d (count=%d)\n",agent_-
>addr(), rply->xid,
         rply->url_entry_count,++rx_count);
  int index=0;
  for (int i=0; i<rply->url_entry_count; i++)
  {
         printf("%s*",((hdr_slp_url *)((rply->url) + index))->url);
         index+=sizeof(hdr_slp_url);
  }
  printf("\n");

}


void SLPua::xid_seed(void)
{
 G_Xid=0;
 //G_Xid = (u_int16_t)Random::uniform(65535);
}

u_int16_t SLPua::xid_generate(void)
{
  G_Xid++;
  return G_Xid;
  //return ((u_int16_t)Random::uniform(65535));
}

rqstHashTable::rqstHashTable(SLPua *s) {
 s_ = s;
 tablesize = SLP_MAX_OUTSTANDING*3;
 table = new rqstNode* [tablesize];
 for (int i=0; i<tablesize; i++) {
  table[i] = NULL;
 }
}

int rqstHashTable::hash (u_int16_t xid) const
{
 return (xid%tablesize);
}
void rqstHashTable::add_pr(u_int16_t xid, int newaddress)
{
 rqstNode *c;
 int key=hash(xid);
 for (c=table[key]; c!=NULL; c=c->next) {
  if (c->xid == xid) {
   add_local_pr(c, newaddress);
   return;
  }
 }
}

void rqstHashTable::cancel_retx(u_int16_t xid)
{
 rqstNode *c;
 int key=hash(xid);
 for (c=table[key]; c!=NULL; c=c->next) {
  if (c->xid == xid) {
   //cancel retransmissions
   c->retx_timer_.cancel();
   //remove node from hash table
   remove_from_table (xid);
   return;
  }
 }
}

void rqstHashTable::add_local_pr(rqstNode* c, int newaddress)
{
 for (int i=0; i<c->number_of_pr; i++)
 {
  if (c->local_pr[i]==newaddress)
  { // already in list
   return;
  }
 }
 if (c->number_of_pr==SLP_MAX_PRLIST)
 {
   printf("Can't add anymore prev. responders, MAX=%d pr per
SA\n",SLP_MAX_PRLIST);
   return;
 }
 c->local_pr[c->number_of_pr]=newaddress;
 c->number_of_pr++;
 c->retxok=true;
 c->gotreply=true;
}

char * rqstHashTable::get_pr(u_int16_t xid) const
{
  rqstNode *c;
  int key = hash(xid);
  for (c=table[key]; c!=NULL; c=c->next) {
    if (c->xid == xid)
      return (get_local_pr(c));
  }
  return ("");
}

char * rqstHashTable::get_local_pr(const rqstNode* c) const
{
  char buf[21];
  char str[SLP_MAX_PRLIST*21]="";
  for (int i=0; i<c->number_of_pr; i++)
  {
    if (i==0) {
      sprintf(buf,"%d",c->local_pr[i]);
      strcat(str,buf);
    }
    else {
      sprintf(buf,",%d",c->local_pr[i]);
      strcat(str,buf);
    }
  }
  return strdup(str);
}

void rqstHashTable::add_resched_send(u_int16_t xid, int size, const char *mssg,
ns_addr_t dst)
{
  /* if size exceeds SLP pkt size, then do nothing */
  if (size > s_->pktsize_) {
```

```
 #ifdef DEBUGUA
 printf("Will not send request, size=%d, max=%d\n",size,s_->pktsize_);
 #endif
 return;
}

/*adding new node*/
rqstNode *c;
int key=hash(xid);
c = new rqstNode(this);
c->xid = xid;
c->number_of_pr = 0;
c->next = table[key];
memcpy(&c->msg,mssg, sizeof(hdr_slp_srvrqst));
c->dest=dst;
c->timeout=SLP_INITIAL_TIMEOUT;
c->expire=CURRENT_TIME()+SLP_CONFIG_MC_MAX;
table[key]=c;

/*rescheduling packet*/
c->retx_timer_.resched(c->timeout);
c->retxok=false;
c->gotreply=false;
/*sending packet*/
s_->agent_->sendto(size, c->msg, c->dest); // send to UDP

#ifdef DEBUGUA
printf("(xid=%d) Current Time=%f, next retx=%f, expire=%f\n",c->xid,
CURRENT_TIME(), c->timeout+CURRENT_TIME(),
  c->expire);
#endif

}

void rqstHashTable::update_resched_resend(rqstNode *n)
{
 //retrasnmit until CONFIG_MC_MAX is reached
 if (CURRENT_TIME() >= n->expire)
 {
   //remove this request history from hashtable
   remove_from_table (n->xid);
   return;
 }

 hdr_slp_srvrqst* srvrqst_buf = (hdr_slp_srvrqst*) n->msg;
```

```
//adding the prlist to srvrqst packet before its resent
char *pr=get_local_pr(n);
srvrqst_buf->length_prlist=strlen(pr);
srvrqst_buf->string_prlist=pr;
srvrqst_buf->length=srvrqst_buf->size();

n->retxok=false;

//retrasnmit until request will no longer fit in a single datagram
if (srvrqst_buf->size() > s_->pktsize_) {
  #ifdef DEBUGUA
  printf("Will not send request, size=%d, max=%d\n",srvrqst_buf->size(),s_->pktsize_);
  #endif
  return;
 }

 #ifdef DEBUGUA
 printf("R(xid=%d) Current Time=%f, next retx=%f\n",n->xid, CURRENT_TIME(), n-
>timeout+(CURRENT_TIME()*2.0));
 #endif

 /*Rescheduling*/
 n->retx_timer_.resched(n->timeout*=2.0);

 ///*Re-sending*/
 s_->agent_->sendto(srvrqst_buf->length,
                    n->msg,
                    n->dest);

}



void rqstHashTable::remove_from_table (u_int16_t xid)
{
 int key=hash(xid);
 rqstNode *p=NULL, *c=table[key];
 while ((c!=NULL) && (xid!=c->xid)) {
   p=c; c=c->next;
 }
 if (c==NULL) {
   printf("Error: Didn't find the node to be deleted from hash table\n");
   return;
 }

 //now adjust pointers
 if (p==NULL) //Node to be deleted is the first node
  table[key]  = c->next;
 else
  p->next = c->next;

 delete c;
}
```

Cache-common.tcl:

```
=# Common parameters
set val(a_sn)           [expr $val(sn)-$val(dup)]
set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)            Mac/802_11
set val(ifq)            Queue/DropTail/PriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(ifqlen)     50          ;# max packet in ifq
set val(adhocRouting)  BCAST
set val(SLPport)        18
set val(MulticastAddr)          0xE000000
set val(extra)          20.00           ;# wait for that amount after simulation ends
set val(motionFile)     "slp_tools/slpMotion/$val(runDirectory)/$val(scenario).motion"
set val(outputFile)     "slp_tools/slpTraces/$val(runDirectory)/cache/$val(scenario).tr"
set val(noutputFile)    "slp_tools/slpTraces/$val(runDirectory)/cache/$val(scenario).nam"
##########################################################################
######
# minimize memory requirements by removing all irrelevant packet headers
remove-packet-header PGM PGM_SPM PGM_NAK Pushback NV LDP MPLS
rtProtoLS Ping
remove-packet-header TFRC TFRC_ACK Diffusion RAP TORA IMEP MIP
remove-packet-header IPinIP Encap HttpInval MFTP SRMEXT SRM aSRM
remove-packet-header mcastCtrl CtrMcast rtProtoDV GAF Snoop TCPA TCP IVS
remove-packet-header Resv UMP Src_rt
# selectively allow only one of the following: AODV, SR, BCAST
remove-packet-header AODV
remove-packet-header SR
# remove-packet-header BCAST
##########################################################################
######
# Initialize Global Variables and Procedures
```

```
#Set up Random Number Generators

#seed the defaultRNG
$defaultRNG seed 9999
set arrivalRNG [new RNG]
set nnodeRNG [new RNG]
set serviceRNG [new RNG]
for {set j 1} {$j < $val(seed)} {incr j} {
 $arrivalRNG next-substream
 $nnodeRNG next-substream
 $serviceRNG next-substream
}
set arrival_ [new RandomVariable/Exponential]
$arrival_ set avg_ $val(exp_avg)
$arrival_ use-rng $arrivalRNG
set nnode_ [new RandomVariable/Uniform]
$nnode_ set min_ 0
$nnode_ set max_ [expr $val(nn)-1]
$nnode_ use-rng $nnodeRNG
set service_ [new RandomVariable/Uniform]
$service_ set min_ 0
if {$val(dup) == 0} {
  $service_ set max_ [expr $val(a_sn)-1]
} else {
  $service_ set max_ $val(a_sn)
}
$service_ use-rng $serviceRNG
# Initialize array of nodes to zero (no nodes used yet)
for {set i 0} {$i < $val(nn) } {incr i} {
  set array($i) 0
}
# Procedure gets a random node number from the list of nodes
# and set its array position to 1, so it won't be polled again
proc get_snode {nnode_ name} {
  upvar $name array
  while (1) {
    set i [expr round([$nnode_ value])]
    if {$array($i) == 0} {
      set array($i) 1
      return $i
    }
  }
}
# Procedure gets a random node number from the list of nodes
# and set its array position to 2, so it won't be polled again
proc get_cnode {nnode_ name} {
 upvar $name array
 while (1) {
  set i [expr round([$nnode_ value])]
  if {$array($i) == 0} {
   set array($i) 2
   return $i
  }
 }
}
# Procedure gets an exponential arrival time (for request)
proc get_time {arrival_ stop} {
 set time [$arrival_ value];
 while {$time >= $stop || $time<1000} {
  set time [$arrival_ value];
 }
 return $time;
}
# create simulator instance
set ns_          [new Simulator]
# setup topography object
set topo         [new Topography]
# create trace object for ns
set tracefd      [open $val(outputFile) w]
# $ns_ use-newtrace
$ns_ trace-all $tracefd
#create namtrace object for ns
set namfd        [open $val(noutputFile) w]
$ns_ namtrace-all-wireless $namfd $val(x) $val(y)
# define topology
$topo load_flatgrid $val(x) $val(y)
#
# Create God
#
set god_ [create-god $val(nn)]
# Global node setting
$ns_ node-config -adhocRouting $val(adhocRouting) \
            -llType $val(ll) \
            -macType $val(mac) \
            -ifqType $val(ifq) \
            -ifqLen $val(ifqlen) \
            -antType $val(ant) \
            -propType $val(prop) \
            -phyType $val(netif) \
            -channelType $val(chan) \
                  -topoInstance $topo \
                  -agentTrace ON \
            -routerTrace ON \
            -macTrace ON

# Create the specified number of nodes [$val(nn)]

for {set i 0} {$i < $val(nn) } {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0          ;# disable random motion
}
# Define node movement model
puts "Loading connection pattern..."
source $val(motionFile)

##############################################################################
######
# Set up SLP servers (nos: 10-19)
for {set i 0} {$i < $val(sn) } {incr i} {
  set j [get_snode $nnode_ array]
  set udp_sa_($i)  [new Agent/UDP/UDPslp]
  $ns_ attach-agent $node_($j) $udp_sa_($i)
  $node_($j) attach $udp_sa_($i) $val(SLPport)
  set slpsa_($i) [new Application/SLPsa]
  $slpsa_($i) attach-agent $udp_sa_($i)
  $ns_ at 0.0100000000  "$node_($j) bcast-join-group $val(MulticastAddr)"
  $ns_ at 0.0 "$slpsa_($i) start"
}
##############################################################################
#####
# Set up SLP clients (nos: 0-9)
for {set i 0} {$i < $val(cn) } {incr i} {
  set j [get_cnode $nnode_ array]
  #Setup a SLP UDP connection
  set udp_ua_($i)  [new Agent/UDP/UDPslp]
  $ns_ attach-agent $node_($j) $udp_ua_($i)
  $node_($j) attach $udp_ua_($i) $val(SLPport)
  set slpua_($i) [new Application/SLPua]
  $slpua_($i) set maxSearch_ $val(maxSearch)
  $slpua_($i) set dstport_ $val(SLPport)
  $slpua_($i) attach-agent $udp_ua_($i)
  $slpua_($i) add-scope DEFAULT
  $ns_ at 0.0 "$slpua_($i) start"
}
for {set i 0} {$i < $val(nn) } {incr i} {
  if {$array($i) == 2} {
    for {set j 0} {$j < $val(nn) } {incr j} {
      if {$array($j) == 2} {
```

```
    $ns_ at 0.0100000000  "$node_($i) bcast-join-group [expr
$val(MulticastAddr)+$j+1]"
    }
   }
  }
}
##########################################################################
#####
# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
        $ns_ initial_node_pos $node_($i) 2
}
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at [expr $val(stop)+$val(extra)] "$node_($i) reset";
}
$ns_ at [expr $val(stop)+$val(extra)+0.002] "puts \"NS EXITING...$val(scenario)\" ;
$ns_ halt"
# Output simulation parameters first to trace file
puts $tracefd "# Scenario: $val(scenario)"
puts $tracefd "# Nodes: $val(nn)"
puts $tracefd "# Server Agents: $val(sn)"
puts $tracefd "# Duplicate Servers: $val(dup)"
puts $tracefd "# Client Agents: $val(cn)"
puts $tracefd "# Total Requests: $val(reqs)"
puts $tracefd "# Simultaneous Requests: $val(simult)"
puts $tracefd "# Max Search: $val(maxSearch)"
puts $tracefd "# Service lifetime: $val(lifetime) s"
puts $tracefd "# Interarrival exponential avg: $val(exp_avg) s"
puts $tracefd "# Area: $val(x)m X $val(y)m"
puts $tracefd "# Routing: $val(adhocRouting)"
puts $tracefd "# Simulation Time: $val(stop) s  (first 1000.0 s discarded)"
##########################################################################
######
# Add service to server agents
for {set i 0} {$i < $val(a_sn) } {incr i} {
 $slpsa_($i) add-service service:ServiceNum$i://134.117.61.$i $val(stop) $val(lifetime)
}
for {set i $val(a_sn)} {$i < $val(sn) } {incr i} {
 $slpsa_($i) add-service service:ServiceNum$val(a_sn)://134.117.61.$val(a_sn)
$val(stop) $val(lifetime)
}
##########################################################################
######
# Clients send requests
set j 0
set simultaneous [get_time $arrival_ $val(stop)]
```

```
for {set i 0} {$i < $val(reqs) } {incr i} {
  set sno [expr round([$service_ value])]
  if {$j<$val(simult)}  {
   $ns_ at $simultaneous "$slpua_($j) rqst-service service:ServiceNum$sno
$val(MulticastAddr)"
  } elseif {$j==$val(simult)} {
   set simultaneous [get_time $arrival_ $val(stop)]
   $ns_ at [get_time $arrival_ $val(stop)] "$slpua_($j) rqst-service
service:ServiceNum$sno $val(MulticastAddr)"
  }  else {
   $ns_ at [get_time $arrival_ $val(stop)] "$slpua_($j) rqst-service
service:ServiceNum$sno $val(MulticastAddr)"
  }
  incr j
  if {$j==$val(cn)} {
   set j 0
  }
}
##########################################################################
######
# Start the Simulation
puts "Starting Simulation..."
$ns_ run
```

7.2 SCREEN SHOTS

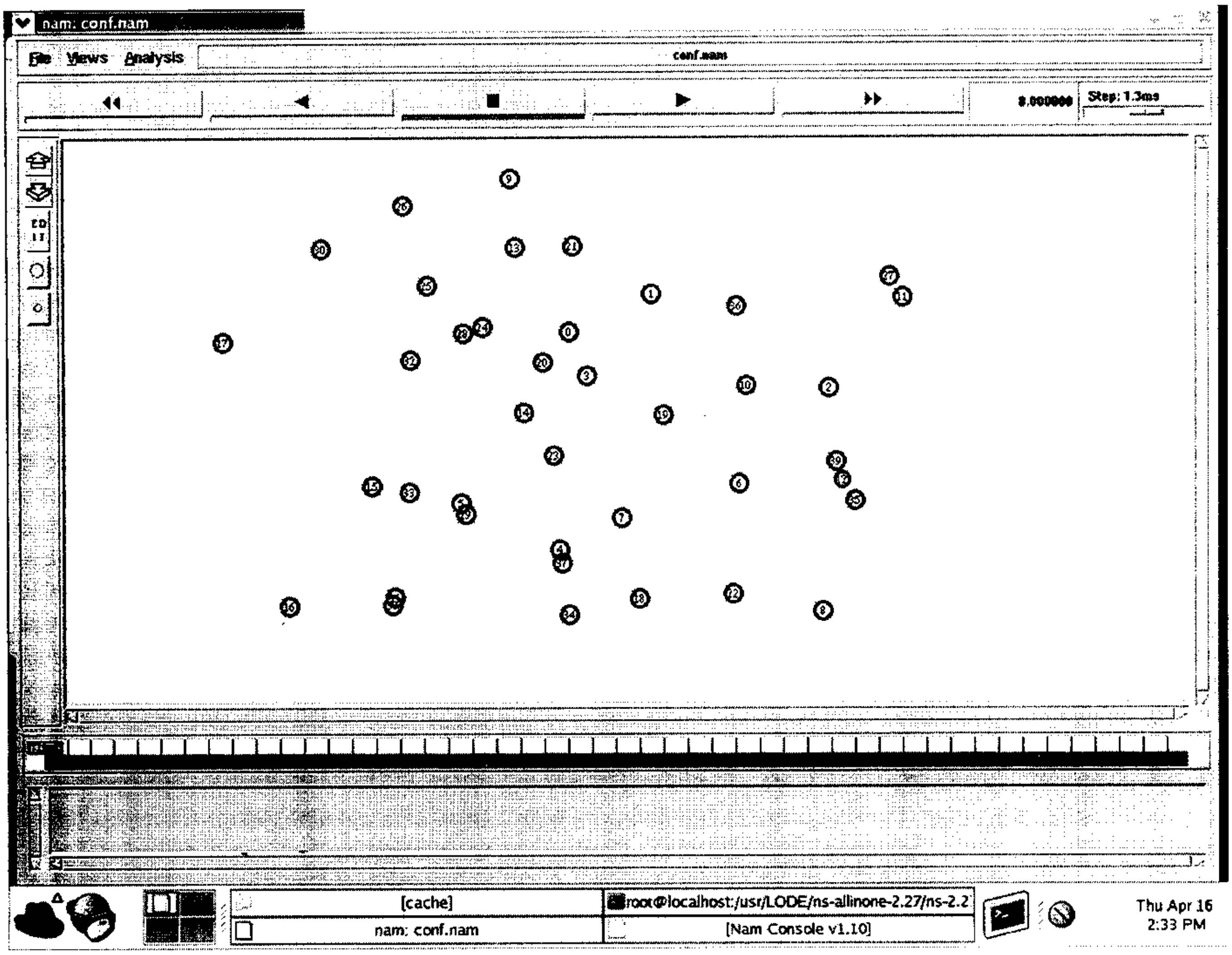


Fig.7.2.1 Positioning nodes

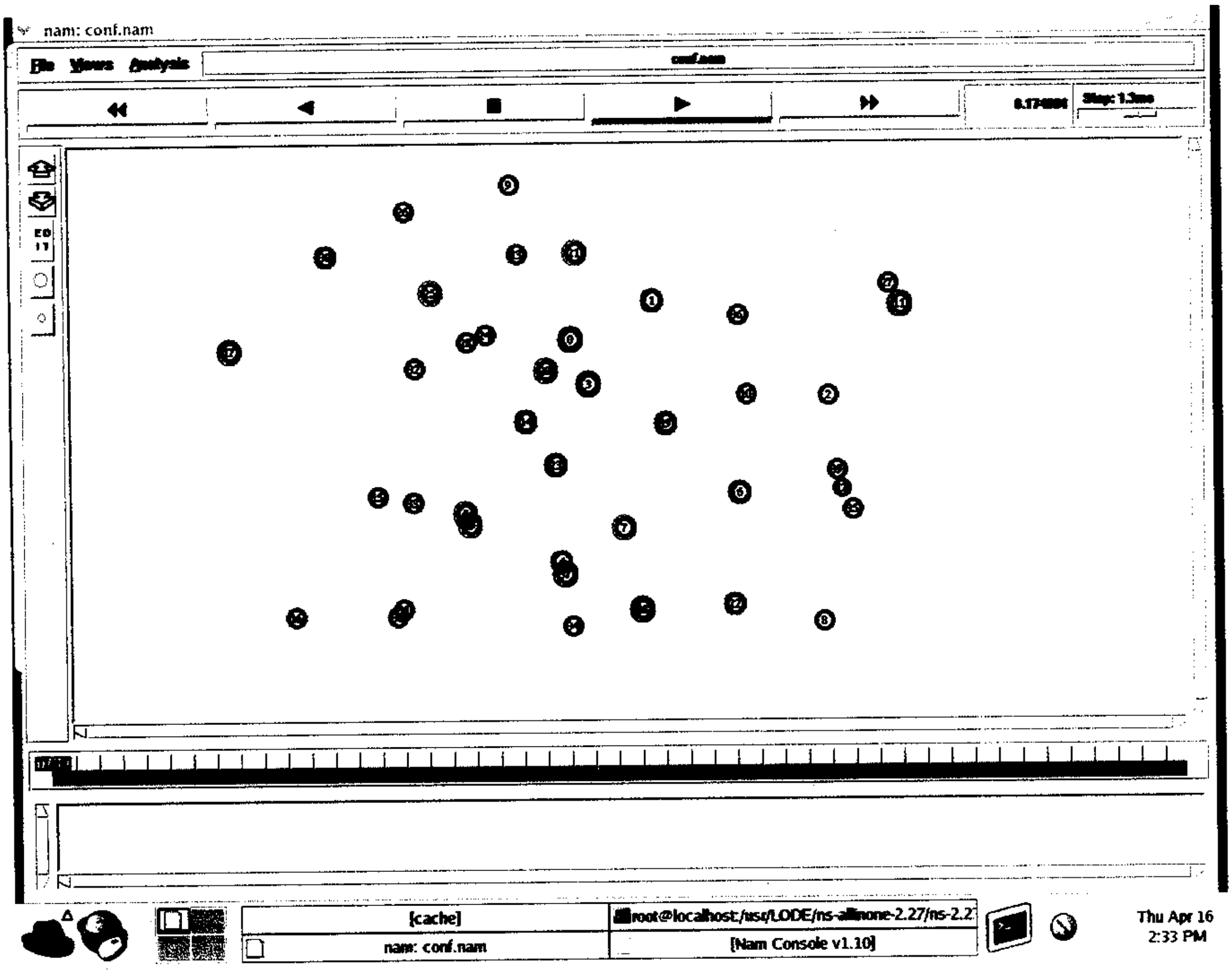


Fig.7.2.2 Client and Server hosts

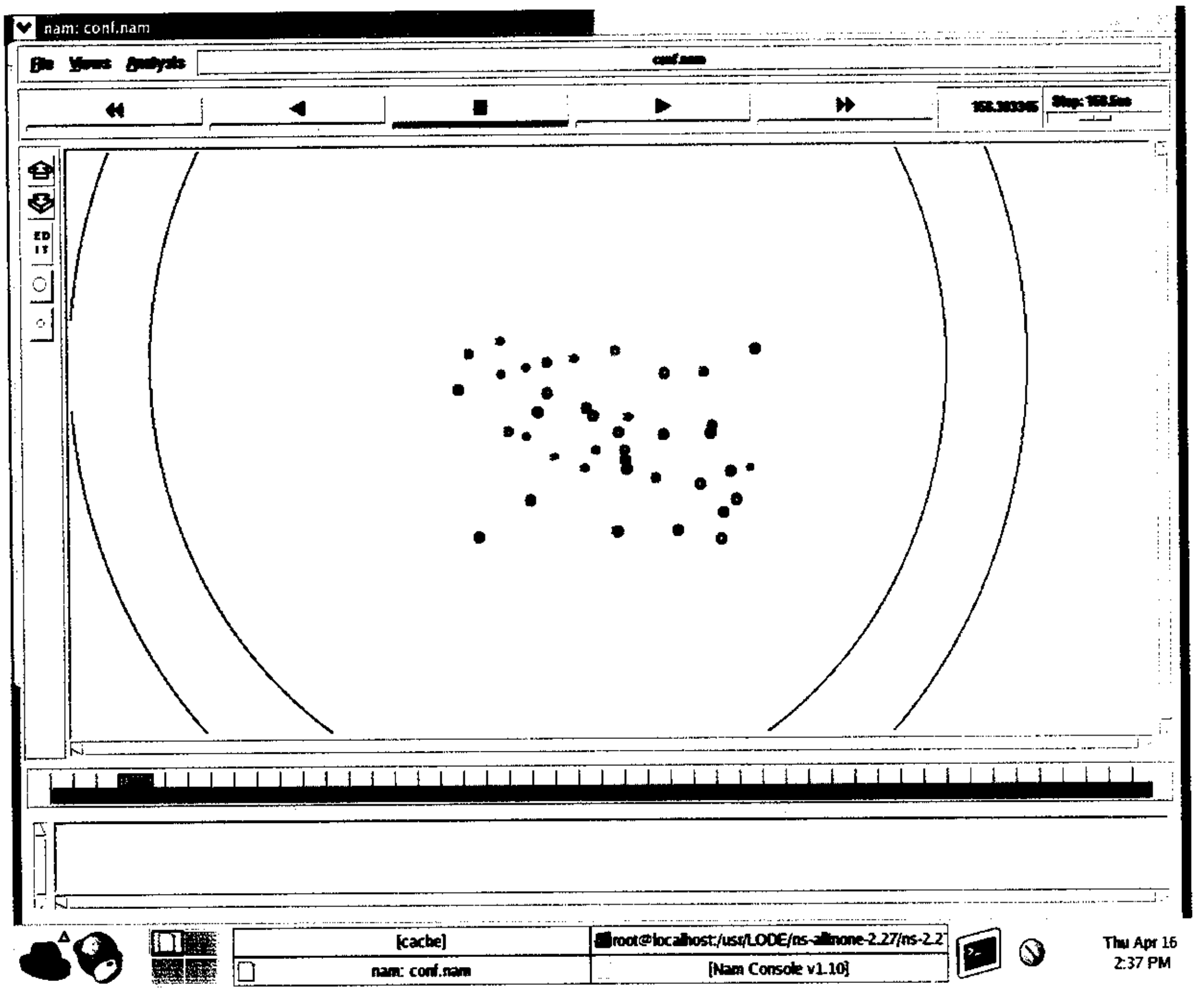


Fig.7.2.3 broadcasting of queries and summaries

8. REFERENCES

[1] H. Zhou, "A survey on routing protocols in MANETs," *Technical report: Michigan State University: MSU-CSE-03-08*, 2003.

[2] D. Chakraborty, A. Joshi, and T. Finin, "GSD: a novel group based service discovery protocol for MANETs," in *Proc. of IEEE MWCN*, 2002.

[3] S. H. Shah, K. Chen, and K. Nahrstedt, "Cross layer design for data accessibility in mobile ad hoc networks," in *Proc. of SCI 2001*, Orlando, Florida, 2001.

[4] E. Guttman, "Service Location Protocol: Automatic discovery of IP network services," *IEEE Internet Computing*, vol. 3, no. 4, 1999.

[5] I. Aydin and C.-C. Shen, "Facilitating match-making service in ad hoc and sensor networks using pseudo quorum," in *Proc. of IEEE ICCCN*, 2002.

[6] J. B. Tchakarov and N. H. Vaidya, "Efficient content location in wireless ad hoc networks," in *Proc. of IEEE MDM*, 2004, pp. 74– 85.

[7] H. M. O. Mokhtar and J. Su, "Universal trajectory queries for moving object databases," in *Proc. of IEEE MDM*, 2004, pp. 133–144.

[8] D. Pfoser, C. Jensen, and Y. Theodoridis, "Novel approaches to the indexing of moving object," in *Proc. of VLDB*, Cairo, Eygpt, 2000.

[9] Y. Cai, K. A. Hua, and G. Cao, "Processing range-monitoring queries on heterogeneous mobile objects," in *Proc. of IEEE MDM*, 2004.

[10] S. Prabhakar, Y. Xia, D. Kalashnikov, W. G. Aref, and S. Hambrusch,"Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects," *IEEE Transactions on Computers*, vol. 51, no. 10, pp. 1124–1140, October 2002.

[11] G. X. C Lu, O. Chipara, C. Fok, and S. Bhattacharya, "A spatiotemporal query service for mobile users in sensor networks," in *Proc. of IEEE ICDCS*, 2005.

[12] B.-S. Lee, K. J. Wong, B.-C. Seet, L. Zhu, and G. Liu, "Performance of mobile ad hoc network in constrained mobility pattern," in *Proc. Of ICWN 2003*, Las Vegas, Nevada, 2003.

[13] P. Yao, E. Krohne, and T. Camp, "Performance comparison of Geocast routing protocols for a MANET," in *Proc. of IC3N*, 2004, pp. 213–220.

[14] M. Greis. Marc Greis' Tutorial for the CB/LBNL/VINT Network Simulator "ns".http://www.isi.edu/nsnam/ns/tutorial/index.html.