2
2006

LIBRARY
Michigan State
University

This is to certify that the
dissertation entitled

OVERLAY MULTICAST IN MANETS

presented by

ABHISHEK PRAMOD PATIL

has been accepted towards fulfillment
of the requirements for the

Ph.D.	degree in	Dept. of Computer Science and Engineering

[illegible]
Major Professor's Signature

12-12-05
Date

MSU is an Affirmative Action/Equal Opportunity Institution

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

2/05 p:/CIRC/DateDue.indd-p.1

OVERLAY MULTICAST IN MANETS

BY

Abhishek Pramod Patil

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2005

ABSTRACT

OVERLAY MULTICAST IN MANETS

BY

Abhishek Pramod Patil

Mobile Ad-hoc Networks (MANETs) are characterized by constantly changing network topology and dynamic group membership. This makes implementing IP layer multicast in MANETs a challenging task. In recent years, several researchers have proposed the concept of overlay multicast (also known as application layer multicast). Even though application layer multicast is not as efficient as traditional IP based multicast, its flexibility in adapting to different environments and ease of implementation has contributed to its steady growth. Overlay multicast networks can be used to build roll-based priority trees. Such priority trees cannot be implemented at IP layer since IP multicast does not allow assigning priorities to different multicast sessions. It is easy to assign priorities to different multicast groups when multicast functionality is performed at the application layer.

In an ad-hoc wireless network, the position of nodes constantly changes; as a result, overlay multicast trees that are built using location information of member nodes will have a low latency. However, the performance gains of such trees are offset by the cost involved in maintaining precise location information. Periodic location updates generate a lot of overhead. As the degree of (location) accuracy increases, the performance improves but the overhead required to update and broadcast this information also increases. This dissertation proposes a design to build a sub-optimal location aided overlay multicast trees, where location updates of each member node are event based and

the location broadcasts are limited to a small set of nodes. Since most ad hoc networks are implemented in indoor environment, this dissertation also looks at various indoor location sensing techniques. It closely examines three such indoor location sensing prototype systems – LANDMARC, Location sensing using Bluetooth and Bluebot. The LANDMARC and Bluetooth Location systems were implemented in the Experimental Laboratory for Advanced Networks and Systems (ELANS) lab at CSE-MSU, while the Bluebot system was designed and implemented at IBM T. J. Watson Research Center (Hawthorne, New York) as part of an internship project.

Finally, the dissertation examines the issue of resource allocation in multicast networks (esp. in MANETs). In a typical multicast network, a single tree is built with the source node as the root. In such a tree, only a few internal nodes contribute most of the resources and are involved in performing the multicast functionality. This leads to an un-even utilization of network resources. This problem is more prevalent in MANETs where network resources are limited. A possible solution to the problem is to split the multicast content over a number of trees. Multiple trees provide several paths for the multicast content and get more nodes involved in implementing the multicast functionality. However, in this setup, not all the trees get to use the best weight edges, thus the overall multicast latency increases. The dissertation examines MEST (Multiple Edge Sharing Trees), a distributed algorithm to construct multiple edge-sharing trees for small group multicast. MEST balances the resource allocation and delay constraints by choosing to overlap certain edges that have low weights.

to my grandparents

ACKNOWLEDGMENT

First and foremost, I would like to deeply thank my advisors Dr. Abdol-Hossein Esfahanian and Dr Lionel Ni. This work would have been incomplete without their continuous guidance. I want to thank Dr. Li Xiao for her support and advice at various stages of my PhD. I am also thankful to Dr. Dan Kim and Dr. Matt Mutka for their suggestions and comments.

Very special thanks to Yunhao Liu for being such a nice friend and a great guide. I would like to thank my family for their love and encouragement. I am also grateful to the members of the eLANS lab especially Hongbo Zhou (Burnie), Frank Zhu, and Zhiwei Cen for exchanging ideas and helping me throughout. Finally, thanks to all my friends for their 'refreshing' diversions.

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

1 Introduction

This dissertation takes a close look at overlay multicast in ad hoc wireless environments. It presents the concept of prioritized overlay multicast and proposes the use of location information in order to improve the efficiency of the multicast network. Later sections also address possible solutions to the resource allocation problem in overlay multicast networks.

1.1 Research Background

Mobile ad hoc networks (MANETs) are characterized by constantly moving nodes and changing network topology. Implementing multicast in such a dynamic environment is a challenging task. As pointed out by [66], traditional IP-layer multicast [49, 58, 82] for MANETs has a lot of signaling overhead as it needs to take into account the network dynamics in addition to the (multicast) group dynamics. The widespread deployment of IP multicast has been held back by a variety of issues [33]. In recent years, several research groups have proposed the idea of overlay multicast for MANETs. Overlay multicast is multicast at the application layer. The entire multicast functionality is implemented at the application layer. The application layer relies on the underlying unicast protocols to adapt to the changing network topology. As a result, the application layer has to track only the (multicast) group dynamic. AMRoute [23], PAST-DM [42], and LGT [25] are some of the overlay multicast protocols that have been proposed for MANETs.

1.2 Executive Summary

Due to its ease of implementation and flexibility to adapt, overlay networks (though not as efficient as IP layer multicast) are finding many practical applications in Ad Hoc wireless environments. One such application is building prioritized overlay multicast trees. Since IP multicast cannot define priorities for each multicast group (or session), it is not possible to employ priority trees using the traditional IP multicast protocols. To implement such a concept would require a major restructuring of the IP multicast protocols. This dissertation presents POMA [94] – Prioritized Overlay Multicast in Ad-hoc wireless environment. POMA builds priority trees with certain nodes carrying important tasks in overlay networks, and rearranges low priority trees whenever some nodes temporarily move to a high priority network.

One major issue with application layer multicast is that it is not as efficient as IP-based multicast. As can be seen from Figure 1, data exchange between member nodes requires traversing other member nodes. The latency increases as the number of nodes increases. This delay can be greatly reduced when the overlay tree is built by taking into account the member node positions (Figure 2c). Such a tree would keep track of member node's movement and would be frequently updated to account for any change in the node positions. The nodes that are physically close to each other would be neighbors in the logical tree (Figure 2c) and the logical distance of any member node from the source node will be proportional to its actual distance from the source.

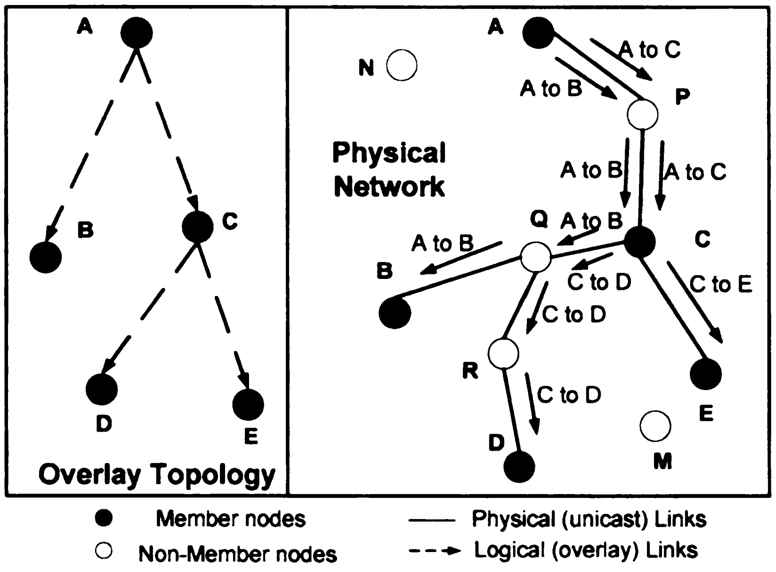


Figure 1: Overlay and physical topology

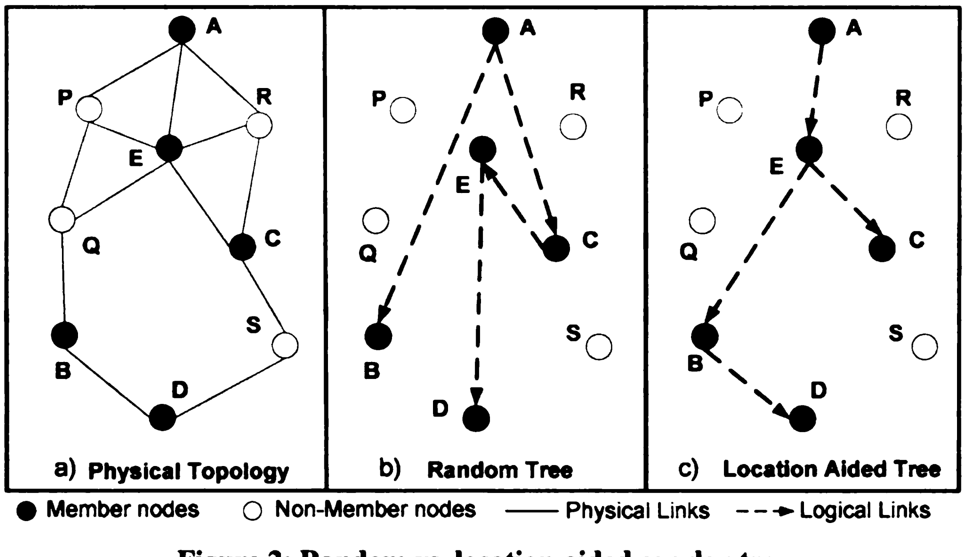


Figure 2: Random vs. location-aided overlay tree.

However, there are several issues with building location aided trees; for example:

i. The frequency at which the location information need to be updated

ii. Accuracy of location information

iii. The easiest and the fastest way to re-build the multicast tree

To address the above issues, the dissertation looks at several location sensing techniques ([63], [75], [71]) and also presents a Sub-Optimal Location-aided Overlay Multicast Network (SOLONet [73]). Unlike several other approaches, SOLONet does not require every packet to carry location information or each node to maintain location information of every other node or carrying out expensive location broadcast for each node. SOLONet strikes a good balance between the advantages of using location information (for building efficient overlay multicast trees) versus the cost of maintaining and distributing location information of every member node.

Another issue with overlay multicast (in MANETs) is the resource allocation problem. A typical overlay multicast network consists of a single tree (with source node as the root), in which only a few internal nodes contribute most resources and are involved in performing the multicast functionality. This leads to an un-even utilization of network resources. Since nodes in MANETs have limited network and battery resources, it is important that these resources are uniformly distributed across all nodes. A possible solution to the problem is to split the multicast content over a number of trees. Multiple trees provide several paths for the multicast content and get more nodes involved in implementing the multicast functionality. However, in this setup, not all the trees get to use the best weight edges, thus the overall multicast latency increases. The dissertation examines MEST [70], a distributed algorithm to construct multiple edge-sharing trees (for small group multicast) to fairly distribute the network resources.

1.3 Organization

The rest of the write-up is structured as follows. Section 2 summarizes some related research. Chapter 3 of this dissertation presents the idea prioritized multicast in overlay networks. Simulation results (section 3.2) are presented to evaluate the performance of such a network. Chapter 4 looks at several location sensing techniques and presents LANDMARC [63], BlueBot [75], and Bluetooth for location sensing [71]. Chapter 5 presents Sub-Optimal Location-aided Overlay networks (SOLONet [73]), a design that uses location information of member nodes to build an efficient overlay multicast tree. Chapter 6 address the resource allocation problem in overlay multicast in MANETs. It presents the MEST (Multiple Edge Sharing Multicast Trees) algorithm [70] for building multicast trees that try to evenly distribute the multicast load amongst all the member nodes. Finally, Section 7 presents directions for future research.

2 Related Research

This chapter looks at some of the related research in the area of overlay multicast, location sensing technologies and resource allocation in multicast. The chapter is divided into three subsections to address each topic separately.

2.1 Overlay Multicast

As mentioned earlier, traditional IP-layer multicast [49, 58, 82] for MANETs have a lot of signaling overhead as it needs to take into account the network dynamics in addition to the (multicast) group dynamics. Overlay multicast aims to solve this problem by implementing the multicast functionality at the application layer. Several overlay multicast protocols [16, 20, 23, 25, 29, 42] have been proposed and studied in recent past. However, since overlay multicast is not as efficient as IP multicast, most of these protocols try to address the issue of building an efficient overlay multicast networks.

2.1.1 NICE

The NICE [20] project aims to address the issues involved in data stream applications – real-time data applications that are characterized by a very large set of re

la

c

o

r

t

receivers having low bandwidth. NICE arranges the end host into sequentially numbered layers, which defines the multicast overlay data path. The basic operation of NICE is to create and maintain a hierarchy consisting of a set of end hosts. The members at the top of the hierarchy maintain state about O(log N) other members, where N is the number of nodes in the network. Member nodes keep information about members 'near' to them in the hierarchy and have limited knowledge about other members. This structure helps localize the effects of a member failure.

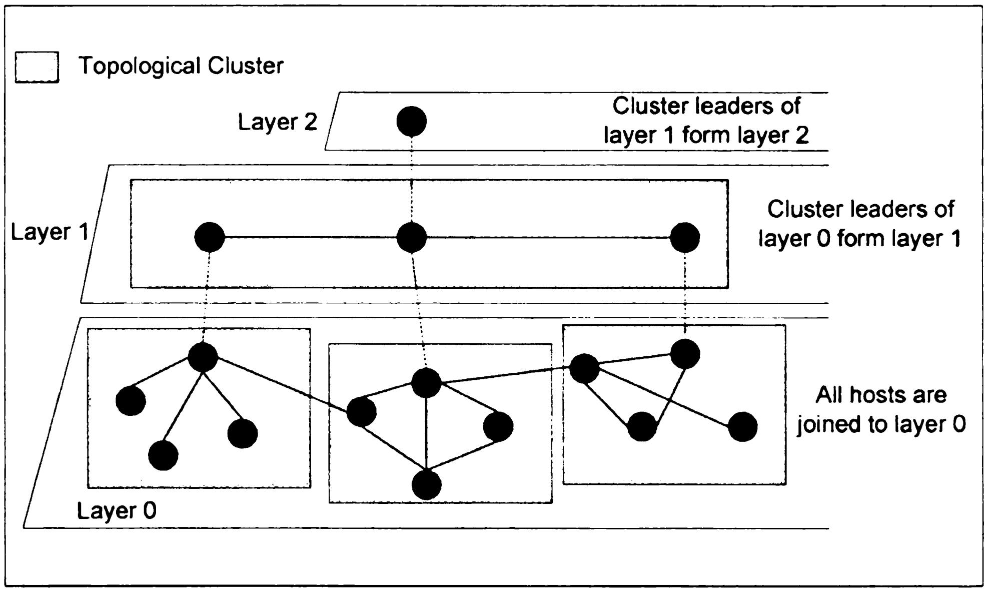

Topological Cluster
Layer 2
Cluster leaders of layer 1 form layer 2
Layer 1
Cluster leaders of layer 0 form layer 1
All hosts are joined to layer 0
Layer 0

Figure 3: Hierarchical arrangement of logical host in the NICE protocol

Hosts at each layer are partitioned into clusters that have a cluster leader. The leader selection in NICE does not make use of any location or battery strength information. In NICE, each cluster size depends on the set of hosts that are close to each other; whereas in our SOLONet approach, the cell edges (physical boundaries) define the membership. NICE is not defined for MANETs and hence it does not take into account node movement. However, the concept of 'Stress' and 'Stretch' defined by NICE can be applied to overlay multicast in MANETs. The Stress at a link defines the number of ide

del

un

2.

m

th

m

E

a

a

v

identical copies of a packet carried by that link. Stretch measures the relative increase in delay incurred by the overlay path between pairs of members with respect to direct unicast path.

2.1.2 PAST-DM

Progressively Adaptive Sub-Tree in Dynamic Mesh PAST-DM [42] is an overlay multicast protocol defined for MANETs. It tries to eliminate redundant physical links so that the overall bandwidth consumption of the multicast session is reduced. The virtual mesh in PAST-DM constantly adapts to the changes in the underlying network topology. Each node implements a neighbor discovery protocol using the extended ring search algorithm. The nodes periodically exchange link state information with their neighbors in a non-flooding manner. Thus, by looking at the link state of each node, a node gets a view of the entire topology. This information is used to build a source-based tree. PAST-DM yields a stable tree quality at the cost of higher overhead, which increases with the periodicity of the link state updates.

2.1.3 AMRoute

Ad-hoc multicast routing protocol (AMRoute [23]) makes use of user-multicast trees and dynamic logical cores to build a robust multicast network. It creates per group multicast distribution tree using unicast tunnels between group members. The bidirectional tunnels are created between member nodes that are close to each other (neighbors in the multicast tree) to form a virtual mesh. From this mesh, a subset of links is used to create a multicast distribution tree). The path taken by the unicast tunnel can change with the changing network topology without affecting the user multicast tree.

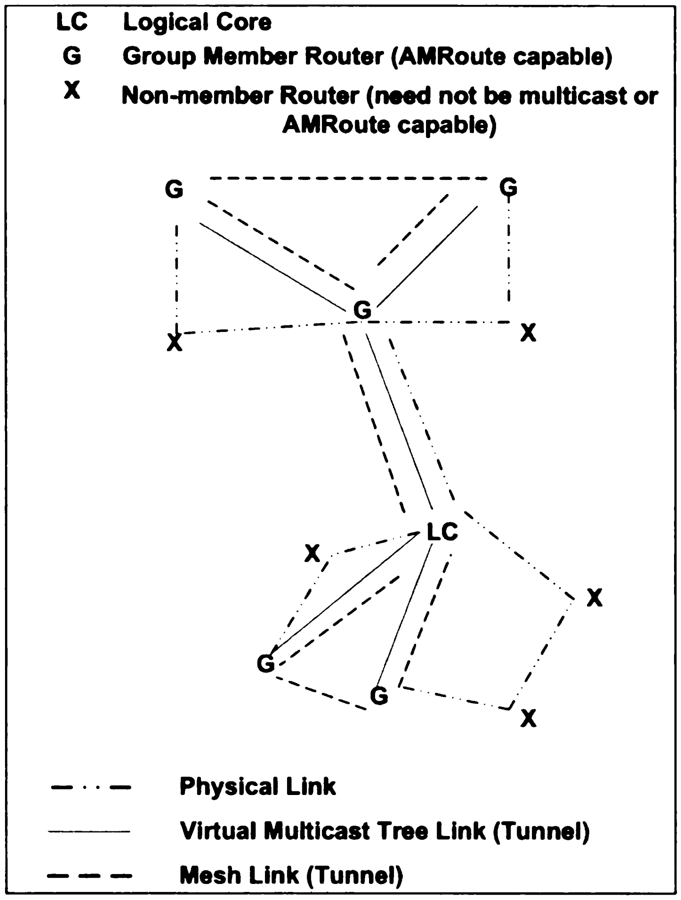

LC Logical Core
G Group Member Router (AMRoute capable)
X Non-member Router (need not be multicast or AMRoute capable)
G
G
G
X
X
LC
X
X
G
G
X
— · · — Physical Link
——— Virtual Multicast Tree Link (Tunnel)
— — — Mesh Link (Tunnel)

Figure 4: A virtual multicast user tree in AMRoute

AMRoute maintains a logical core in every tree that is responsible for mesh creation and tree creation. Non-core members cannot perform these actions; they can only act as passive responding agent. The logical core is responsible for discovering new group members and creating and maintaining the multicast tree for data distribution. The core can migrate dynamically depending on the group membership or network connectivity. Having one core per group has the advantage of reducing the signaling overhead. In absence of a core node, every node would have to respond to the signaling message sent by every other node in the network resulting in a control traffic that is proportional to the square of the group size. Since the non-core nodes have to respond to the signaling messages sent by the core node only, the signaling traffic in the network is directly pro

gro

alg

sa

lea

2.

[5

he

ar

r

(

(

proportional to the group size and not the total number of nodes in the environment. A group may have more than one core for a short period. A special core resolution algorithm is designed to solve the problem of more than one core in the same group. The same algorithm can help in the assignment of a new core incase the earlier core node leaves the group or is unavailable due to link or node failure.

2.1.4 Location Guided Tree (LGT)

Location Guided Tree (LGT) [25] is a small group multicast scheme similar to DDM [50]. It builds overlay multicast trees (in MANET) using geometric distance as the heuristic of link costs. The scheme proposes two tree construction algorithms: greedy k-ary tree construction (LGK) and Steiner tree construction (LGS).

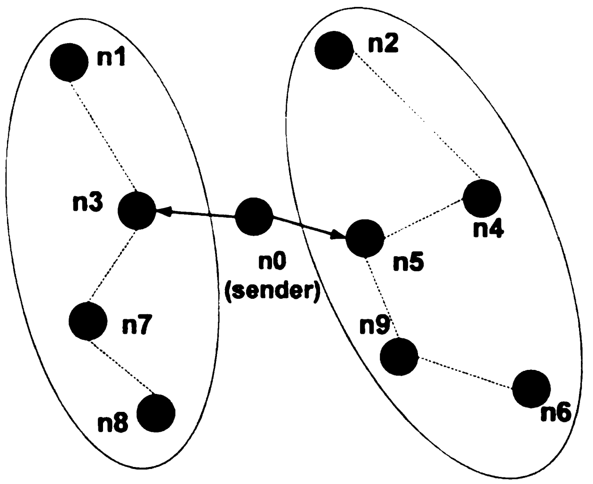


Figure 5: Location Guided k-ary tree construction. Only member nodes are shown

The algorithms are based on the assumption that longer geometric distances require more network-level hops to reach destination. LGS constructs the Steiner tree using link costs as their geometric lengths. With LGK, source node selects k nearest neighbors as its children and partitions the remaining nodes according to their distance to the children nod

How

freq

2

v

A

I

nodes. Similar to LGT, our approach uses location information to form an overlay tree. However in our approach, the location updates are event based and hence not very frequent. LGT is a small group multicast scheme and may not scale well.

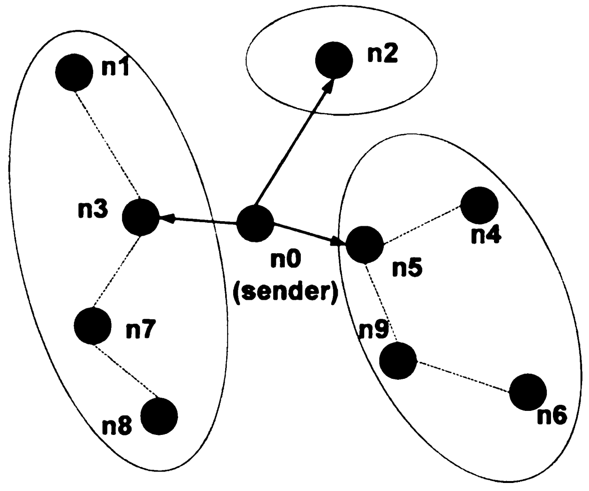


Figure 6: Location Guided Steiner tree construction. Only member nodes are shown

2.2 Location Tracking Systems for Indoor Environment

Over years, several research groups (both in the industry and academia) have been working on the topic of indoor location sensing. AT&T Olivetti Research Laboratory's Active Badge [91] represents pioneering work in this area based on infrared technology. However, due to the line-of-sight requirement and short-range signal transmission, researchers realized that infrared technology is not a very good solution for this problem. In recent years, most of the researches have been adopting the radio frequency (RF) technology for this purpose instead. Examples are RADAR project by Microsoft Research [19], SpotON [46] done at University of Washington. Like the RADAR project, Project Aura [85] done at Carnegie Mellon University also tries to utilize the IEEE 802.11 wireless technology for location sensing in addition to its use as a network infrastructure. Other projects also include Active Bats system [2] by AT&T Research

L

y

U

is

k

p

c

2

i

Laboratory using ultrasonic technology, the Cricket Indoor Location System [4, 78] at MIT with a combination of ultrasonic and RF technologies, TinyOS [53] RF motes by UC Berkeley, and the Cooltown project [6] by Hewlett Packard. The HiBall Tracker [92] is a location positioning system that works by placing light emitting diodes (LEDs) in a known pattern on the ceiling and using a special LED detector, called a HiBall, on a person's head. The HiBall senses the LEDs on the ceiling and uses this information to compute location using Kalman-based filters to negate the effects of noise.

2.2.1 RADAR

RADAR [19] is a radio-frequency (RF) based system for locating and tracking users inside buildings. RADAR operates by recording and processing signal strength information at multiple base stations positioned to provide overlapping coverage in the area of interest. It combines empirical measurements with signal propagation modeling to determine user location and thereby enable location aware services and applications. The main advantage of the RADAR system is its simplicity and ease of set-up. Since the system leverages the signal strength and signal to noise ratio available from the WaveLAN network interface, it requires very few base stations and uses the same infrastructure that provides general wireless networking in the building. The difficulty is that the object being tracked must support a WaveLAN NIC, which may be impractical on small or power constrained devices. In addition, it is not trivial to generalize this approach to multi-floored buildings or three dimensions. Besides, signal behavior in an indoor environment is unpredictable and depends on various factors. The RADAR system can position an object in a 3-4m circle with 50% probability, which is one major disadvantage of the system.

2.2.2

(

locat

sprea

MHz

ultra

the

they

max

sim

the

inc

inf

of

ne

in

2.

b

h

t

e

a

2.2.2 Cricket

Cricket [4, 78] uses a combination of RF and ultrasound technologies to provide a location-support service to users and applications. Wall and ceiling-mounted beacons are spread through the building, publishing information on an RF signal operating in the 418 MHz AM band. With each RF advertisement, the beacon transmits a concurrent ultrasonic pulse. Listeners attached to devices listen for RF signals, and upon receipt of the first few bits, listen for the corresponding ultrasonic pulse. When this pulse arrives, they obtain a distance estimate for the corresponding beacon. The listeners run maximum-likelihood estimators to correlate RF and ultrasound samples (the latter are simple pulses with no data encoded on them) to pick the best one. The main advantage of the Cricket system is its accuracy – 30cm. It scales well as the number of devices increases. Its decentralized architecture makes it easy to deploy. However, the infrastructure involves installing and coordinating a lot of devices. The system makes use of ultrasonic devices, which are expensive and difficult to handle. The ultrasonic devices need to be synchronized so that their beacons don't interfere. This requires additional RF infrastructure further increasing the cost.

2.2.3 SpotON

SpotON [46] is a new tagging technology for three dimensional location sensing based on radio signal strength analysis. SpotON researchers have designed and built hardware that will serve as object location tags, part of a project called SpotON. SpotON tags use received radio signal strength information (RSSI) as a sensor measurement for estimating inter-tag distance. Using many collocated nodes, the measured positional accuracy can be improved through algorithmic techniques and erroneous distance meas

auto

desig

abou

simi

sigr

2.2.

Me

car

tria

int

en

pa

2.

si

y

6

d

measurements caused by signal attenuation (e.g. by metal objects in the area) can be automatically factored out. The main contribution of this paper was to come with a design and analysis of hardware meant for location sensing only. Another good thing about SpotON project is that it aims to gather real-world information not available using simulation techniques. However, using algorithmic techniques to remove errors due to signal variation may not be the best solution.

2.2.4 Aura

The Aura [85] project makes use of the wireless network infrastructure at Carnegie Mellon University for determining a user's location both indoors and outdoors while on campus, and at a higher resolution than GPS. It employs a table-based lookup for triangulation of a user's location. The system makes use of existing wireless infrastructure, which is a major advantage. However, signal strength in indoor environment varies randomly (due to movement of people, placement of cubicle partitions, furniture, etc) and hence cannot be used for distance calculation.

2.3 Resource allocation in Multicast Networks

As seen from section 2.1, most of the multicast protocols are based on building a single multicast tree and geared towards improving the multicast efficiency. In recent years, the issue of resource allocation has caught the attention of many researchers [57, 60, 67, 96]. This section looks at some of the existing protocols that try to evenly distribute the load amongst member nodes.

2.3.1

Y

Their

work

that

of st

rejec

it is

algo

mul

wei

be

sin

wo

2.

m

(I

tr

n

a

u

2.3.1 K-MST

Young [96] suggested a k-MST algorithm for building a reliable mesh structure. Their algorithm builds multiple edge-disjoint minimum spanning trees. The algorithm works by removing an edge from the graph once a tree uses it. This tends to build trees that greatly vary in their weight and hence the multicast latency. During the later stages of such an algorithm, the newer trees end-up having the high weight edges (which were rejected by the earlier trees). The final tree would be the one with the highest delay since it is using all the high weight edges that were not used by the earlier trees. In the k-MST algorithm, if one tree fails, data can be sent over another redundant tree. However, the multicast delay would increase since the data is being transferred over a tree with a high weight. It should be noted that such a method works well only for a mesh case. It cannot be used to address the resource allocation problem since the delay involved in simultaneous transfer of multicast content would be unacceptable. Some of the content would come over a tree that has the highest weight (maximum delay).

2.3.2 SplitStream

SplitStream algorithm [60] tries to distribute the multicast load by constructing multiple multicast trees. However, SplitStream is based on a complex infrastructure (Pastry [80] and Scribe [81]) and requires the help of non-members to construct multicast trees. Although, SplitStream focuses on improving the resource utilization of the network, it doesn't pay much attention to the delay constraints. The algorithm attempts to accommodate member nodes with different bandwidth capacities. In doing so, it makes use of the Scribe mechanism to limit the number of outgoing connections from a member node.

result

2.3.3

acco

serv

up

ded

node. As a result, a 'child node' might be forced to connect further down the tree resulting in higher delay.

2.3.3 Miscellaneous

The strategy adopted in [57, 67] are essentially different from ours. They rely on accounting and charging methods to solve the unfairness problem, i.e., group members serving as forwarders charges for their service rendered to their children in order to build up its contribution to the system. Overcast [48] maintains a single tree and uses a dedicated server to optimize the bandwidth utilization across the network.

3

mul

imp

serv

nod

hig

sm

ov

fle

ov

di

ca

ch

re

co

3 Prioritized Overlay Multicast

In many systems, some participating nodes might be members of more than one multicast trees or may wish to build a temporary tree in order to perform certain important tasks. The priority of these trees can be defined by the importance of the service they provide. For the success of such an application, it is necessary that member nodes affiliated to more than one multicast tree listen to the members belonging to the highest priority tree (in their affiliation) only. These member nodes would have to be smart enough to ignore incoming messages from nodes in the low priority trees. In overlay networks since the topology is built at the application layer, it provides greater flexibility in the design. Building role-based priority trees is one such advantage of overlay networks. Different priority trees can be built in the same environment to provide different kinds of services. With multiple priority trees, nodes belonging to different trees can switch among networks depending on what functionality they want to provide. This chapter explains the concept of prioritized overlay multicast trees and presents simulation results that show the feasibility of that idea. POM [72], [94] builds priority trees with certain nodes carrying important tasks in overlay networks, and rearranges low priority trees

identi

to bu

on sy

3.1

kind

they

bet

tim

its

tre

Th

cu

nc

m

tc

le

a

(

r

v

t

trees whenever some nodes temporarily move to a high priority network. The simulations identify a suitable unicast (ad hoc) routing protocol, explore use of location information to build efficient trees, and study the impact of density of wireless nodes and packet size on system performance.

3.1 POMA Model

Different priority trees can be built in the same environment to provide different kinds of services. The priority of the trees would depend on the importance of the service they provide. With multiple priority trees, nodes belonging to different trees can switch between networks depending on what functionality they want to provide. At any given time, a node would associate itself only with one priority tree (the highest priority tree in its set) and it would have to ignore incoming messages/data from members in low priority trees. A node that initiates the formation of a new priority tree can supply priority tokens. The value of the priority token can determine the priority of the tree. Thus if a node is currently a member of priority tree '*i*', it would not listen to message/data from member nodes belonging to '*i-1*' or lower priority trees. Once the '*i*' priority tree is dissolved, a member node will down-shift to the next highest priority in its set. When a node decides to form a high priority tree or receives an invitation to join a high priority network, it may leave behind several 'orphan' nodes. These orphan nodes would now have to attach to another node in the original network to maintain their connectivity with the original tree. One way to tackle this problem would be for the departing node to send a control message to its children informing them of its intension of leaving the network. Along with this message, the exiting node will give them its parent address. The child node can thus contact its grandparent node, connect to it and start getting data from it as illustrated in F

Thu

wou

dis

wo

phy

noo

Fig

M

in Figure 7a. The member nodes use multi-hop means of communicating with each other. Thus referring to Figure 7a again, nodes F and I may not be close to each other and yet would be neighbors to each other in the logical topology. Later sections of the dissertation describe the idea of location-aware overlay trees, where the logical topology would try to match the physical topology; thus making F and I close to each other in the physical topology too. If for some reason the grandparent is unable to support the new node(s), it will pass on the connection request to the source node (node A in case of Figure 7a).

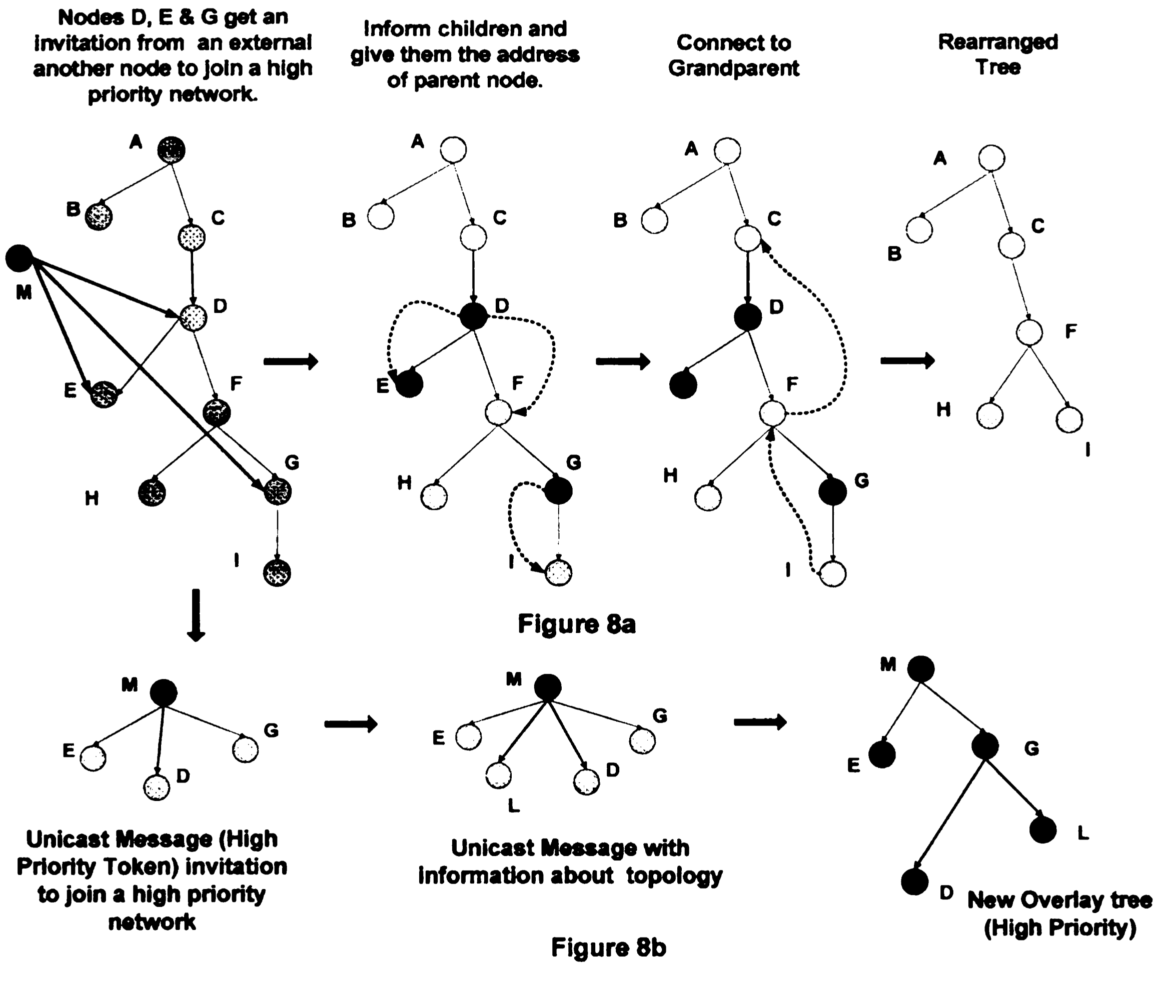

Nodes D, E & G get an invitation from an external another node to join a high priority network.
Inform children and give them the address of parent node.
Connect to Grandparent
Rearranged Tree
A
B
C
D
E
F
G
H
I
M
Figure 8a
L
Unicast Message (High Priority Token) invitation to join a high priority network
Unicast Message with Information about topology
Figure 8b
New Overlay tree (High Priority)

Figure 7: Formation of a high priority tree and rearrangement of the old tree

top

Fig

exte

to f

uni

abo

loc

the

use

so

3.

ex

ex

pr

re

p

p

s

o

l

s

In location-aware trees, the source node has location information of the entire topology and should be able to redirect the orphan node's connection to a suitable node. Figure 7b shows the formation of the new (high priority) tree. At the beginning, the external node M contacts nodes D, E & G of the original tree and another external node L to form a high priority network. In the first step, M asserts its priority by sending a unicast token message to each of the desired nodes. In Step 2, M exchanges information about the formation of the tree topology. The topology information can be based on the location information of nodes with respect to each other. Step 3 is the final formation of the tree. The number of steps can vary depending on the implementation or the algorithm used for tree formation. In our approach, the location information is available to the source node and hence it fixes the topology and informs the other member nodes.

3.2 Simulation Methodology and Results Analysis

Simulations were carried out using Network Simulator (ns2.26) [8] with the CMU's extension (MONARCH Project) for MANETs. As of this writing, ns2 does not have any extension for simulating overlay multicast. With the help of bash scripting and C-programming, the traffic pattern generated by CMU's cbrgen utility was modified to represent an overlay network. CMU's setdest utility was used to generate different node positions and movement patterns. The parameters considered were number of nodes, pause time, speed, time of simulation and the area of simulation. The nodes in the simulation move according to the 'random waypoint' model [51]. Since the performance of the protocols is very sensitive to the movement pattern, the result values are average of 10 different movement patterns (for every combination considered). The first set of simulations identifies a good unicast routing protocol that can be used with overlay network

efficient

nodes to

3.2.1

Sir

nodes.

unicast

protoco

MANE

to find

order

show

due t

The

(hun

mer

to t

pro

all

pe

D

c

I

network. Section 3.2.2 then explores the use of location information to build more efficient overlay tree and later, Section 3.2.3 studies the impact of density of wireless nodes to the system performance.

3.2.1 Protocol Identification

Since an overlay network forms a logical network consisting of multicast member nodes, the underlying network looks at the data exchange between member nodes as a unicast communication. This unicast communication can make use of the various routing protocols DSR [51], AODV [77], DSDV [76] or TORA [68] that have been proposed for MANET. This section examines different unicast ad-hoc routing protocols in an attempt to find an efficient protocol with low latency, less drop rate and minimal overhead. In order to identify a good ad-hoc routing protocol, simulation results for overlay trees shown in Figure 8 were analyzed. TORA was one of the candidate protocols; however, due to its very poor performance it was eliminated in the initial rounds of simulations. The speeds considered in the simulation were 1m/sec (human walking) and 5m/sec (human running). The average time to complete the transfer of a 100K file to all the member nodes along with the average drop ratio (ratio of total number of packet dropped to the total number of sent packets) and average protocol ratio (ratio of total number of protocol message packets to the total no of sent packets) was observed and analyzed. For all of the above three, lower values mean better performance. Figure 9 compares the performance of the three protocols in terms of the average completion time. It is clear that DSDV shows poor performance. Figure 10 shows that AODV has a very high drop ratio compared to the other two protocols, while Figure 11 indicates that DSDV has a very high protocol overhead. Table 1 shows the parameters used for the simulations.

Table 1: Simulation parameters for protocol comparison

Simulation Area (m^2)	500x500
Total no. of nodes	25
Total no. of member nodes	15 (Figure 8)
Speed of nodes (m/sec)	1, 5
Pause Times (Sec)	0, 5, 10, 15, 20
Simulation Time (Sec)	400
File Size (KB)	100
Packet Size (bytes)	512, 1024
Routing protocols	DSR, DSDV, AODV
No. of scenario patterns tested	10

Table 2: Simulation parameters for location-aided tree.

Simulation Area (m^2)	800x800
Total no. of nodes	25
Total no. of member nodes	10
Speed of nodes (m/sec)	1
Pause Times (Sec)	0 (continuous movement)
Simulation Time (Sec)	400
File Size (KB)	500
Packet Size (bytes)	1024
Routing protocols	DSR
No. of scenario patterns tested	7

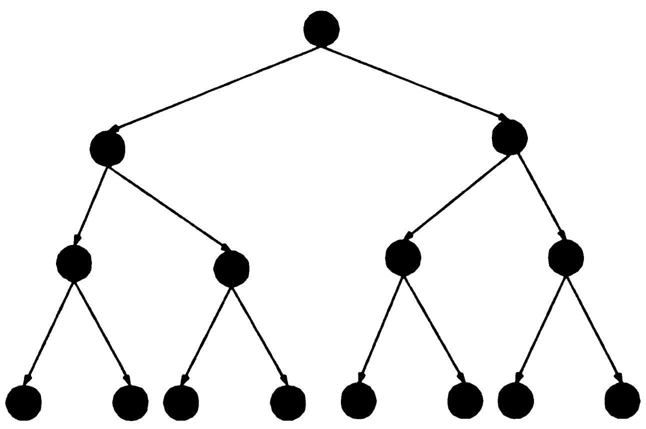

(Twin Topology)

Figure 8 : Topology used for testing different protocols.

DSR and AODV show similar performance in terms of transfer time. However, AODV has a very high drop ratio and the drop ratio increases with the packet size. It was also ob

protoco

of a *he*

also observed that higher packet size reduces the transfer time. AODV has higher protocol overhead compared to DSR. AODV normally requires periodical transmission of a *hello* message (with a default rate of one per sec).

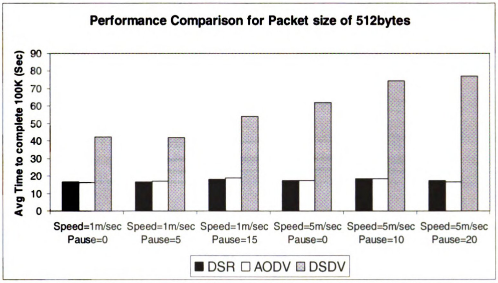


Figure 9: Comparison of avg completion time for packet size of 512 for 3 protocols

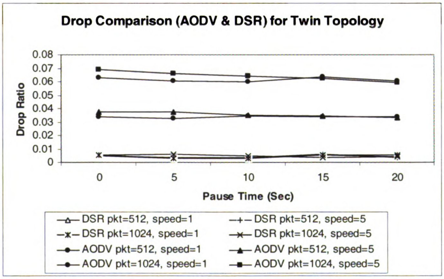


Figure 10: Drop ratio comparison for DSR & AODV (Twin Topology).

DSR carries with it the advantage of source routing where the packets carry the information about the route to the destination. As a result, aside from the initial route discovery overhead, DSR does not exhibit a high routing overhead. On the other hand, in case o

maint

DSR

case of AODV, each node participating between the source and the destination needs to maintain information about the route. With these factors in mind, *packet size of 1024 and DSR (underlying routing protocol) were chosen for further simulations.*

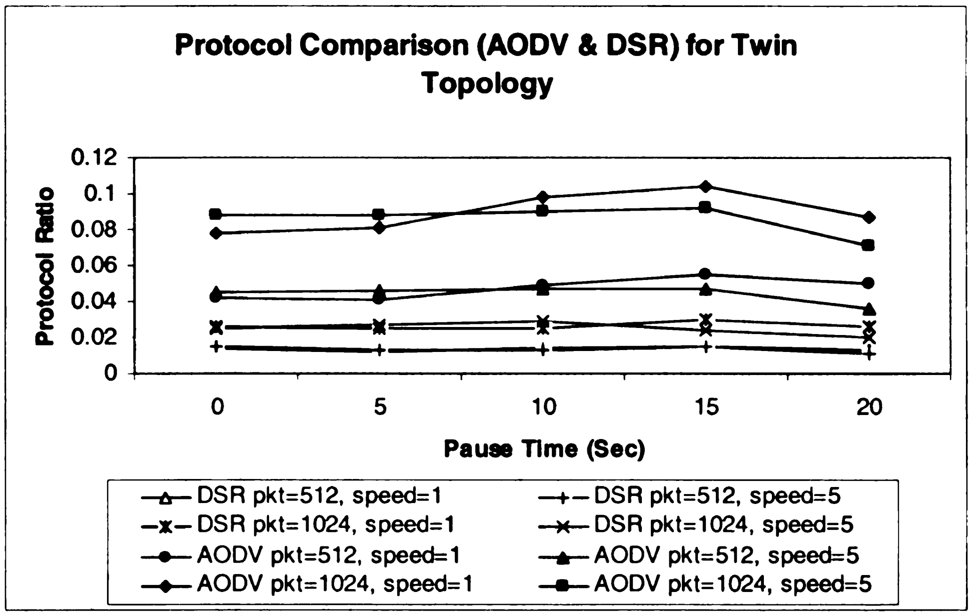


Figure 11: Performance in terms of protocol overhead for twin topology.

3.2.2 Location-based Trees

Earlier, in Section 2.1.4, an approach to use the geometric distance as a heuristic in tree formation in ad-hoc networks was shown [25]. Simulations were carried to compare the performance of a location aware overlay trees versus a tree built at arbitrarily. Table 2 shows the simulation parameters. Figure 12 shows the random tree used for this set of simulations. Figure 13 shows the performance for seven different movement scenarios. It can be seen that location-aware overlay trees have a lesser latency compared with trees built without any location information.

Obtaining exact location of an object is a difficult task, especially in indoor environments where ad hoc networks are usually implemented. Frequent updates would be necessary to maintain the accuracy of this information. As the node number and mobilit

tradeof

distrib

motiv

accur

mobility increases, the update frequency would exponentially increase. There is a tradeoff between the advantages of building a location-aided overlay tree versus the distribution and precision of this location information. This was one of the key motivations for investigating location aided overlay trees and techniques to gather accurate location information.

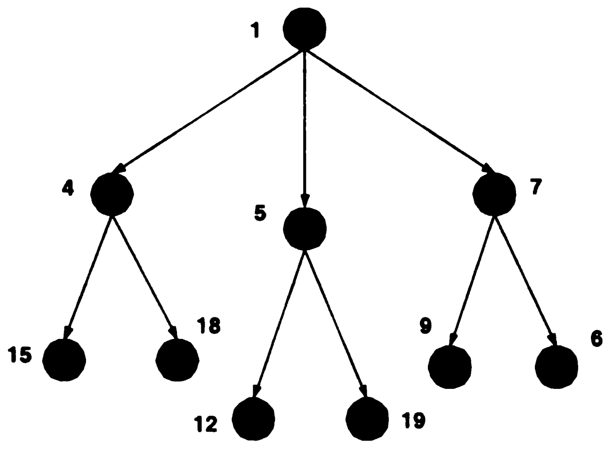


Figure 12: Overlay tree without any location information.

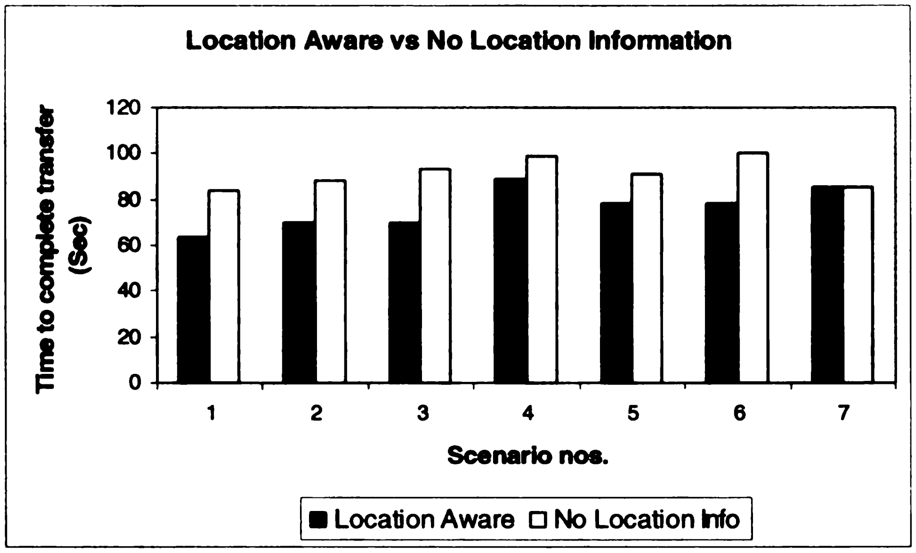


Figure 13: Performance of location aware overlay tree and an overlay tree without any location information about member nodes.

3.2.3

V

node:

densi

resul

8. Tl

15.

Figu

nod

the

sca

sou

3.2.3 Density of Wireless Nodes

Wireless nodes have a limited coverage area. As a result, the density of wireless nodes in a given area greatly influences the performance of the network. With higher density of nodes in a given area, there are more nodes to perform multi-hop forwarding resulting in improvement in the overall performance. The logical tree was same as Figure 8. The area was incrementally changed from $100x100m^2$ to $800x800m^2$. Also, a total of 15, 25, 40 and 60 nodes were tested with the different areas. The results are presented in Figure 14 and Figure 15. In smaller areas, even with fewer nodes, the coverage region of nodes overlaps and hence the performance is hardly affected by the number of nodes in the environment. However, as the area in the simulation is increased, the nodes are scattered and the coverage areas have very little overlap. The number of hops from source to destination increases and, as a result, the latency increases.

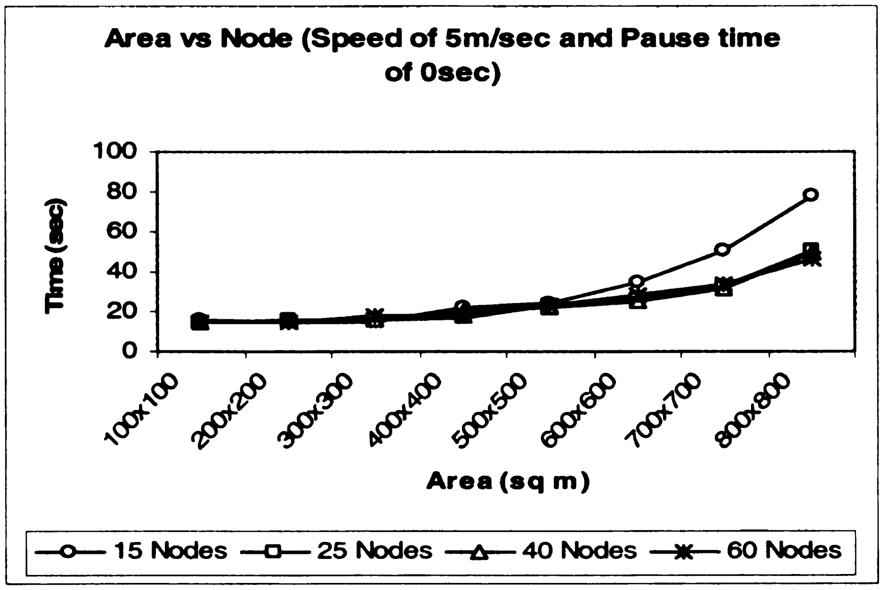


Figure 14: Performance comparison for different number of nodes and areas.

3.3

over

the

that

info

buil

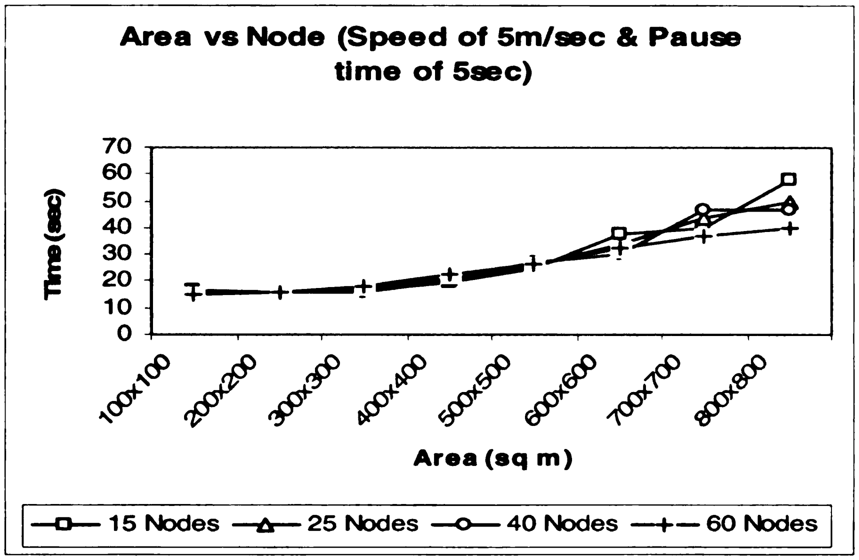


Figure 15: Performance comparison for different number of nodes and areas.

3.3 Contributions

This dissertation chapter proposes a new infrastructure-less need-based prioritized overlay multicast model (POM). The idea of building several role-based priority trees in the same environment can find many interesting applications. The chapter also showed that overlay trees with lower latencies could be designed by making use of location information. Later, in Chapter 5, location information is used along with other factors to build efficient overlay trees.

4 Location Sensing Techniques

The proliferation of wireless technologies, mobile computing devices, and the Internet has fostered a growing interest in location-aware systems and services [19, 46, 54, 85, 97]. Many applications need to know the physical location of objects. One such application is position based approach for routing [25, 88]. Over the years, many systems have addressed the problem of automatic location sensing. Triangulation, scene analysis, and proximity are the three principal techniques for automatic location-sensing [44]. One of the most well known location-based systems is the Global Positioning System (GPS), a satellite-based navigation system made up of a network of 24 satellites placed into orbit. GPS is widely used to track moving objects located outdoors. However, GPS, as it is satellite dependent, has an inherent problem of accurately determining the location of objects located inside buildings. Most ad hoc wireless networks are implemented in indoor environment. The next three sub-sections of this dissertation present location positioning/tracking techniques for indoor environment. Section 4.1 presents LANDMARC [63] – a prototype indoor location sensing system that uses active RFID technology. Section 4.2 presents BlueBot [75], location tracking using off-the-shelf Wi-

Fi position system and passive RFID. And finally, section 4.3 presents Bluetooth for location sensing [71]. The LANDMARC and Bluetooth systems were implemented in the Experimental Laboratory for Advanced Networks and Systems (ELANS) lab at CSE-MSU, while the Bluebot system was designed and implemented at IBM T. J. Watson Research Center (Hawthorne, New York) as part of an internship project.

4.1 LANDMARC: Location-Sensing Using RFID

The LANDMARC [63] system uses active Radio Frequency Identification (RFID) technology and the concept of reference tags.

4.1.1 RFID Basics

RFID is a means of storing and retrieving data through electromagnetic transmission to an RF compatible integrated circuit. There are several advantages of using RFID technology – no contact and non-line-of-sight, etc. All RF tags can be read despite extreme environmental factors, such as snow, fog, ice, paint, and other visually and environmentally challenging conditions. They can also work at remarkable speeds. In some cases, tags can be read in less than a 100 milliseconds. The other advantages are their promising transmission range and cost-effectiveness.

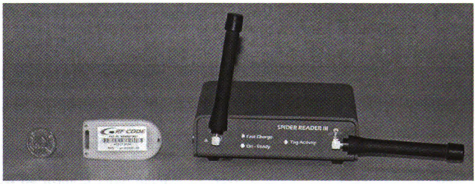


Figure 16: The RFID reader and tag used in our prototype system.

RFID tags are categorized as either passive or active. Passive RFID tags may operate either with or without a battery. They reflect the RF signal transmitted to them from a reader and add information by modulating the reflected signal. Passive tags are much lighter and less expensive than active tags, and offer a virtually unlimited operational lifetime. However, their read ranges are very limited. Active tags contain both a radio transceiver and battery to power the transceiver. Since there is an onboard radio on the tag, active tags have more range than passive tags. For instance, the products used in our experiment were active tags, which have a read range of 150 feet. If necessary, this range can be increased to 1000 feet with special antenna.

In the LANDMARC system, the RFID Reader's operating frequency is 308 MHz. It also has an 802.11b interface to communicate with other machines. The detection range is 150 feet. The reader provides digital control of read range by providing configuration software and API with 8 incremental read ranges. Each reader can detect up to 500 tags in 7.5 seconds. Each RFID tag is pre-programmed with a unique 7-character ID for identification by readers. Its battery life is 3-5 years. Each tag selects a random time (with an average of 7.5 seconds) to send its unique ID signal. Note that the RFID reader has eight different power levels. Based on the signal strength received by the RFID reader, the reader will report or ignore the received ID, where power level 1 has the shortest range and level 8 has the longest range.

4.1.2 LANDMARC Approach

Due to the dynamic interferences and various environmental factors, even a static object could be reported in different locations from time to time. Accuracy can be improved by using a large number of readers and proper placement of those readers.

RFID readers are very expensive. In order to increase accuracy without placing more readers, the LANDARC (Location Identification based on Dynamic Active RFID Calibration) system employs the idea of having extra fixed location reference tags to help location calibration. These reference tags serve as reference points in the system (like landmarks in our daily life). The proposed approach has three major advantages. First, there is no need for a large number of expensive RFID readers. Instead, LANDMARC uses several cheaper RFID tags. Second, the environmental dynamics can easily be accommodated. Our approach helps offset many environmental factors that contribute to the variations in detected range because the reference tags are subject to the same effect in the environment as the tags to be located. Thus, it can dynamically update the reference information for lookup based on the detected range from the reference tags in real-time. Third, the location information is more accurate and reliable. The LANDMARC approach is more flexible and dynamic and can achieve much more accurate and close to real-time location sensing. Obviously, the placement of readers and reference tags is very important to the overall accuracy of the system.

Suppose there are ***n*** RF readers along with ***m*** tags as reference tags and ***u*** tracking tags as objects being tracked. All the readers are configured in continuous mode (continuously reporting the tags that are within the specified range) and a detection range of 1-8 (meaning the reader will scan from range 1 to 8 and keep repeating the cycle with a rate of 30 seconds per range). The Signal Strength Vector of a tracking/moving tag is denoted as: $\vec{S} = (S_1, S_2, \ldots\ldots, S_n)$ where S_i denotes the signal strength of the tracking tag perceived on reader *i*, where $i \in (1, n)$. For the reference tags, the corresponding Signal Strength vector is denoted as: $\vec{\theta} = (\theta_1, \theta_2 \ldots\ldots \theta_n)$ where θ_i denotes the signal strength. For each individual tracking tag p, where $p \in (1,u)$, we define: $E_j = \sqrt{\sum_{i=1}^{n} (\theta_i - S_i)^2}$

where $j \in (1,m)$, as the Euclidian distance in signal strength between a tracking tag and a reference tag r_j. Let E denotes the location relationship between the reference tags and the tracking tag, i.e., the nearer reference tag to the tracking tag is supposed to have a smaller E value. When there are m reference tags, a tracking tag has its E vector as $\vec{E} = (E_1, E_2, ...E_m)$. The algorithm subsequently finds the unknown tracking tags' nearest neighbors by comparing different E values. Since these E values are only used to reflect the relations of the tags, the reported value of the power level are used to replace the value of signal strength in the equation.

There are three key issues that are examined during the process of locating the unknown tracking tags. The first issue is the placement of the reference tags. Since the unknown tag is ultimately located in a cell surrounded by some reference tags, the layout of reference tags may significantly affect the location accuracy of an algorithm. The second issue is to determine the number of reference tags in a reference cell that are used in obtaining the most accurate approximate coordinate for each unknown tracking tag. For example, the simplest way to find the nearest reference tag to the tracking tag is to use the coordinate of the reference tag with the smallest E value as the unknown tag's coordinate. This is called the 1-nearest neighbor algorithm. This concept can be further extended where one can choose a tracking tag's two nearest neighbors and call it 2-nearest neighbor algorithm. When one uses k nearest reference tags' coordinates to locate an unknown tag, it is called the *k*-nearest neighbor algorithm. The unknown tracking tag' coordinate (x, y) is obtained by:

$$(x, y) = \sum_{i=1}^{k} w_i (x_i, y_i)$$

where w_i is the weighting factor to the i-th neighboring reference tag. The choice of these weighting factors is another design parameter. Giving all k nearest neighbors with the same weight (i.e., $w_i = 1/k$) would make a lot of errors. Thus, the third issue is to determine the optimal weighing factors that should be assigned to different neighbors. Intuitively, w_i should depend on the E value of each reference tag in the cell, i.e., w_i is a function of the E values of k-nearest neighbors.

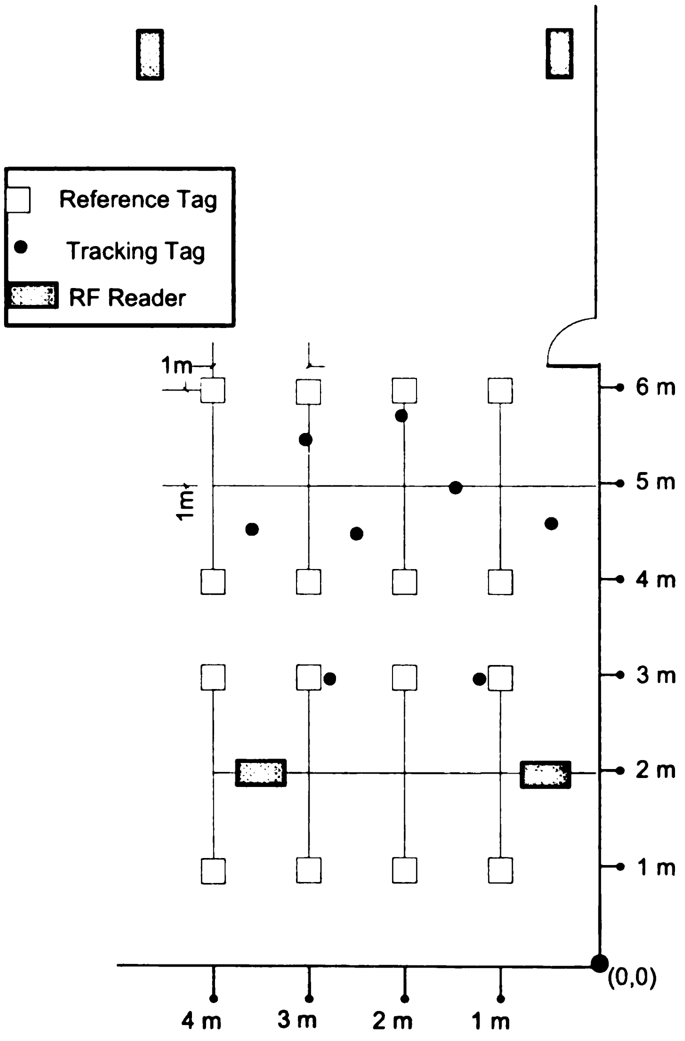


Figure 17: Placements of RF Readers and Tags

Empirically, in LANDMARC, weight is given by:

$$w_j = \frac{\frac{1}{E_i^2}}{\sum_{i=1}^{k} \frac{1}{E_i^2}}$$

This means the reference tag with the smallest *E* value has the largest weight. Note that our approach can be easily extended to a 3-dimensional coordinate.

4.1.3 Experimental Results and Performance Evaluation

A series of experiments were conducted to evaluate performance of the positioning of the LANDMARC System. In the setup, 4 RF readers (n=4) were used and 16 tags (m=16) were used as reference tags. 8 other tags (u=8) were used as objects being tracked, as illustrated in Figure 17. To quantify how well the LANDMARC system performs, the error distance is used as the basis for the accuracy of the system. The location estimation error, e, was defined to be the linear distance between the tracking tag's real coordinates and the computed coordinates, given by:

$$e = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

With the placement of the reference tags and the tracking tags shown in Figure 17 for over 48 hours, we keep collecting data of the power levels from 4 RF readers continuously. Thus we obtain 48 groups of one-hour data. For each of 8 tracking tags per hour, the system computes the coordinates of this tag by using the algorithm discussed in Section 4.1.2. We then compute the location error e for each tracking tag. Thus, we have 48 groups of 8 e values. The location accuracy was examined by analyzing the distribution of these e values under different conditions. Note the experiments were repeated several times to avoid statistical errors.

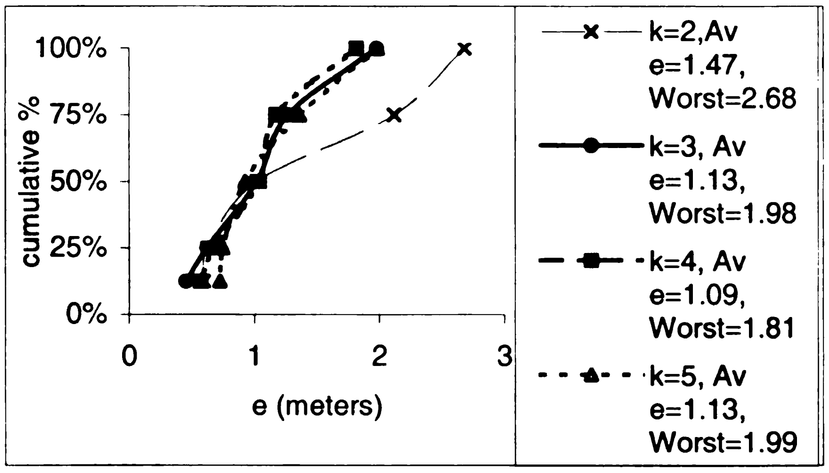


Figure 18: Cumulative percentile of error distance for k from 2 to 5.

4.1.3.1 Effect of the Number of Nearest Neighbors

One of the key issues is to find a best k value in the algorithm. We choose different k values as k=1, 2, 3, 4, and 5 and compute the coordinates of the tracking tags, respectively. Figure 18 shows the results of using different k values in the formula. As shown in Figure 18, k=4 works the best and the positioning accuracy does not improve as the k value further increases. Keeping the same placement, we repeat the process for another 48 hours. Though the positioning error distribution changes, k=4 still gives the best location information. In fact, in all the later experiments (except for a few occasions where k=3 and k=5 worked better), k=4 is the best choice. Hence, we set 4 as the value of k in our formula in the following experiments. Based on the statistics, it can be seen that the 50 percentile has an error distance of around 1 meter while the maximum error distances are less than 2 meters. This is very promising result compared to the RADAR project. In RADAR, the 50 percentile is around 2.37m-2.65m and its 90 percentile is around 5.93m-5.97m.

4.1.3.2 Influence of the Environmental Factors

In order to see how well the LANDMARC approach works in different environments, we collect 10 groups of data from midnight to early morning (little movement) and another 10 groups of data from 10 AM to 3:00 PM (varying level of activities that would result in variations in transmission of the tags). Figure 19 shows the comparison.

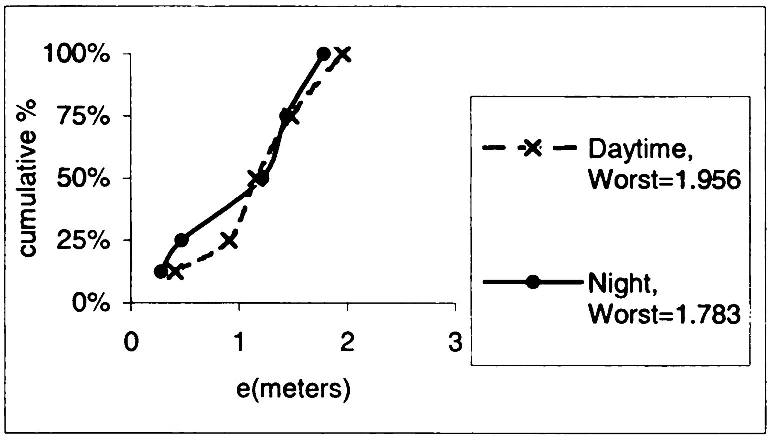


Figure 19: Cumulative percentile of error distance in the daytime and at night.

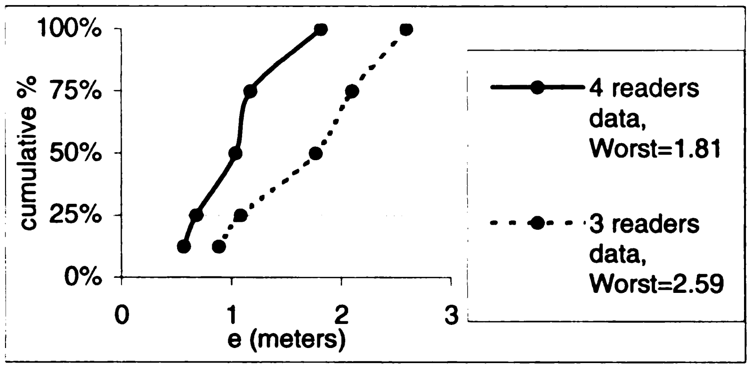


Figure 20: Cumulative percentile of error distance for 3 and 4 RF readers.

Since the lab is very busy with many people around during the day time, one would expect the interference to be more during the day as compared to night. However, from the results, we do not notice any significant difference in the overall accuracy. This shows that our reference tag approach can successfully offset the dynamics of interference.

4.1.3.3 Effect of the Number of Readers

One of the problems of using RF to locate objects is the inconsistency of the signal strength reception. This can primarily be due to the environment and the device itself. In most cases, the environmental factors always have the most impact on the accuracy and maximum detectable range. These include issues like furniture placement, people's movement, and so on. In addition, non-line of sight (NLOS) is another source of reducing the location sensing accuracy. Although NLOS does not prohibit RF transmission as that of infrared, it does create the multi-path problem, meaning the signal can possibly take different paths to reach the receiver and result in interference among the received signals. To better deal with the problem, we can use more RF readers to improve the accuracy. With more RF readers, a better decision can be made for location sensing because more data can be gathered by having extra readers to do the sensing as shown in Figure 20. However, the RF readers are usually quite expensive. Placing more readers means extra costs for the whole system. Due to budget constraints, we have only four RF readers. Adding more readers may not necessarily increase the accuracy. However, it would have increased the processing overhead.

4.1.4 Future Research

This section presented a prototype indoor location sensing system using active RFID. Although RFID is not designed for indoor location sensing, the proposed LANDMARC approach does show that active RFID is a viable cost-effective candidate for indoor location sensing. In our experiments all reference tags are organized in a grid array. This may explain the reason of using 4 nearest neighbors. The influence of having other shapes of reference tags to the selection of the number of nearest neighbors will be investigated. Our methodology can easily be applied to 3D coordinates. The accuracy of the 3D case will also be studied.

4.2 Bluebot

The Bluebot system [75] was designed and implemented at IBM T. J. Watson Research Center (Hawthorne, New York) as part of an internship project. The system combines RFID technology and off-the-shelf Wi-Fi location positioning for continuous positioning.

4.2.1 Acknowledgment

I would like to extend my thanks to the group at IBM Watson: my mentor, Jonathan Munson, for all the guidance during this project, my manager, David Wood, for all the support and help and Alan Cole for his useful suggestions and comments. I would also like to thank Sastry Duri and Amaresh Rajasekharan at IBM Watson for their assistance and ideas. Also thanks to Intermec for donating a PCMCIA card reader and its software for this project.

4.2.2 Motivation

Consider a typical public library in which each book has its own place on a particular shelf. Often, readers will remove books from perhaps multiple shelves and browse through them until they find the right book. While some readers manage to return the unwanted books to the correct shelf, many of them either leave the books in some corner of the library or worse, place them back in the wrong location. This latter situation is hard to detect and can become a librarian's nightmare. A similar situation exists in many retail stores where the customers can tryout several items before deciding which one to buy. In most cases, the customer never returns the tested item to its correct shelf. Some kind of automated tracking mechanism is required. The problem of asset tracking is not restricted to libraries or small retail stores however. Many companies are realizing the importance of increasing the visibility within their supply chain. Asset tracking – knowing what you have and where it is located – is essential for the smooth operation of many enterprises. It also helps big retailers (e.g., Wal-Mart) identify bottlenecks in their supply chain, reduce overstocking or locate spoiled cargo. Government and military organizations are on the lookout for affordable and more efficient ways to track their assets and equipment. Automatic location sensing is the key to realizing these efficiencies and knowledge. One of the most well-know positioning systems is GPS, which relies on satellites to track location. However, due to its dependence on the satellites, GPS lacks the ability to accurately determine location inside buildings. In order to achieve location tracking inside buildings, researchers and industry have proposed several systems, which differ with respect to the technology used, accuracy, coverage, frequency of updates and the cost of installation and maintenance [2, 4, 6, 13, 19, 46, 63, 85, 97]. Steggles and Cadman [86] provide a good comparison of various RF-tag-based location sensing technologies. Many of the current location sensing systems are radio based (e.g., Wi-Fi [1, 5, 19, 43, 65, 85, 97], Bluetooth [1, 3]). By using base station visibility and signal strength or time of flight, it is possible to locate Wi-Fi devices with an accuracy of several meters. In recent years, RFID technology has attracted considerable attention. RFID is emerging as an important technology that is reshaping supply chain management. RFID not only replaces current barcode technology but also provides a greater degree of flexibility in terms of range and access mechanisms. For example, an RFID scanner can read the encoded information even if the tag is concealed (for either aesthetic or security reasons). Several companies (e.g., Wal-Mart, Gillette, CVS) are proposing to use RFID for identifying large lots of goods at the pallet and carton level. Passive tags are preferred for tagging goods, as they are less expensive, longer lived, lighter weight and have a smaller footprint. However since passive tags work without a battery, they have a limited detection range. Current RFID systems are portal-based in which tagged items are scanned either when they enter or leave a facility. This scheme does not provide any information about the exact location of the item once it is moved away from the portal.

4.2.3 Asset Tracking Technologies

Table 3: Tracking Systems Taxonomy

		Continuous in Space	
		YES	NO
Continuous in Time	YES	GPS, TDOA, EOTD, Wi-Fi signal strength, etc.	Simple "presence" technologies (e.g., cellular system where cellID is reported as the cellphone's location)
	NO	***BlueBot***	Fixed Beacon (e.g., EZPass, Bluetooth)

Typically, asset-tracking technologies are geared towards tracking items that individually have high value (e.g., emergency medical equipment or a surgeon). These items require continuous tracking and justify the use of expensive tracking equipment. However, in many tracking applications (e.g. the library scenario described earlier) the object being tracked is either too small or inexpensive to justify the use of a tracking system with high per-item cost. Moreover, many of these applications do not require continuous tracking. We believe there are many applications in which it is valuable to know the precise location of an asset, yet it is sufficient for an asset's location to be updated on a periodic basis – nightly, for example. The BlueBot system is geared towards such applications. We characterize different tracking technologies by distinguishing between continuity in space and continuity in time, as illustrated in Table 1. A technology that provides space-continuous position estimates is able to provide a position estimate that falls anywhere within the space of interest – GPS is an example. A technology that provides time-continuous position estimates is able to provide position estimates at any point in time (intervals between estimates limited only by the performance of the system). Most positioning technologies are continuous in both space and time. An example that is continuous in time but *not* in space is "cell-ID," in which the location of an entity as assumed to be that of the wireless base station with which it is communicating. The location is known to be somewhere in the coverage region of the base station but at no greater resolution than that. An example that is *neither* continuous in space *nor* in time is a typical RFID deployment, such as the EZ-Pass toll collection system, in which the location of a tagged item is known only at the times the item is identified by a reader. BlueBot, which provides position estimates continuous in space but not in time, represents a new point in the taxonomy. As such, it points the way to tracking solutions that provide the precise location estimates needed by some applications, but by sacrificing continuity in time, at a much lower cost.

4.2.4 The BlueBot System

The prototype system combines (passive) RFID technology and a Wi-Fi-based (802.11b) continuous positioning system to enable a periodic asset-locating sweep.

Although, the system uses Wi-Fi based location positioning, it can work with any continuous positioning technology. The prototype system not only identifies but also provides location information of every RFID-tagged item in the sweep space. A portable system (e.g. laptop or PDA) running a Wi-Fi client and connected to an RF reader is mounted on a robot that moves autonomously through the space. As the robot moves, the RF reader continually samples which tags are detectable. At each sample time, the robot's position is obtained from the positioning system. For each detected tag, given the estimate of the robot's current position, knowledge of the reader's physical detection range, and the robot's position estimates at previous detections, an algorithm computes an estimate of the tag's position. In summary, our experiments with the prototype system show that we are able to estimate positions of tagged entities to within 1.5m, given an accuracy of the raw positioning system of about 4m. We experimented with different position estimation algorithms and found that certain algorithms work better than others do when the raw positioning system is capable of giving better accuracy.

In the following section we provide a brief overview of the positioning technology, algorithms and experimental results. However, for a detailed review of the system set up, we refer the reader to our technical report on BlueBot [74].

4.2.4.1 Wi-Fi Positioning System

We use an off-the-shelf Wi-Fi (802.11b) positioning system to track our client system. The positioning system computed the location by using signal strength information (as perceived by the client device being tracked). The positioning engine advertised an average accuracy of 1m (3.5 ft).

4.2.4.1.1 Wi-Fi Calibration

The positioning system we used needs to be calibrated before location scanning can begin. The calibration process establishes an RF-signal strength plot of the area into the positioning system's engine. The calibration process involves drawing a series of straight lines on the area map and recording *sample points* – signal strength at a given location along the line. While at a particular location, the recording device (client device e.g. laptop/PDA wirelessly connected to the positioning engine) is turned around 360° to record the signals from all directions. Several sample points are taken at 3-5m intervals along the lines. The calibration process is repeated until the entire area is covered. In general, a higher number of calibration points give better accuracy. Since the Wi-Fi system that we used was signal strength based, any major change in the environmental conditions in the room (e.g. moving of office partition, metal shelves/cupboards) required re-calibration to ensure maximum accuracy.

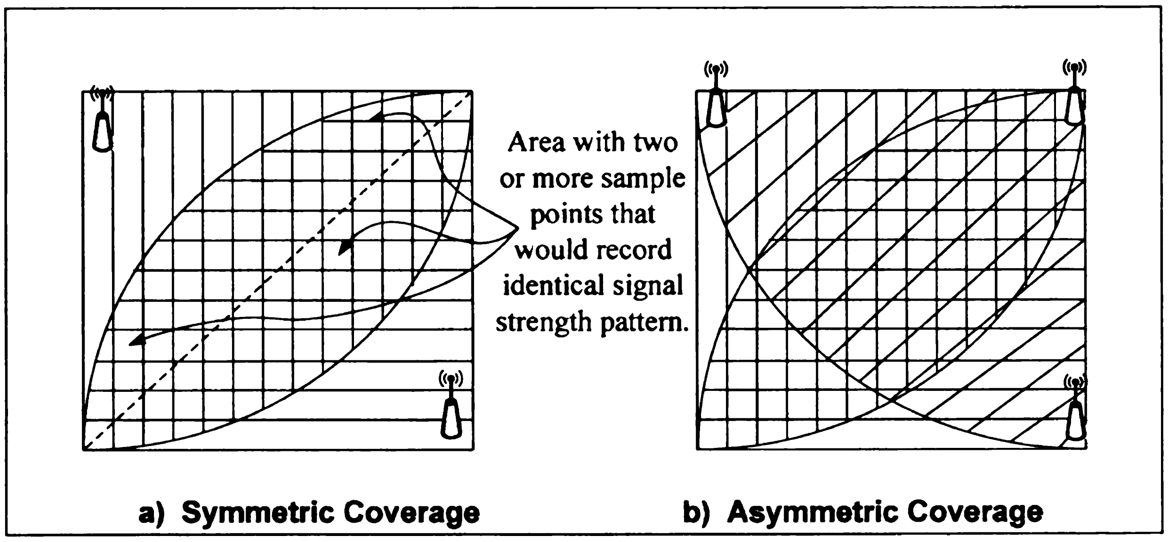

Area with two or more sample points that would record identical signal strength pattern.
a) Symmetric Coverage
b) Asymmetric Coverage

Figure 21: Asymmetric coverage improves positioning system's accuracy.

The calibration process needs to be repeated if more access points are added, removed, or moved from the calibrated area. There are some simple techniques to improve the overall accuracy of the system. One such technique is to avoid symmetric coverage within a room. For example, using two omni directional access points (APs) for coverage in an open room can cause a symmetric RF pattern. As a result, there would be two or more sample points that would record the same signal strength pattern from the two APs (Figure 21a). Using APs with directional antennas or having an asymmetric coverage by introducing a third access point (APs at 3 corners of the room - Figure 21b) can solve this problem.

4.2.4.1.2 Positioning Technology Accuracy Analysis

Because our tag-position estimation algorithms (described later) use characteristics of the positioning technology used to position the robot, we carried out two sets of experiments to examine the performance of our positioning system. The two sets differed in the density of calibration points and the size of the experiment area.

Case A

The experiment area chosen was a large open 12.5m x 13.75m room (Figure 22). The positioning system was calibrated to work with only the three access points that were installed in the room (during the calibration process, the positioning engine was programmed to ignore any other access points installed in the building). The calibration points were spaced at a distance of approximately three meters. The three access points were powered down to run at 15mW. To improve accuracy of the system, we wanted to have the access point coverage such it gives the steepest signal strength gradient across the room. Our Cisco 340 access points had four power levels (100mW, 30mW, 15mW and 5mW). Using the positioning system's software, we found that 5mW power level did not give complete coverage in the room, which resulted in a higher error. We selected 15mW which seemed to give the best gradient.

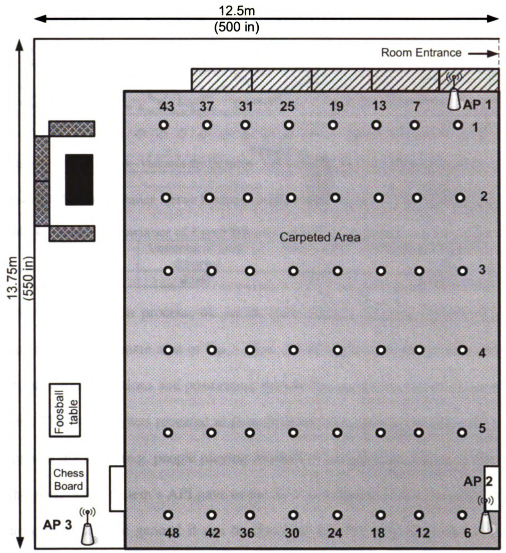


Figure 22: Experiment setup (case A)

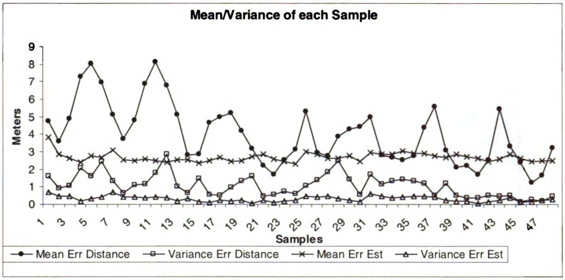


Figure 23: Mean/Variance Error Distance and Error Estimate for each sample point.

Table 4: Mean and Variance of Error Distance and Error Estimate for the entire set.

Mean error distance	Variance in error distance	Mean error estimate	Variance in error estimate
4.001	2.047	2.681	0.421

After the calibration process, 48 points (refer Figure 22) were chosen as sample locations in an approximate area of 9m x 12m. An 801.11b client device was moved to each of the sample locations and positioning system was queried to report the position of the client. This process was repeated at four different times during the day with varying conditions in the room (e.g. people playing foosball or having a discussion on the couch, etc). The positioning system's API gave us the X, Y coordinates of the client and an error estimate *e* in meters. In general it can be observed that the error distance is more for points near the wall (e.g. sample points 1-6 and multiples of 6 – 6, 12, 18). There is no definite pattern for the error estimate values. Figure 23 gives the mean and variance of the error distance and error estimate for each of the sample points. Table 4 shows the mean (and variance) error distance (the positioning system's expected error) for this setup. Our experiments showed that the average positioning error (for the positioning system) was between 3.5-4 meters for this setup.

Case B

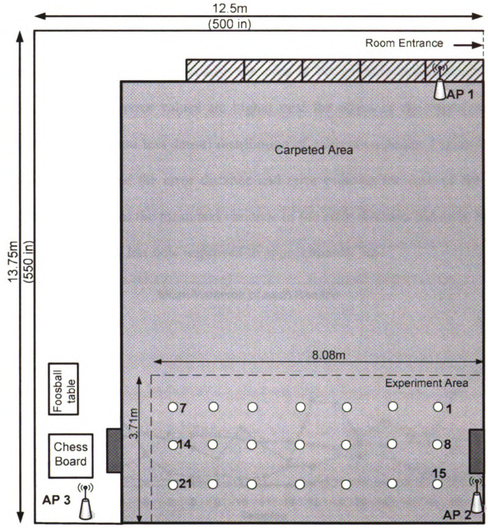


Figure 24: Experiment setup (case B).

In our next set of tests, the positioning engine was calibrated for a smaller area (8.08m x 3.71m) within the room (see Figure 24). Unlike case A, the calibration points in this experiment were closely spaced (1m). The rest of the calibration setup (position of access points and their power level) was the same as case A. After the calibration process, 21 points (refer Figure 24) were chosen as sample points. We followed the same procedure as described in case A – the Wi-Fi client device was moved to each of the sample points and positioning engine was queried to report the location of the client (the process was repeated at four different times during the day with varying conditions in the room). The error distance and the error estimate values are much lower in this case (compared to case A) probably because the area is better-calibrated (densely spaced calibration points). The error values are higher near the edges of the experiment area where there is uneven (and less dense) distribution of calibration points. Figure 25 gives the mean and variance of the error distance and error estimate for each of the sample points while Table 5 gives the mean and variance of the error distance and error estimate. The positioning accuracy has now improved to approximately 3m.

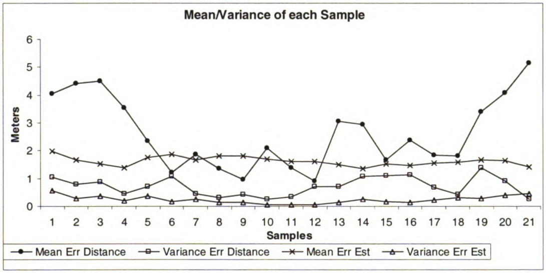


Figure 25: Mean/Variance Error Distance and Error Estimate for each sample point.

Table 5: Mean and Variance of Error Distance and Error Estimate for the entire set.

Mean error distance	Variance in error distance	Mean error estimate	Variance in error estimate
2.619	1.386	1.624	0.280

4.2.4.2 RFID Setup

We used the Intermec [7] PCMCIA RFID reader (Figure 26) in our prototype system. The reader operates at a frequency of 915MHz and can be connected to any device that has a PCMCIA port. For our experiments, we connected the reader to a laptop. In time, we expect a smaller device, such as a PDA, or a custom-built PC, would be used for additional portability.

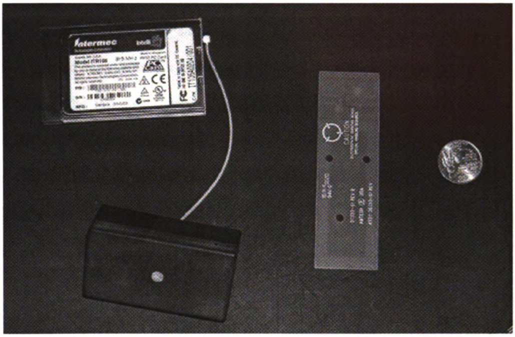


Figure 26: Intermec's PCMCIA reader and passive tag.

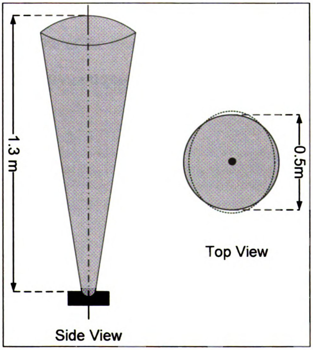


Figure 27: RF characteristic of reader antenna.

The reader is attached to a directional antenna, which has a limited range of about 1.6m (5 ft.). The reader's antenna is placed facing the ceiling such that its maximum gain is in the vertical direction. The RF characteristics of the reader approximate a vertical cone with its vertex on the reader's antenna. We observed that tags at a height of 1.3m (4 ft.) were detected with 90% (or better) probability when the reader was horizontally within 0.5m of the tag (see Figure 27). Due to the rectangular base of the reader's antenna, the RF characteristics had a slight elliptical shape. During initial tests, we found that the eccentricity was close to 1. For all our experiments, we have considered the reader's RF characteristics to be circular.

4.2.5 BlueBot Setup

For our experiments, we used the Roomba Robotic Floorvac [11] as the robot that moves autonomously in the sweep space. The Roomba uses intelligent navigation technology to automatically move around the room without any human direction. The Roomba expects to cover 90% of the room. A server machine running the positioning engine (PE) tracks our client device (laptop/PDA) that sits on the robot. Figure 28 depicts the BlueBot setup.

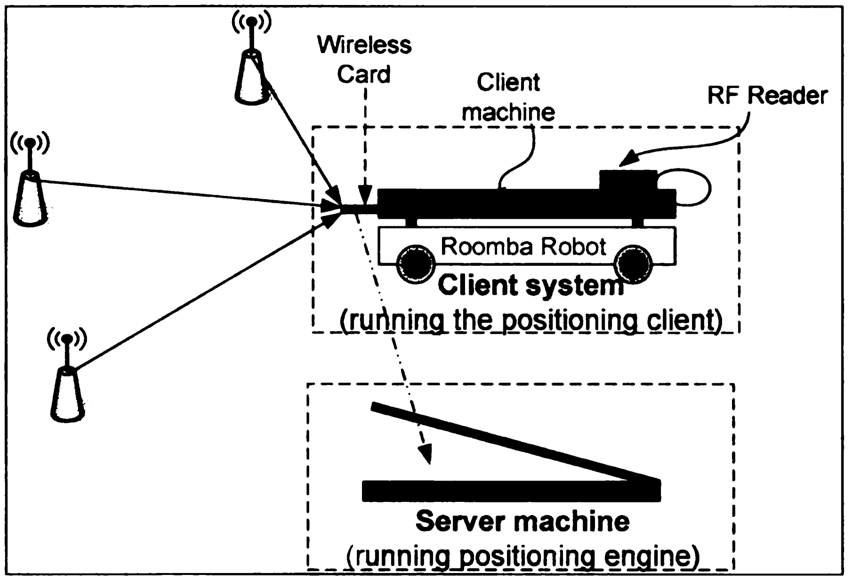


Figure 28: BlueBot Setup

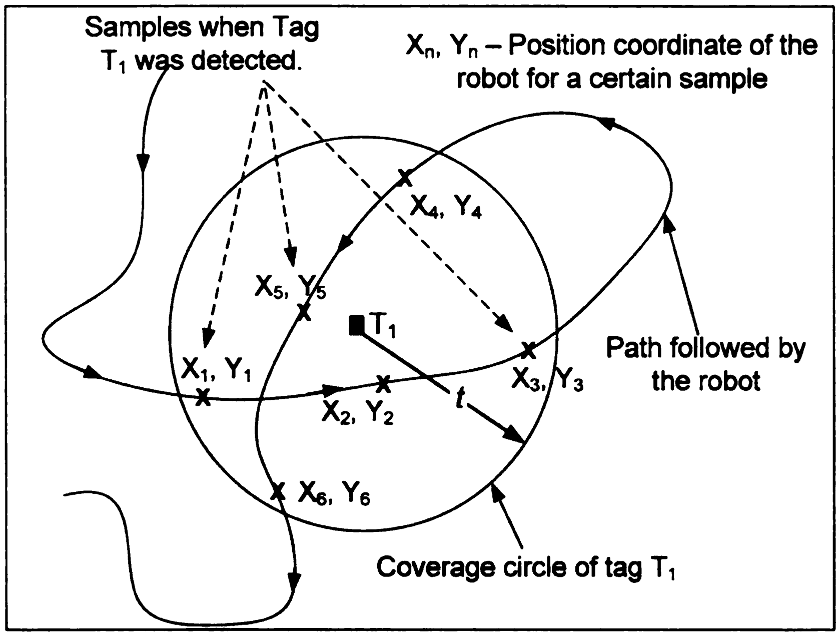
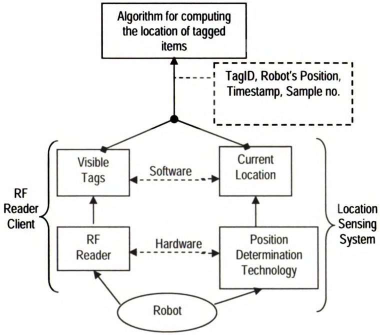

Samples when Tag T1 was detected.
Xn, Yn – Position coordinate of the robot for a certain sample
X4, Y4
X5, Y5
T1
X1, Y1
X2, Y2
X3, Y3
t
Path followed by the robot
X6, Y6
Coverage circle of tag T1

Figure 29: Samples for a particular tag during robot's random walk.

The PE is calibrated to work with the access points placed in the corners of the experiment room. The RFID reader is connected to the client device and records all the tags that it detects as the Roomba moves about the room. In our experiments, the tagged items were placed at a height of approximately 1.3m (4ft) from the floor. Whenever the reader detects a tag, the client machine sends a message containing the tag's id to the server. The server then notes the current position of the client and associates it with the detected tag. Our algorithms combine this sample with the previous samples to refine the position of the tagged item over time. As seen from Figure 29, a tag will be sampled only when the RF-reader enters its coverage area. Figure 30 gives a logical flow chart of the system. Due to the random movement of the robot, consecutive samples for the same tag might not be equally spaced in time. Experimental results in discussed later confirm this.



Algorithm for computing the location of tagged items
TagID, Robot's Position, Timestamp, Sample no.
Visible Tags
Software
Current Location
RF Reader Client
RF Reader
Hardware
Position Determination Technology
Location Sensing System
Robot

Figure 30: Logical flowchart of the system.

The use of a robot like Roomba may not be appropriate in all environments. For example, in a large warehouse, where items are stacked on high shelves, we suggest the use of a forklift with an RFID reader mounted on it. When the forklift moves a product, it can record the product in transit and the new bay where it is dropped. Similarly, in a large car lot, a patrol car can have a reader, which records the cars as it passes by parked cars.

In general, the idea is to mount the readers on existing infrastructure that moves around in the environment as part of its existing functionality.

4.2.5.1 Algorithms

The Wi-Fi positioning system we used reported the X, Y coordinates and an error estimate *ee* in meters [74]. We have seen before that the RF characteristics of the reader are in the form of a cone expanding outwards in the vertical direction. With the tags placed at 1.3m (4ft) from the ground, the reader's detection circle was determined to have a radius (*r*) close to 0.25m. A circle drawn with center at (X, Y) and radius (R) of *ee+r* (Figure 31) will include the tag being tracked. We call this circle the *confidence circle*.

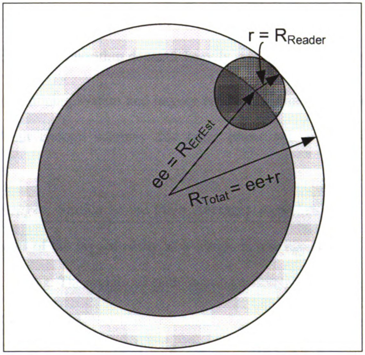


Figure 31: Total radius is the sum of the reader's coverage and the uncertainty circle.

We define the following:

t – represents the tag with id t

N_t – is the total number of samples for a given tag, t.

(X_{ti}, Y_{ti}) – is the location estimate for sample i of tag t

ee_{ti} – is the error estimate for the positioning report for sample i of tag t

r –is a constant representing the read radius of the RFID reader.

$R_{ti} = ee_{ti} + r$ - is the radius of the confidence circle for sample i for tag t.

$C[(X_{ti}, Y_{ti}), R_{ti}]$ - is the confidence circle for sample i of tag t.

With these in mind, we provide four algorithms to compute the location.

i) Plain Averages: This most simple algorithm computes the location coordinates of the tagged entity as the statistical average of the reader's location when it detected the entity.

$$(X_t, Y_t) = [\, \sum (X_{ti}, Y_{ti}) \,] / N_t$$

The accuracy of this algorithm depends heavily on the estimated position (X_i, Y_i) as reported by the positioning system and largely on the distribution of samples around the tagged entity. With enough samples this latter effect will be averaged out of the computation.

ii) Weighted Averages: Similar to the Plain Averages algorithm, here we compute the location coordinates of the tagged entity as a weighted average of the reader's locations when it detected the tag. The weight of each location estimate is inversely proportional to the square of the error radius.

$$(X_t, Y_t) = [\, \sum \{ 1/ee_i^2 * (X_i, Y_i) \} \,] / (\sum 1/ee_i^2)$$

The positioning system's error estimates with a smaller error radius tend to be closer to the tag. Therefore, by using $1/ee_i^2$, the algorithm is able to give higher weight to sample points that are closer to the tag. The accuracy of this algorithm is similar to Plain Averages algorithm although, we expect earlier convergence to the actual location.

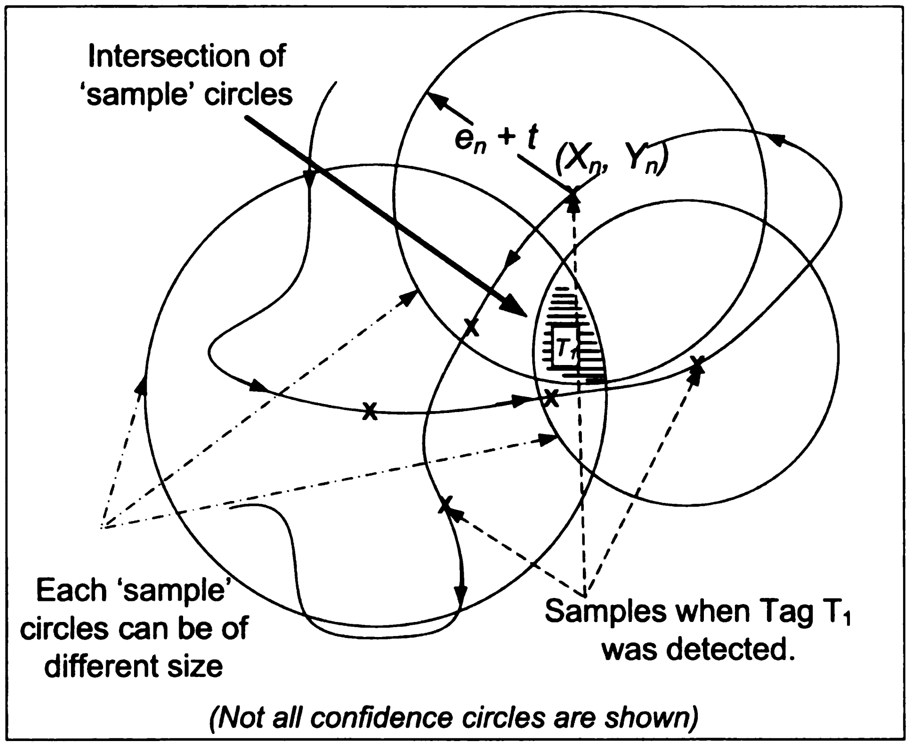

Intersection of 'sample' circles
$e_n + t$
(X_n, Y_n)
T_1
Each 'sample' circles can be of different size
Samples when Tag T_1 was detected.
(Not all confidence circles are shown)

Figure 32: Intersection of the different 'sample' circles converges to the tag location.

iii) Intersection Algorithm: Intersection of several confidence circles provides a finer estimate of a tag's position. We represent the tag's location as the centroid of the bounding box of this intersection area. As the number of samples increases, the intersection area decreases (Figure 32), thus improving the accuracy of the tag's expected location. The expected location is expressed as:

$$(X_t, Y_t) = Centroid\{\, BoundingBox\,(\, C[(X_{t1}, Y_{t1}), R_1] \cap C[(X_{t2}, Y_{t2}), R_{t2}] \cap \ldots \cap C[(X_{Nt}, Y_{Nt}), R_{Nt}])\,\}$$

The precision of this algorithm is inversely proportional to the size of the intersection region. Smaller intersections imply higher confidence in the expected location.

iv) Min-Max Algorithm: This algorithm is similar to the intersection algorithm but has a more mathematical approach. We compute the coordinate as follows:

$$X = [\ min\{X_i + ee_i\} + max\{X_i - ee_i\}\ ] / 2$$

$$Y = [\ min\{Y_i + ee_i\} + max\{Y_i - ee_i\}\ ] / 2$$

4.2.6 BlueBot Performance

We used a large open 12.5m x 13.75m room (Figure 33) to carry out a series of experiments to examine the performance of our prototype system.

4.2.6.1 Experiment Set A

In the first round of experiments, the positioning engine was calibrated for the entire room as described in Section 4.2.1 of [74]. For this setup (as seen in Table 2 of [74]), the positioning system's accuracy was around 4m. The tag placement for this set of experiments is as shown in Figure 33. We recorded the performance of the four algorithms for this setup for four different runs of the experiment. Figure 35 shows the performance for one such run. It is easy to see that the positioning accuracy of the four algorithms greatly varies between tags. Part of the reason is due to the large variation in the number of detections for each tag.

In our experiments, we started the Roomba from the center of the experiment area. However, we noticed that since it moved in a random fashion, the number of 'detection' samples for each tag is not a uniform distribution. We expect that the averaging effect (from several runs of the experiment) to smooth out the large variation. Figure 34 shows the time distribution between consecutive detections. Since the number of samples per tag per run is different, the values (per run) have been normalized to be on the same scale. We

and the a

Th

more

the pe

of the

state

coord

scale. We define the term *error distance* as the difference between the estimated location and the actual location of the tagged item.

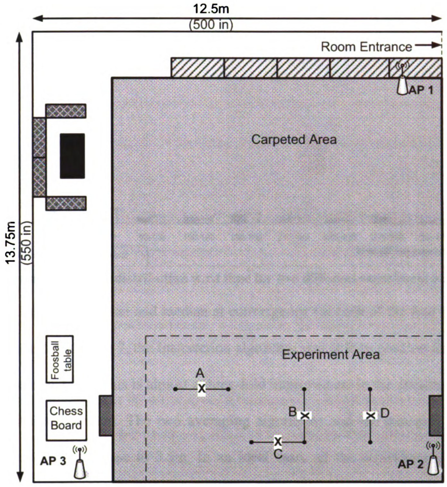


Figure 33: BlueBot setup with non-uniform placement of tags.

The four positioning algorithms start with a large error distance and as they acquire more samples, they slowly converge to the actual location of the tag. In order to quantify the performance of our system, we define a convergence point for each tag (in each run of the experiment). The convergence point can be thought of as the start of the steady state for a tag. Beyond this point, there is no significant change in the computed coordinates for that tag. We use convergence values to find the accuracy of our system.

Table 6 shows the average time and the average number of samples at which the algorithms converged for each run of the experiment.

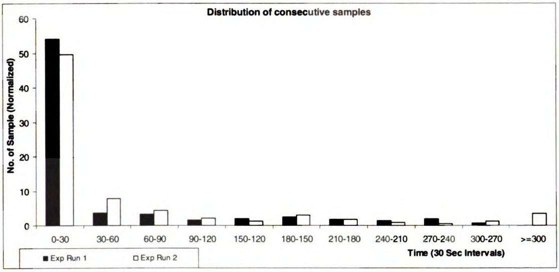


Figure 34: Sample distribution w.r.t time for two different experiment runs

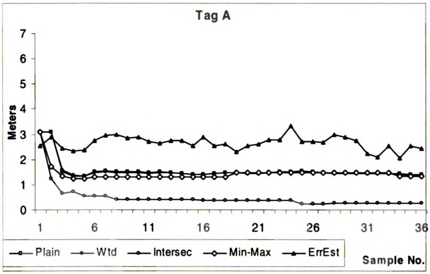
Table 7 shows the mean and median at convergence for each of the four algorithms. As we can see from Table 7, the Intersection algorithm was able to position the tags with accuracy close to 1.5m. This is almost a three-fold improvement in the accuracy provided by the positioning system. The two averaging algorithms and the min-max algorithm, gave an average error close to 3.5m. In an ideal case, all the algorithms should have converged to the same value. In the following discuss, we try to investigate the reason why the intersection algorithm did not converge to the same error distance as the other algorithms.

Table 6: No. of samples and time (sec) at convergence

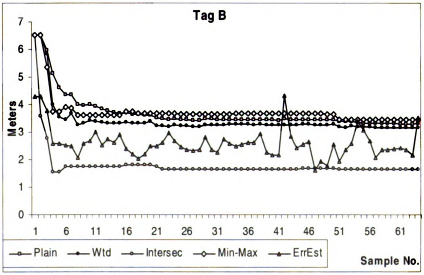
Run No.	No. of samples at Convergence			Time at Convergence (sec)		
	Mean	Median	Std Dev	Mean	Median	Std Dev
1	21.5	21	2.645	1623.75	1397	540.95
2	13.25	14	3.774	1178.25	1169.5	448.973
3	24.25	23	8.845	1706	1553.5	442.621
4	12.75	11.5	6.238	916.75	879	261.515
Average	17.9375	17.375	5.3755	1356.188	1249.75	423.514

Table 7: Error distance at convergence for each algorithm

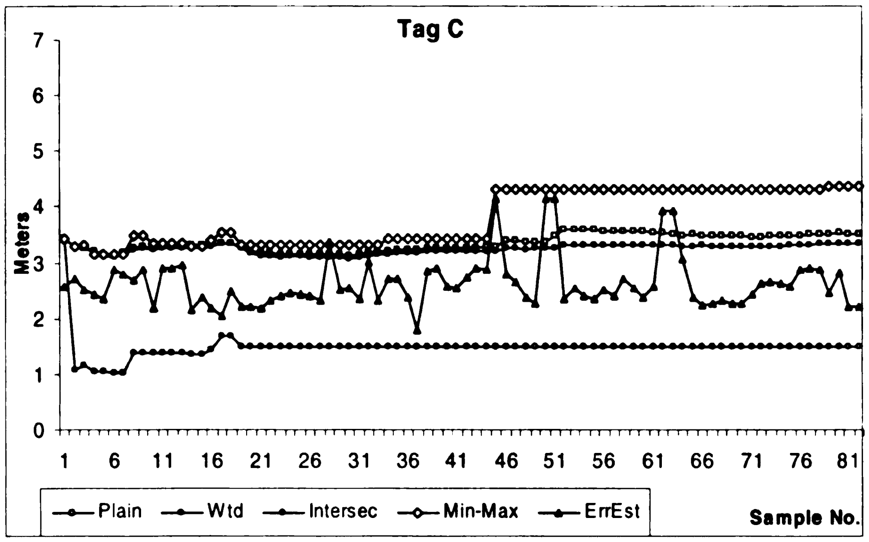
Run No.	Plain Averages Algorithm			Weighted Averages Algorithm			Min-Max Algorithm			Intersection Algorithm		
	Mean	Median	Std Dev	Mean	Median	Std Dev	Mean	Median	Std Dev	Mean	Median	Std Dev
1	3.214	3.369	1.314	3.135	3.277	1.255	3.440	3.486	1.582	1.521	1.564	0.905
2	3.276	3.220	1.055	3.180	3.094	1.111	3.739	3.533	1.346	1.727	1.683	0.864
3	3.376	3.558	1.460	3.287	3.427	1.318	3.119	2.741	2.028	1.973	1.958	1.177
4	3.139	3.066	0.767	3.028	3.072	0.747	2.997	3.064	0.315	1.289	1.058	0.577
AVG	3.251	3.303	1.149	3.158	3.218	1.108	3.324	3.206	1.318	1.628	1.566	0.881



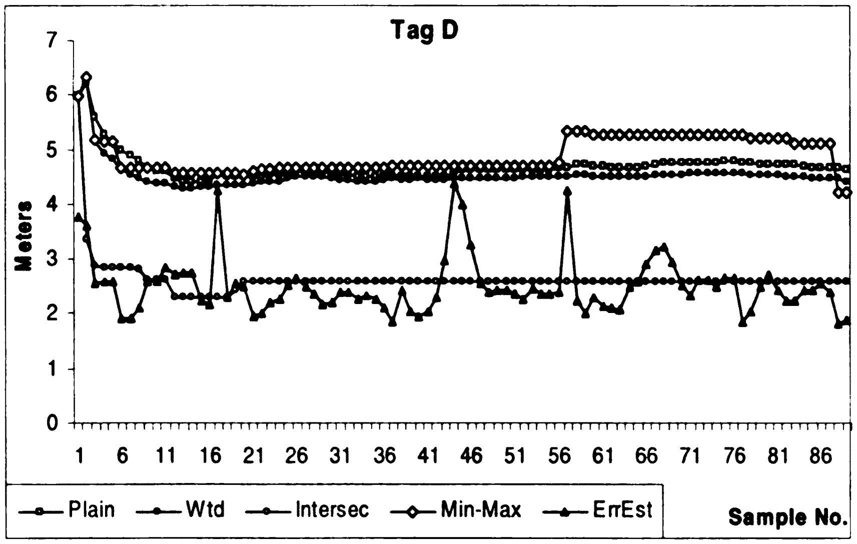
a)



b)



c)



d)

Figure 35: Error distance convergence w.r.t sample no. for each tag

4.2.6.2 Statistical vs Non-Statistical Algorithm

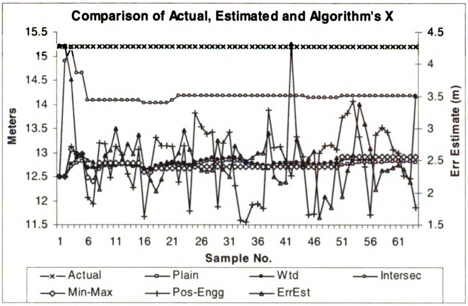
The intersection algorithm is based on certain assumptions:

1. The reader's detection zone is an unchanging cone in the vertical direction. This assumption is true only in the ideal case. RF signals are characterized by constantly varying pattern that strongly depend on environmental conditions.

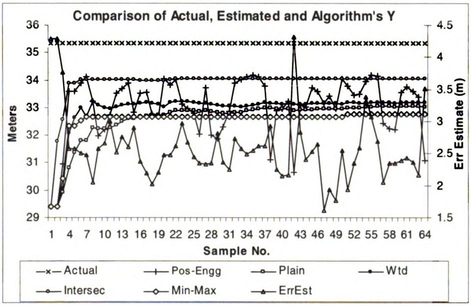
2. The positioning engine's error distance estimate is accurate and uniformly distribute (circle) around the estimated location. This assumption may not hold true all the time.

It seems that given the stochastic nature of the data, statistics-based algorithms are the best choice. As an example, we look at Figure 36, which shows the performance of the Wi-Fi positioning system and each of the four algorithms at every sample (for experiment set A tag B – Figure 35b). The two averaging algorithms depend more on the positioning system's accuracy and hence, their performance is close to the positioning system's accuracy, which from section 4.2.1 of [74], was close to 4m. Referring to Figure 36a, one would expect a uniform distribution of the estimate (from the positioning engine) around the actual X=15.2m, however that was not the case. The positioning engine was observed to have an offset close to 3.5m in the estimated X (For example, when X=15.2m, the estimated X from positioning engine varied from 11.5m to 13m). This offset, while unfortunate is an artifact that must be accommodated if accuracies greater than that provided by the position determination technology are desired.

For the averaging algorithms to converge to the actual location, the offset from the positioning engines needs to be uniformly spread around the actual location. In this case, the offset was always below; hence the large error. The intersection algorithm on the other hand is based on the intersection of several *confidence circles*. The intersection algorithm gives an area where the tracked entity is located. To better represent the result of the algorithm, the center of the bounding box around the intersection area is reported as the computed location.



a) X-coordinate



b) Y-Coordinate

Figure 36: Computed, Estimated and Actual Coordinates

Another factor, which we feel, might have contributed to the difference in the error distance for the intersection algorithm is the implementation for computing the center of the bounding box. We implemented used the Java AWT class however, we found that Java makes some approximation which might not give the actual value. This might explain why the Min-Max algorithm and the Intersection algorithm did not converge to the same value.

4.2.6.3 Experiment Set B

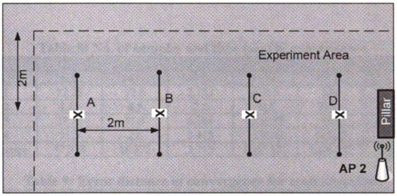


Figure 37: Uniform placement of tags.

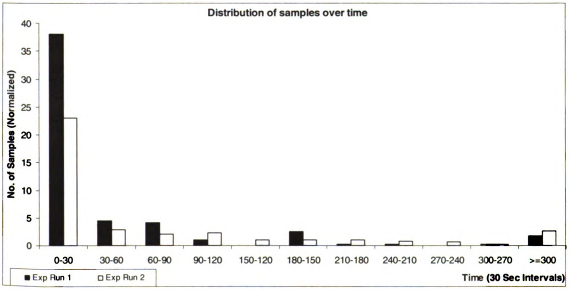


Figure 38: Sample distribution w.r.t time

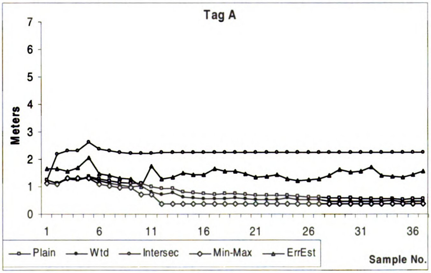
The calibration setup for our second round of experiments was same as that for Case B as described in section 4.2.2 of [74]. The positioning engine was calibrated to work with the smaller experiment area and the calibration points were more closely spaced. As seen in section 4.2.2 of [74], this setup gave accuracies close to 2.5m (as compared to 4m in the previous configuration). Figure 37 shows the placement of the tags. In this scenario, the positioning technology gave better accuracy and less variation in the error estimate. As a result, our averaging algorithms performed better, giving us accuracies close to 1m (Table 9). It was observed that in most cases, the averaging algorithms surpassed the performance of intersection algorithm by at least 0.5m.

Table 8: No. of samples and time (sec) at convergence

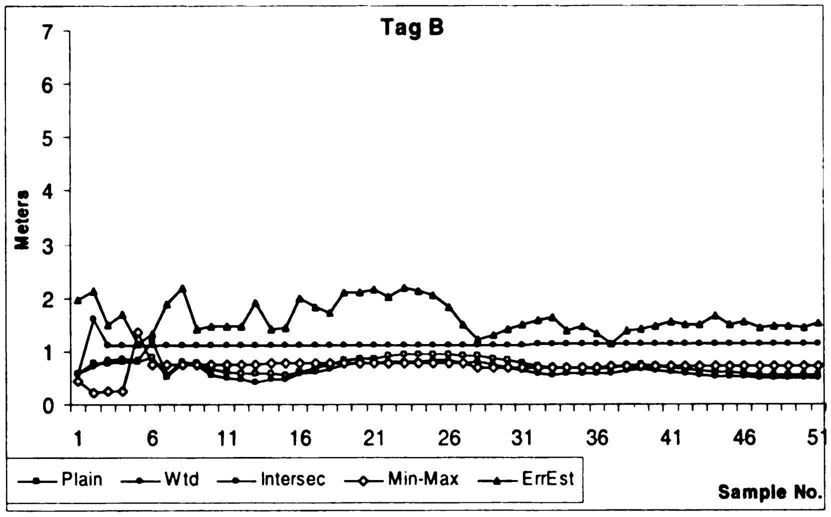
Run No.	No. of samples at Convergence			Time at Convergence (sec)		
	Mean	Median	Std Dev	Mean	Median	Std Dev
1	12.25	12.5	2.753	678.5	580	285.487
2	8.25	8.5	2.5	716	715.5	134.806
3	29.75	31.5	6.184	1335.75	1358.5	679.237
4	21.5	21	3.415	1538.25	1651.5	498.893
Average	17.938	18.375	3.713	1067.125	1076.375	399.606

Table 9: Error distance at convergence for each algorithm

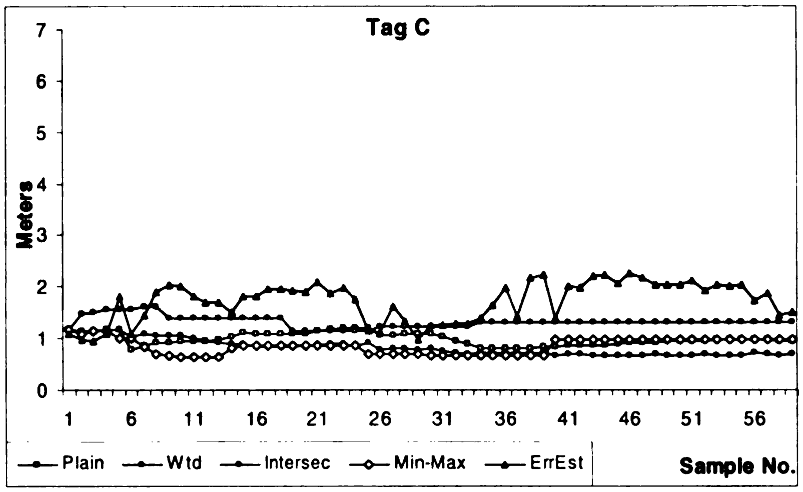
Run No.	Plain Averages Algorithm			Weighted Averages Algorithm			Min-Max Algorithm			Intersection Algorithm		
	Mean	Median	Std Dev	Mean	Median	Std Dev	Mean	Median	Std Dev	Mean	Median	Std Dev
1	1.465	1.315	0.791	1.631	1.643	0.794	1.517	1.361	1.107	3.057	2.682	1.937
2	0.863	0.908	0.609	1.116	1.031	0.380	1.159	1.091	0.488	2.005	2.024	0.660
3	1.019	0.820	0.572	0.821	0.652	0.466	1.079	0.683	1.013	1.266	1.18	0.738
4	1.473	1.045	1.392	1.649	1.072	1.549	1.374	0.929	1.145	2.486	2.14	1.282
AVG	1.205	1.022	0.841	1.304	1.100	0.797	1.282	1.016	0.938	2.204	2.008	1.154



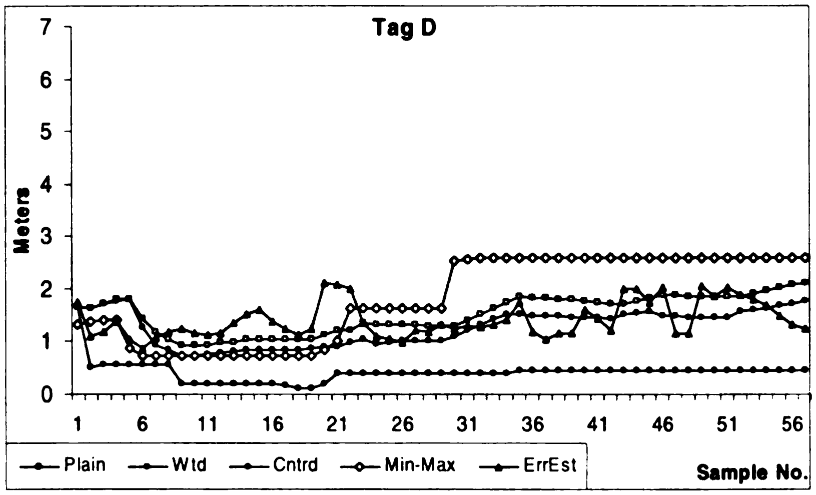
a)



b)



c)



d)

Figure 39: Error distance convergence w.r.t to sample no. for each tag

4.2.7 Future Research

In our current system, there is a high variation in the amount of time and number of samples required to converge to a certain level of accuracy. In the future, we are planning to reduce this variation by using a robot that can be controlled to move in a directed pattern. We have considered employing a feedback system so that the direction of the robot is controlled by the RF system. A dual-variable gain antenna system can be used such that the high gain antenna controls the movement of the robot until its low gain partner sees the tag being hunted. We noticed that signal strength based Wi-Fi systems are easily affected by changes in the surroundings. In the future we plan to make use of systems that are immune to environmental changes (e.g. Time-of-flight, Time-of-arrival and Angle-of-arrival based positioning systems) and analyze the performance of our BlueBot system. Another approach would be to continue to use Wi-Fi signal strength, but add reference RFID tags through the environment that could be used to remove any positional offsets that might be present relative to the original calibration. We are also looking at various ways to make 3D positioning possible.

4.3 Bluetooth for Location Sensing

Location-aware computing is regarded as a key feature of many future mobile applications. GPS serves well for most outdoor applications; however, its dependence on satellites makes it ineffective in indoor environments. Several technologies such as infrared sensing, radio frequency, ultrasonic and RFIDs have been proposed for indoor location sensing. Each of these methods has its own merits and shortcomings. Recently there has been an increase in the use of commodity wireless technologies like 802.11 or Bluetooth for indoor location positioning. This section presents two techniques of how

Blueto

inform

the B

(Lan

syst

4.3.

info

trac

the

as c

mus

techi

an e

prop

info

in t

Blu

ind

de

ub

be

L

Bluetooth technology can be used (with and without the use of signal strength information) for location sensing. The section concludes by suggesting some changes to the Bluetooth architecture to improve its capabilities for location positioning. WEBOTS (Lambert [56]) is an example of an implementation of a Bluetooth based location sensing system, similar to the one presented here.

4.3.1 Research Background

The location of people and objects relative to their environment is a crucial piece of information. Indoor location sensing has already found many applications other than asset tracking and security. The more traditional applications might be as simple as tracking the location of a valuable shipping carton or detecting the theft of a laptop computer, or as complex as helping someone to find his or her way around an unfamiliar building (e.g., museums or art galleries). Hospitals and day care centers have started using positioning technologies to locate personnel or the nearest doctor (or medical equipment) in case of an emergency [10, 12]. Several researchers in the area of ad hoc wireless networks have proposed to improve the efficiency of MANET routing algorithms by using location information of member nodes [25, 61, 73, 88, 95]. Last few years have seen an increase in the use of commodity wireless technologies like WLAN [1, 5, 19, 43, 65, 85, 97] and Bluetooth (e.g., Bluetags Corporation and AeroScout, formally known as Bluesoft) for indoor location sensing. It is interesting to note that both these technologies were designed without location sensing in mind. These two technologies are fast becoming ubiquitous in office and home environments; thus requiring no additional infrastructure to be in place for indoor location sensing. The Bluetooth SIG is also working on a new Local Positioning (LP) Profile (version 0.95 as of this writing) [9]. This profile specification defines a mechanism and format for the transfer of position related data over Bluetooth. The profile also supports position determination and location awareness. Bluetooth and 802.11b use the same 2.4GHz unlicensed frequency band; as a result, there has been a lot of concerns about interference between them [14, 27, 41, 83]. In case of indoor location positioning, it is estimated that on an average, the probability of a Wi-Fi transmission colliding with a simultaneous Bluetooth transmission is around 55% [14]. Therefore, it is an important issue to examine the frequency conflict between two technologies indoor environments. [69, 71] presents experimental results of the performance of Bluetooth and 802.11b in presence of interference from the other technology in order to examine which technology is better suited for doing indoor location sensing. From the experiments described in [69, 71], we can conclude:

- As the number of 802.11b devices operating over the same channel increases, the interference between them increases.
- There is significant interference between Bluetooth and 802.11b devices. The interference is higher when the devices are close to each other. Beyond the range of Bluetooth, it drops down significantly. In addition, the effect of Bluetooth is more on 802.11b than the reverse case.
- There is very little interference between two Bluetooth devices place in close vicinity.

In a location-sensing environment, we would need to have several devices of the same type operating on the same network. Hence, it would be important that these devices have very little interference between them. With the above results, it can be seen that Bluetooth can satisfy these requirements and hence can be a good candidate for use in indoor location sensing. Since Bluetooth is becoming a standard feature in most hand-held devices, it would eliminate the need for the user to carry additional sensory devices, as needed by several location-sensing technologies of earlier times. A Bluetooth piconet can have 7 active slaves and up to 200 inactive devices in parked mode. For location sensing applications, it is enough to 'see' another Bluetooth device. In addition, it doesn't matter if the other device is a master or a slave. The device should be within the range to be detected. This implies that a Bluetooth sensor would be able to detect up to 200 other Bluetooth devices.

Signal strength varies randomly with time. By using the concept of reference tags, a real-time system can be designed. The signal strength variation in such a system will affect the tracking and the reference tag in the same manner. With the advent of Bluetooth ASIC chips, the price of Bluetooth devices is expected to drop down significantly in the near future and the size of a Bluetooth sensor would be very small. Thus, using Bluetooth reference tags would be a feasible idea. We consider two possibilities – location sensing when signal strength information is not made available and location sensing when that information is available. As of this writing, none of the Bluetooth devices available in the market make signal strength information available. Making it available is optional according to the Bluetooth specification.

4.3.2 Location Sensing Without Signal Strength Information

Similar to RFIDs [86], every Bluetooth device has a 48-bit address. We can use Bluetooth tags as reference tags, which can be uniformly spread, in the area of interest. The Bluetooth sensors can be placed at 5m each so that they overlap in their range. Since the reference tags are positioned in-between the sensors and the sensor range overlap, each reference tag would be seen by more than one sensor. Figure 40 shows such a setup.

We define a table whose rows represent individual tags and the columns represent every reader. We can follow this approach for an unknown device whose location is to be determined. By comparing the row of an unknown device with those for known devices, we can predict where the device might be present. Table 10 shows an example.

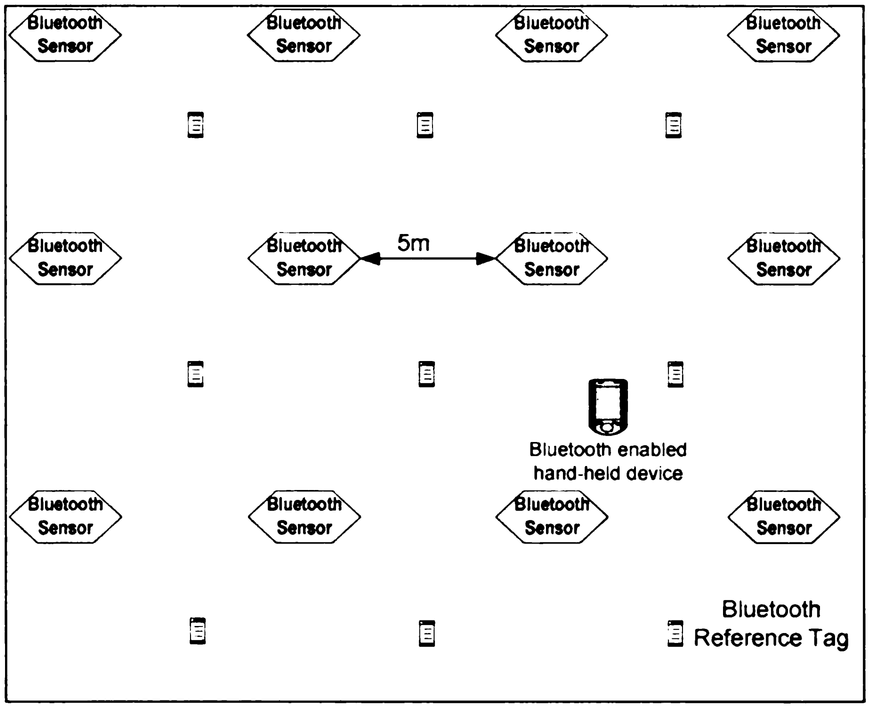


Figure 40: Bluetooth location sensing using the concept of reference tags

Table 10: Locating unknown tag using reference tag

Reader	1	2	3	4
Tag – 1	0	1	1	0
Tag – 2	1	1	1	0
Tag – 3	0	0	1	1
Unknown – 1	0	0	1	1
Unknown – 2	1	0	1	0

We can see from the table that the Unknown-1 device's row information matches with that of Tag 3, which means that they are very close to each other (if not at the same place). The row information for Unknown – 2 closely matches with that for reference tag 2. Hence, it must be located somewhere near Ref Tag 2. The accuracy of this approach depends on how tight the reference tags are placed, the geometric placement of readers and reference tags and the amount of overlap between neighboring readers.

4.3.3 Location Sensing If Signal Strength Information Available

If in the future Bluetooth products meant for location sensing could give signal strength information then, a real-time system can be implemented. Referring to Figure 41, we do the following analysis. Assume there are *n* Bluetooth readers and *m* reference tags. We define the Range Vector of an unknown tag as:

$$u = (u_1, u_2, \ldots\ldots, u_n)$$

where u_i denotes the signal strength value of the unknown tag perceived on Bluetooth reader *i*, $i \in (1, n)$. For the reference tags, each one also has its range vector as:

$$r_i = (r_{i,1}, r_{i,2} \ldots\ldots r_{i,n}) \text{ where } i \in (1, m).$$

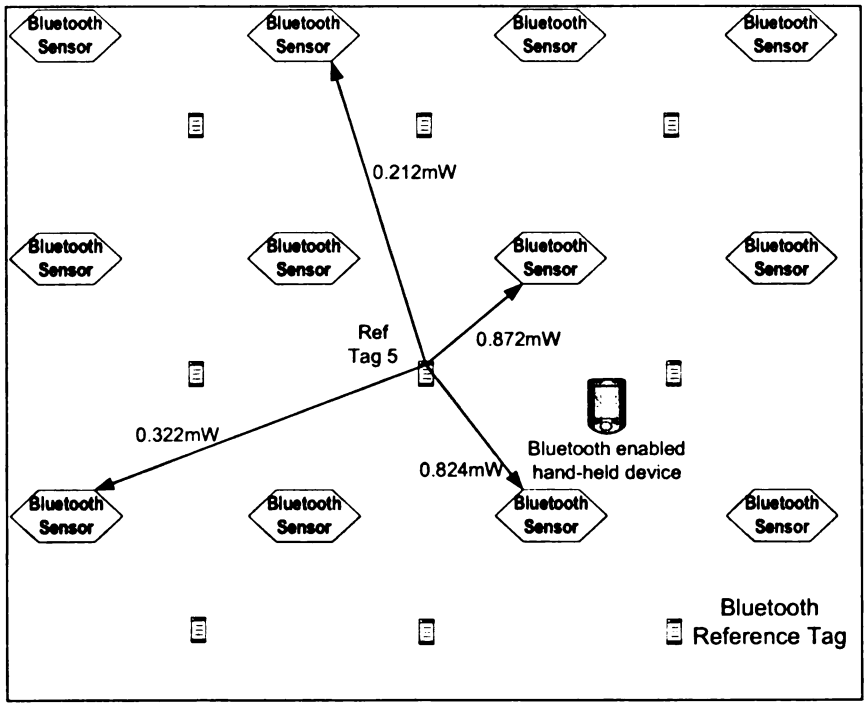


Figure 41: Signal strength of Ref Tag 5 as perceived by Bluetooth readers

Due to the instability of the signals, we cannot obtain the physical distance between the reader and a tag (reference tag or unknown tag) directly from the signal strength. However, with the known coordinates of all the reference tags, we are able to physically locate an unknown tag based on the reference cell of the unknown tag. We can introduce the Euclidian distance in signal strength. For each individual unknown tag, we define:

$$E_i = \sqrt{\sum_{k=1}^{n}(r_{i,k} - u_k)^2}$$

as the Euclidian distance in signal strength between an unknown tag and a reference tag r_i.

To simplify the description of our approach, let us assume there are 4 RF readers and 16 reference tags in the experimental environment. Our approach can be easily extended to an environment that has more than 4 RF readers and 16 reference tags. The signal strength vector of the reference tag and the unknown tag is $S = (s_1, s_2, s_3, s_4)$. When we consider one individual unknown tag, its vectors E_i (for each of the i^{th} reference tag) are given by:

$$E_1 = \sqrt{\sum_{k=1}^{4}(r_{1,k} - u_k)^2} \ldots E_2 = \sqrt{\sum_{k=1}^{4}(r_{2,k} - u_k)^2} \ldots\ldots E_{16} = \sqrt{\sum_{k=1}^{4}(r_{16k} - u_k)^2}$$

Let E denotes the location relationship between the reference tags and this unknown one. We examine three key issues in the process of locating the unknown tag. The first issue is the placement of the reference tags. Since the unknown tag is ultimately located in a reference tag cell, the layout of reference tags may significantly affect the location accuracy of an algorithm.

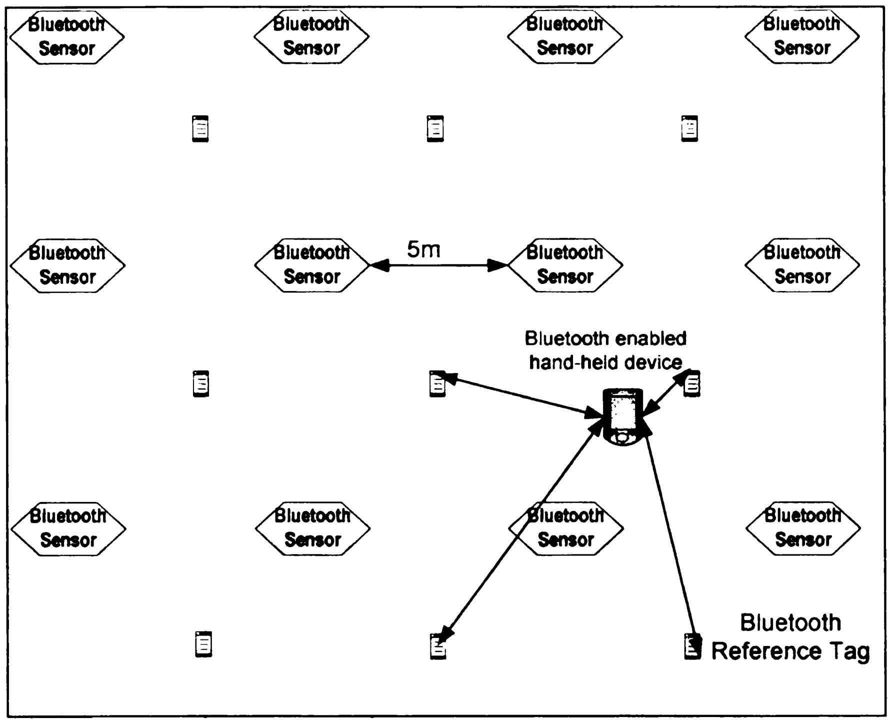


Figure 42: Case of 4-nearest Neighbors

The second issue is to determine the number of reference tags in a reference cell that are used in obtaining the approximate coordinate of the unknown tags. This may also be termed as selecting 'k' nearest neighbors. Now, for example we may use the coordinate of the reference tag with the smallest E value to the tracking tag as this unknown tag's coordinate (k=1), or we can choose 2 nearest tags (k=2) and the unknown tag's coordinates can be simply determined by the arithmetic means of the coordinates of those two nearest tags as:

$$x_{unknown} = \frac{1}{2}(x_{nearest1} + x_{neraest2})$$

$$y_{unknown} = \frac{1}{2}(y_{nearest1} + y_{neraest2})$$

When we are using k nearest reference tags' coordinate to locate one unknown tag, the following equation could be introduced:

$$(x, y) = \frac{1}{k}\sum_{i=1}^{k}(x_{ri}, y_{ri})$$

However, since the nearest neighbors are not at the same distance from the unknown tag, we need to assign weight so that the nearest tag gets more importance than the one far. This becomes the third issue in this approach. Thus, the unknown tag's coordinate can be obtained as:

$$(x, y) = \sum_{i=1}^{k} w_i (x_{ri}, y_{ri})$$

Intuitively, this must be done based on the E value of each reference tag in the cell. Instead of giving weight to all k-nearest neighbors the same weights in averaging, w_i is introduced and it is a function of the E of all k-nearest neighbors. Our approach of the weight is depending on the E as:

$$w_j = \frac{\frac{1}{E_i}}{\sum_{i=1}^{k}\frac{1}{E_i}}$$

In this approach, the reference tag with the smallest E value has the largest weight.

4.3.4 Bluetooth Wish List

Cost of Bluetooth chips is expected to greatly reduce making it possible to have Bluetooth devices that cost less than a dollar. This will make location sensing with Bluetooth very inexpensive. Bluetooth was not designed for location sensing and hence there are certain enhancements that it would have to undergo so that it is best suited for indoor location sensing. Here is a wish list for Bluetooth location sensing:

a. An important improvement would be to reduce the time taken by a Bluetooth device to detect other Bluetooth devices in its vicinity. In addition, many of the BT products have fixed scan (refresh) rate. For example, in the 3COM USB adapter, the lowest refresh time is 5mins. In order to track moving objects, future BT devices should have a shorter wait period between successive scans.

b. The Bluetooth core specifications do not require that signal strength (RSSI) values be available to higher-level software. If this information is made available, it can greatly aid in accurate location sensing. With signal strength information, the nearest neighbor concept developed in [63] and section 4.3.3 can be applied.

c. The Bluetooth spec specifies 3 power levels at which Bluetooth devices can operate. The minimum power level (1mW in class 1) gives a coverage range of roughly 10m. If future implementations can give a shorter range, then location sensing can be made more accurate. Since Bluetooth devices are expected to be cheaper in the future, we can use more sensors covering very short area thus improving accuracy.

4.4 Contributions

This dissertation chapter presents detailed description of three systems for indoor location positioning and tracking. The three systems differ in their technology and method of location computation. The chapter also presents work done by various groups (in both academia and industry) in solving the problem of location positioning in indoor environment.

5 Location Aided Overlay Multicast

Overlay networks have made it easy to implement multicast functionality in MANETs. Their flexibility to adapt to different environments has helped in their steady growth. Overlay multicast trees that are built using location information account for node mobility and have a low latency. However, the performance gains of such trees are offset by the overhead involved in distributing and maintaining precise location information. As the degree of (location) accuracy increases, the performance improves but the overhead required to store and broadcast this information also increases. In this dissertation, we present SOLONet, a design to build a sub-optimal location aided overlay multicast tree, where location updates of each member node are event based. Unlike several other approaches, SOLONet does not require every packet to carry location information or each node maintain location information of every other node or carrying out expensive location broadcast for each node. Our simulation results indicate that SOLONet is scalable and its sub-optimal tree performs very similar to an overlay tree built by using precise location information. SOLONet strikes a good balance between the advantages of using location information (for building efficient overlay multicast trees) versus the cost of maintaining and distributing location information of every member nodes.

5.1 Introduction

Constantly moving nodes and continuously changing network topology characterize mobile ad hoc networks (MANETs). Implementing multicast in such a dynamic environment is a challenging task. A recent survey of the various multicast routing protocols for ad hoc networks is presented in [31]. As pointed out by [66], traditional IP-layer multicast [58], [82], [49] for MANETs have a lot of signaling overhead as it needs to take into account the network dynamics in addition to the (multicast) group dynamics. The widespread deployment of IP multicast has been held back by a variety of issues [33]. Overlay multicast is a new mechanism to support group communication. In comparison to traditional IP multicast, application layer (overlay) multicast relies on the underlying unicast protocols to adapt to the changing network topology. As a result, the application layer has to track only the group dynamic. Due to its ease of implementation and flexibility to adapt, overlay multicast networks (though not as efficient as IP layer multicast) are finding many practical applications in MANETs. AMRoute [23], PAST-DM [42], and LGT [25] are some of the overlay multicast protocols that have been proposed for MANETs. In recent years, research community has shown great interest towards context aware computing and location-sensing techniques [19, 46, 54, 85, 97]. Consequently, position based approach for routing [25, 88] is becoming practical. Figure 43 shows a typical overlay tree with its underlying physical layer links. As seen from the figure, data exchange between member nodes requires traversing other member nodes. This introduces high latency, which increases as the number of member nodes increases.

Also, there are several links that carry identical packets (meant for different member nodes). As a result, application layer multicast is not as efficient as IP-based multicast.

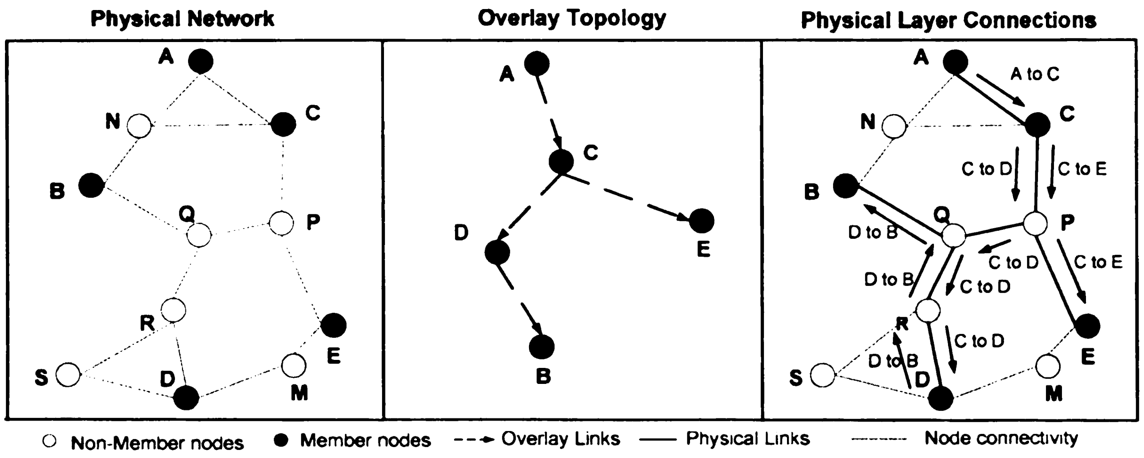


Figure 43: Overlay tree and it corresponding network layer links

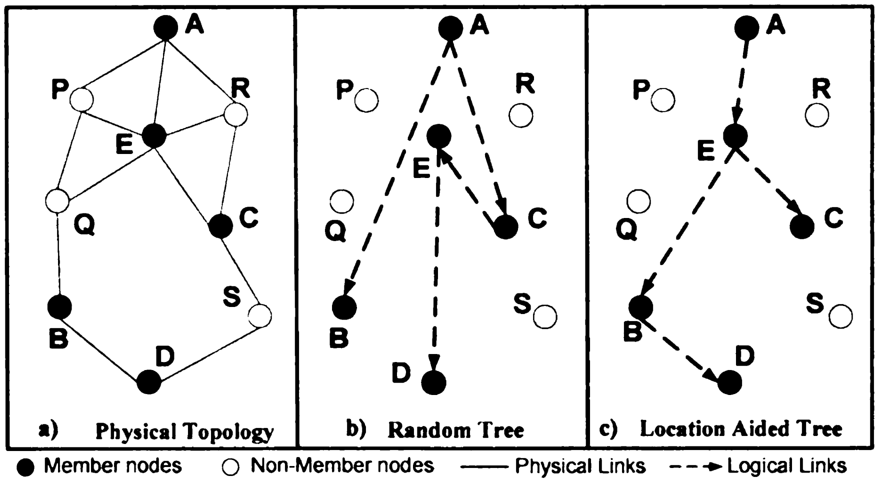


Figure 44: Random vs location-aided overlay tree.

Recently, in an attempt to improve the efficiency of overlay multicast, researchers have proposed the idea of using location-aware overlay multicast trees [25]. Figure 44 shows such an example. A location-aided tree would keep track of member node's movement and would be frequently updated to account for any change in the node positions. The nodes that are physically close to each other would be neighbors in the logical tree (Figure 44c) and the delay at a particular member node will be proportional to its actual distance from the source node. There are several issues with a location-aided approach that need close attention. For example, how to effectively distribute the location information of each member node to other members? How precise should the location information be? How often should this information be updated? Chen and Nahrstedt [25] propose two algorithms (LGK and LGS) to build overlay multicast trees using location information. However, their approach involves storing location information of every member node at every other member node and sending location information in every packet header or periodic location update packets. Location broadcast is a costly affair and if every node starts to broadcast its current location information, then it would quickly lead to the broadcast storm problem [64, 89].

The method proposed in this dissertation builds location-aided overlay trees without requiring every member node to know the location of every other node, or doing costly location update broadcasts periodically. Another aspect of LGT [25] is the use of exact location information for each node. Intuitively, precise location information would lead to a better overlay tree. However, obtaining exact location of a node is a difficult task, especially in indoor environments where ad hoc networks are usually implemented. Frequent updates would be necessary to maintain the accuracy of this information. As the node number and mobility increases, the update frequency would exponentially increase. There is a tradeoff between the advantages of building a location-aided overlay tree versus the distribution and precision of this location information.

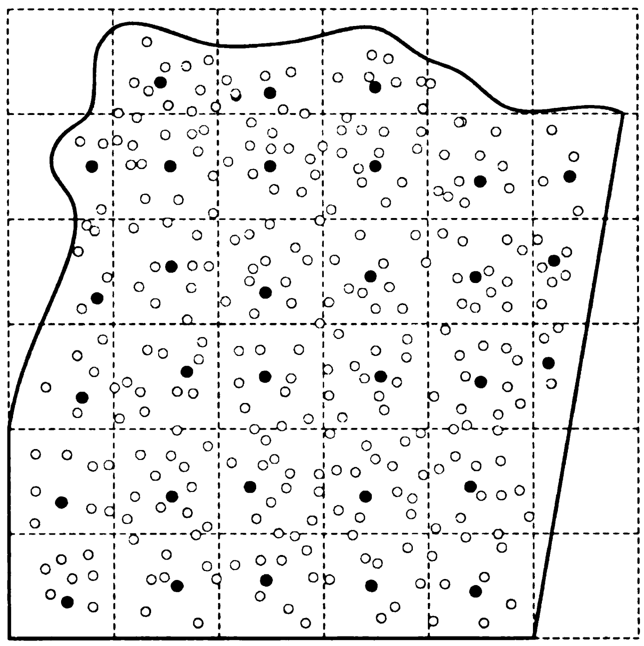


Figure 45: Entire topology partitioned into smaller cells.

In SOLONet, the physical topology is partitioned into smaller areas (cells) having a certain geometric shape (e.g., triangle, square, hexagon, etc). Figure 3 shows an example of such a partitioning. The idea is to make location updates event-based. A node will report a change in its location only when it crosses border to a neighboring cell. While in a particular cell, the node will report the center of that cell as its location. A leader node is selected in each cell to localize certain operation, aid in service discovery and to reduce the number of broadcast messages. A node wishing to form or join an existing overlay network (for a particular service) would query its local leader instead of broadcasting the query to each node in the network. Using ns2 [8] (for simulations), we compare the performance of our design with an 'optimal' overlay tree. We also evaluate the scalability of SOLONet, look at the effects of different location update frequencies on performance and take a closer look at its performance for different cell areas (and member size).

The rest of this chapter is structured as follows. Section 5.2 presents an in-depth description of our design. Section 5.3 presents a detailed analysis of our leader selection algorithm. Section 5.4 presents our simulation results. Finally, Section 5.5 concludes the dissertation and presents directions for future research.

5.2 Design of SOLONet

We begin the description of our SOLONet design by first discussing the various components (like location computation, cell area, cell shape etc) involved.

5.2.1 Location and Geometry Identification

In SOLONet, the entire topology is partitioned into smaller cells (Figure 45). We assume that each node has knowledge of the cell structure for the given topology. Organizers of special events (like games or concerts), where overlay multicast may be implemented, may distribute a coordinate file or a hash function that defines the geometry of the topology. Mobile users, who wish to participate in the multicast network may download the coordinate file (or hash function) from the organizer's website. An alternative approach would be to automate this process so that the nodes obtain the coordinates (topology map) at startup during IP assignment (or they request a copy from a neighboring node that has already joined the network). We also assume that each node knows its current location. This information may be relative to some reference point in the topology and can be computed using the hash function or the coordinate file. To monitor their movement, nodes may use location sensing techniques like GPS in outdoor environment or one of the several indoor location sensing techniques [19, 43, 46, 63, 65, 85] available.

5.2.2 Initial Setup

When the ad-hoc network comes online for the first time, it would run some form of an IP allocation algorithm (e.g., Prophet Algorithm [98]) to get a unique IP address. This IP address will be used to identify each node. In the next few sub-sections, we discuss some characteristics of the partitioned topology.

5.2.2.1 Shape of the Cell

Our design does not put any restriction on the shape of the cells. In theory, the topology can be partitioned into several (non-overlapping) cells of any shape. However, by using repetitive regular cell structures, we can make use of the geometric properties for that shape. Each shape will have it own advantages. A hexagon can closely resemble a coverage area for a given cell, while a square shape is easier to divide into smaller areas if the density of nodes in a cell is high.

5.2.2.2 Size of the Cell

Our design requires that all nodes in any cell are one-hop neighbors of each other. In case of a square cell, for example, nodes at the two end of the diagonal are farthest apart. Therefore, the size of a square cell should be such that the length of its diagonal is less than the coverage radius. This will allow any node in the cell to directly communicate with any other node in that cell. Similar calculations can be done for cells of other shapes using the geometric properties for that shape. Later in the simulation section (Section 5.4.3), we have a detailed discussion on the effect of cell size and node density.

5.2.2.3 Cell Size and Transmission Range

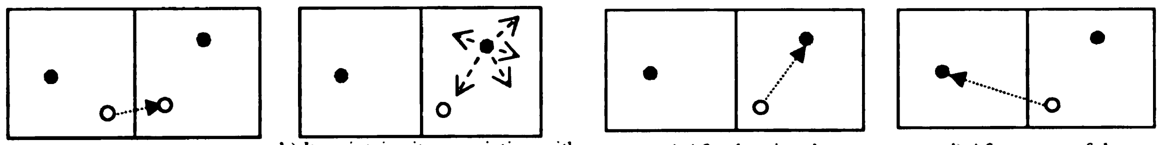
A major factor that needs to be considered when deciding the size of the cell is the coverage area for the technology being used to implement the system. G. Anastasi and group [15], through various experiments, have shown that in case of 802.11b ad hoc networks, the coverage area for a node is around 100-130m at the lowest data rate – 1Mbps. Their study also reveals that the transmission area and throughput depends on the height of the transmitter with respect to the receiver. Similar results may be used for other networks in order to determine the optimal cell size. Our design criteria states that each node in a cell should be able to communicate with every other node in the same cell – meaning the communication has to be single hop with nodes in the same cell. In case of 802.11b technology, a cell size of $75x75m^2$ (or $100x100m^2$) may be used in light loads. If the density of node is very high, $50x50m^2$ may be a good choice. Similar choices can be made for other wireless technologies.

5.2.2.4 Center of the Cell

Since each node knows the cell topology, computing the center of its current cell would not be difficult (using some geometric property of the cell's shape). Each node maintains a cell-ID (CID). During the startup phase, a node checks its position to determine which cell it is currently present. This information helps the node to set its CID parameter. The CID would change when the node moves to a different cell.

5.2.3 Member Nodes and Leaders

Each cell would have a local leader to assist in the service discovery (and tree building) process. This section gives an overview of the responsibilities of a member and a leader node.



a) A member node cross the border into a neighboring cell

b) It maintains its association with the old leader till it hears a beacon from the new leader.

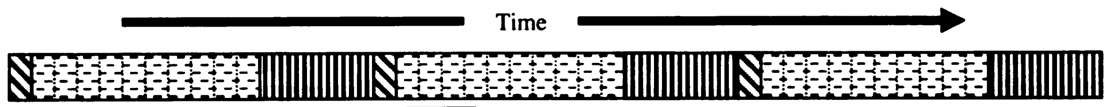
c) After hearing the beacon, it associates with the new leader

d) After successful association, it disconnects with the old leader.

Figure 46: Node association with a new leader after crossing cell boundary.

Table 11: Typical state table at each local leader node

Node ID	X	Y	Type of Service	Battery strength	Time in Cell
12	76.22	108.37	Gateway to Internet	50%	4 min 27 sec
57	73.76	111.29	Live Surgery Video	89%	1 min 12 sec
23	75.32	105.12	Medical Record Files	24%	4 Sec



▨ Beacon Message

▤ Node transmissions (may be slotted)

▥ Forward any service discovery broadcast received. If there are any service discovery request from local nodes, send them to other leaders. Reply to any pending service discovery queries (from other leaders) if the requested node is in this cell.

Figure 47: Time-line showing all the communication happening in a cell (during normal operation).

5.2.3.1 Leader responsibilities

Member nodes constantly update their local leader with information about their location (and service type, battery strength, etc). A leader node maintains a table containing information about each node in its cell. Rows in such a table may look like the ones shown in Table 11. The leader purges a node's entry after it receiving a *disconnect* message from it. The disconnect message may be sent either when the node is gracefully shutdown or has moved to a neighboring cell (and associated with the local leader in that cell). The leader node periodically broadcasts a *beacon* packet to all the nodes in its cell. This beacon message aids in leader selection process, serves as a feedback to the nodes and indicates that the leader node is alive. The details of the beacon packet are discussed in Section 5.3.2. Figure 47 shows a typical communication time-line in every cell. Storing service type of each node is important for service discovery by other nodes in the network. The presence of leader nodes helps in localizing certain operations and limiting the service discovery broadcast to a small fraction of nodes.

Additional responsibilities of a leader could be time slicing of node transmissions or cell splitting. If the node density in a cell is higher than a certain threshold, the leader may assign time slots to each node to avoid collisions between their transmissions. Leader nodes may coordinate with neighboring leaders to assign screwed transmission timeslots so that there is minimal collision between the two cells. These time slots may be carried in the beacon message sent by the leader (Figure 47). Another alternative to solve the high cell density problem could be cell splitting. We do not give a detail discussion of cell splitting or time slot assignment in this dissertation. Several papers in the literature explain how this is carried out.

5.2.3.2 Node responsibilities

When a node first comes online, it waits and listens for the beacon message from the leader node. Once it hears the beacon, it knows who the local leader is. The node provides the leader with information about its current location, battery power and services that it can provide. After the first update, all subsequent updates to this leader carry only the battery and current location information.

When a node enters a new cell, it defers it's *disconnect* message to the old leader till it is able to connect to the leader in the new cell. The node waits for a time equal to the periodicity of the beacon in an attempt to find the local leader (Figure 46a). After receiving the beacon message, this node would know the leader's IP address or other relevant details (Figure 46b). The node would then send an association message to the new leader. The scenario where the leader's beacon message is lost or the neighboring cell has no leader is discussed in later sections. After successfully associating with a new leader, the node disconnects from the old leader (Figure 46c, d). This deferred disconnect procedure ensure that a node is always connected to at least one leader. Since the node has just crossed the cell boundary, it is most likely going to be in the coverage of the old leader. After association with the new leader, the node provides it with information about its current location, battery power and services that it can provide. Similar to the startup case, after the first update, all subsequent updates to this leader carry only the battery and current location information.

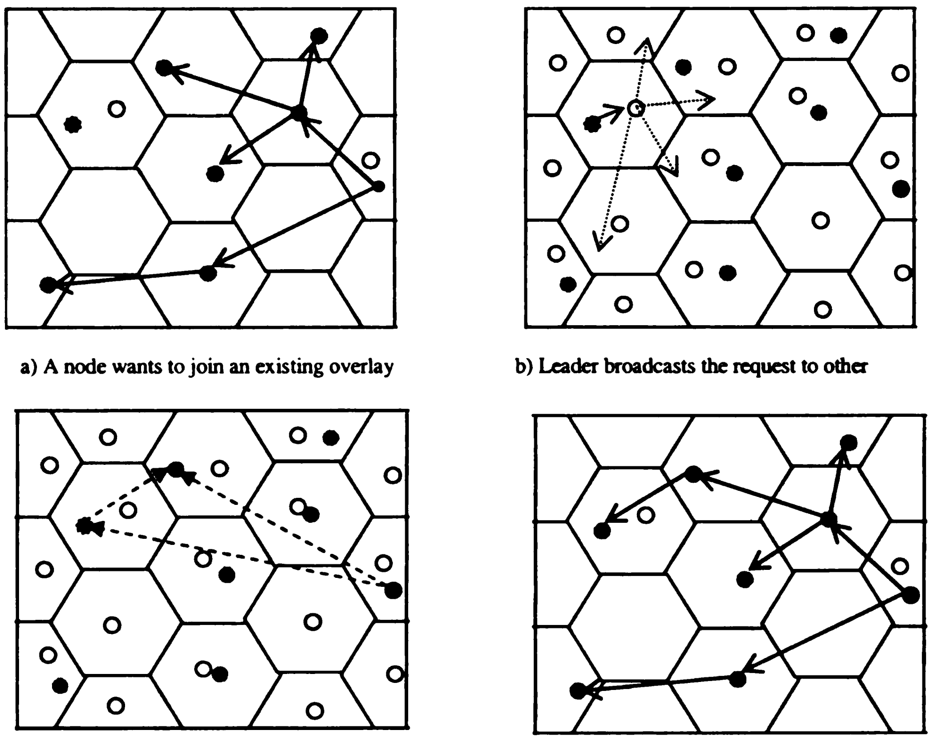
The next section gives a detail description of the role a leader node plays in service discovery and how a location-aided overlay tree is formed.

5.2.4 Service Discovery

When a member node wishes to get a particular service, it would query its local leader. The node may provide the address (if known) of the source node that provides the requested service. For example, there may be a few nodes in the network that act as gateway nodes and provide access to the Internet. A node that wishes to access the Internet may request its local leader to find a gateway node. If the requesting node knows the IP of a gateway node, it may provide the IP along with its request to the local leader nodes. After receiving such a request, the leader node checks its local table (similar to Table 11) to see if the source node is in its list (i.e., in the same cell). If the requested node is not found locally, the leader forwards the request to its neighboring leaders (Figure 48b) using the expanded ring search algorithm [77]. The leader first forwards the request to its immediate neighboring leaders. If that fails, it increases its search space (expands the search radius) and forwards the request to the next set of leaders which, are further away. It must be noted that this message exchange between leaders may be multi-hop communication (and might involving non-member nodes) as far away leader nodes will not be one-hop neighbors. Since the service discovery message is exchanged only between leader nodes, it can be viewed as a multicast between leader nodes. The depth of this multicast tree depends on the cell in which the requested node is found. In its message, the leader node provides IP address and the cell ID of the requesting node. We have tried to keep this message as small as possible by having only two items (IP and CID) in the message. Since there are very few leader nodes in the entire network the overhead will be very low. Each leader node, upon receiving the service discovery message, checks their local node list to see if the requested (service) node is in its cell. If the requested node is found, then the leader forwards the request (message) to that node (Figure 48) and sends a positive ack to the requesting leader.

In the event that the requesting node doesn't get any response for a timeout period, it resends its request. Certain request may not generate any reply either because there is no node in the entire network that can provide the requested service or because the leader in the requested node's cell is dead (Figure 52). Both of these problems can be solved if the timeout period follows an exponential back-off scheme. Such a scheme would give enough time for any leader selection process, which may have started during the service discovery process, to complete. The system can be configured so that after a certain number of timeouts, a node stops making request for that service.

5.2.5 Joining an Overlay Tree


a) A node wants to join an existing overlay

b) Leader broadcasts the request to other

c) Source node located. Source contacts the requestor and gives address of the nearest node to connect to.

d) New overlay tree.

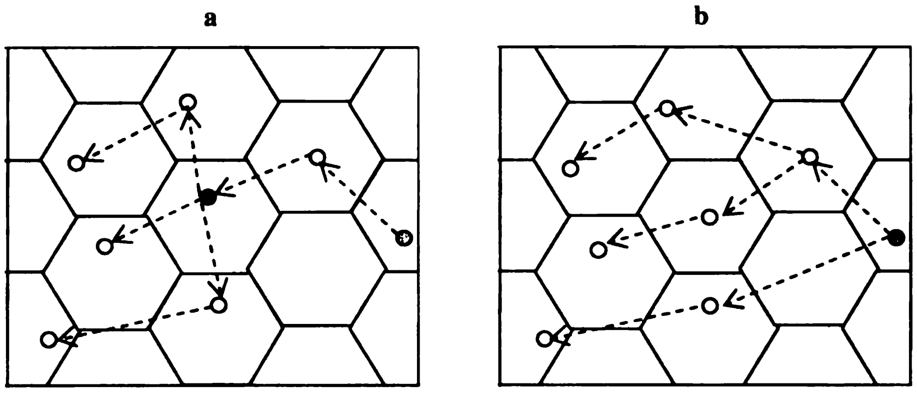
○ Leader nodes ● Member nodes ● Requesting node ● Source node

Figure 48: Building a location aware overlay tree.

The node that provides services (like data storage, access to Internet, computing resources etc) to member nodes in a multicast tree is referred to as the source node. This source node would usually have more resources at its disposal. It is the root of the multicast tree, which is built for providing a particular service. The source node is responsible for building and the growth of the multicast tree. It stores the CIDs of the member nodes that it is currently serving. When it receives a request for service (from a new member), it extracts the CID (from the request message) and finds the position of the requesting node. The position is assumed to be in the center of the requestor's cell. Simulations in Section 5.4.1 investigate the performance with this assumption. The source node now checks its internal tree-table to find node(s) that may be in the same cell or neighboring (closest) cell as the requesting node. It now contacts the requesting node and provides it with the IP address of a member node nearest to it. The source also provides appropriate information to this designated 'nearest' relay node about the requesting node (Figure 48c). With this information, the requesting node can now connect to a near-by (physically close) member node, which would in turn provide the required service (Figure 48d). As our simulations confirm, the latency with this approach is much lower compared to the case where a source node randomly selects a node in the tree for the requestor to connect to. In case of prioritized overlay multicast [72], [94], the source node will also provide information about the priority of the service that it provides. This would help in the formation of a prioritized overlay tree. At the time of request, the requesting node may be part of some other group(s) having a different priority. For example, in a particular hospital, all doctors, nurses or resident students doctors may have their own category (viz *doctor_net*, *nurse_grp*, *student_org*) in addition to a common (low priority) group called *hospital_net*. In an emergency, a group of doctors, nurses and residents may form a high priority network to address a specific patient case. A detailed discussion on priority trees and their formation can be found in [72], [94].

5.2.6 Degree Bounded Locations based Tree

Consider an example of building a simple location-aided (sub-optimal) tree using the model (Figure 49a) described in the previous section. The request to join will propagate through leaders until it finally reaches the source node. The source node now looks at its internal topology map and finds a member node that is nearest to the requesting node. This 'plain' approach may lead to the case where one or more of the relay nodes become a single point of failure. In addition, it may happen that the packet delivery load may not been evenly distributed amongst the relay nodes creating several connections at a node; thus making it a bottleneck. More connections mean larger state-tables and higher power consumption. In short, this would lead to faster depletion of resources (battery power, memory, etc) at the bottleneck node. It may also cause collisions between the different connections directly affecting the throughput.



○ Source Node ● Bottleneck Node ○ Member Nodes

Figure 49: Degree bound location aware tree.

Table 12: Weight calculation for a requesting node

Node ID	Location	Relative Position	W_N	W_D = Degree	$X = \sqrt{(W_D^2 + W_N^2)}$	$W_T = (X)^{-1}$
24	Cell (4E)	One cell away	2	2	2.83	0.35
8	Cell (2B)	Two cells away	3	2	3.87	0.25
17	Cell (2F)	Neighboring cell	1	4	4.12	0.24

To eliminate the weakness of such a 'plain' tree, we propose a degree bounded architecture. In this approach, after a source node receives a request, it looks at its internal topology map and finds a set of member nodes that are near to the requesting node. This set will consist of nodes that are in the same cell, immediate neighboring cell and up to three cells away from the requesting node. In order to keep the search result small, we chose to look up to three cells away. This default setting may be changed in a dense topology where the multicast tree may have several member nodes. In such a case, the source may consider member nodes that are in the same cell, immediate neighboring cell and one cell away. Restricting the ring of nearest neighbors will keep the found set as small as possible. The source now sorts this found set by assigning weights proportional to the degree (number of connections) of the node and its 'closeness' to the requesting node. The total weight (W_T) for a node is calculated such that a node closer to the requesting node would contribute positively while a node with high degree node would tend to bring down the W_T value. Table 12 shows one such calculation. In this example, nodes in immediate neighboring cells are assigned a neighbor weight (W_N) of 1. The W_N value for a node is higher if it is farther away from the requesting node. The degree weight (W_D) of a node is directly proportional to its current degree. In our example, we have kept the W_D of a node equal to its current degree. To make the process of relay node selection lightweight, the computation of W_T is kept as simple as possible. The source selects the node with the highest W_T value as the relay node and provides the requesting node with its address. The requesting node can now connect to this node. This approach would ensure that a near-by node with little or no connections get selected as relay node for a new connection (Figure 49b); thus evenly distributing the load. A similar algorithm can also be used when the source node decides to reorganizes the multicast tree to compensate for changes in member node positions due to their mobility.

5.2.7 Event-based Update

Every source maintains a list of member nodes that it serves (either directly or through other members). A member node updates the source with its new location (CID) only if it changes its cell. This approach reduces the amount of location updates and the associated overhead. The updates are sent to the source in-band along with the ack packets. The source node can alter the tree structure if the location update(s) indicate a major change in the topology. Although the resulting (location aware) tree is sub-optimal, one can appreciate the gains of this method compared to the expensive broadcast approach which is used when precise location information is needed for tree construction. Simulation results in Section 5.4 show that this approach pays a very little performance penalty compared to an approach where exact location information is used.

5.3 Leader Selection for Cells

Before we get into the details of the leader section process, the next two sub-sections look at some of the arrangements in our design that aids the leader selection process.

5.3.1 Responsibilities of Leaders

Every leader performs activities that can aid in the selection of a new leader in case of its failure. In addition to the location and the service information, each leader also maintains battery status and the time a particular node spent in its cell (Table 11). Weights are assigned to the battery strength, time-in-cell and node's current location information. The entries in the list are sorted according to the result of this weighting function. The first two nodes or the top 10% nodes (whichever is greater) in this sorted list are called *candidate nodes* – meaning that these nodes are potential candidates for leadership in case the current leader fails. Listed below are some important parameters used to choose the candidate nodes.

5.3.1.1 Time information

It has been observed in [34] that hosts that have been stationary for a period of time are more likely to remain stationary as compared to those currently in motion. Thus, choosing a node that has shown little movement or no movement as the leader would greatly increase the chances that it would stay in that cell for a long time.

5.3.1.2 Battery strength

A leader has a lot of responsibilities and this demands battery power. A node, which has good battery strength, should be selected as the leader, so that it can perform the leadership responsibilities without interruption for a long time.

5.3.1.3 Location

A leader situated more or less towards the center of the cell can serve all the cell nodes with little delay. Even if this node were to be in motion, it would take a longer time for it to move out of the cell due to its distance from the cell's periphery.

The next section shows how the sorted list is made available to all the nodes in the cell through the beacon message.

5.3.2 Beacon Message

Every leader periodically broadcasts a beacon packet containing a sorted list of nodes in its cell. These beacon messages serve three purposes.

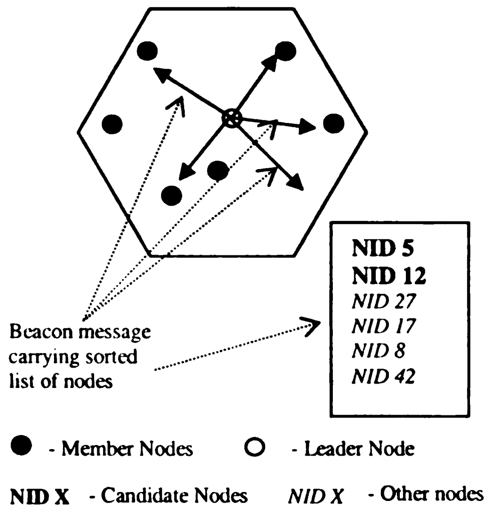


Figure 50: Beacon message from the leader node.

5.3.2.1 Leader's heartbeat

Beacon is a way for the leader to tell the other nodes that it is alive. If nodes do not receive beacons for a pre-configured timeout period, they would suspect that the leader is no longer available. The use of timeout will help prevent false detection. A beacon packet may be lost due to noise, multi-path fading or collision with some other transmission. Noise and fading depend on environmental conditions while collision (although rare) may depend on density of member nodes. Assigning time-slots to each node can reduce collisions. The timeout value is not fixed and depends on factors like noise and collision rate in that environment. It will be available to the nodes at start up when they acquire the topology coordinates (or hash function). The leader node may become unavailable for the following two reasons. The user 'pulled the plug' – turned off the device in an unconventional manner (e.g., suddenly removed the batteries). In such a scenario, the leader node would die without informing any other node. The other case is when a leader sends a message saying that it is stepping down but the message was lost (perhaps due to noise or collision). The use of periodic beacon will help detect the above two scenarios.

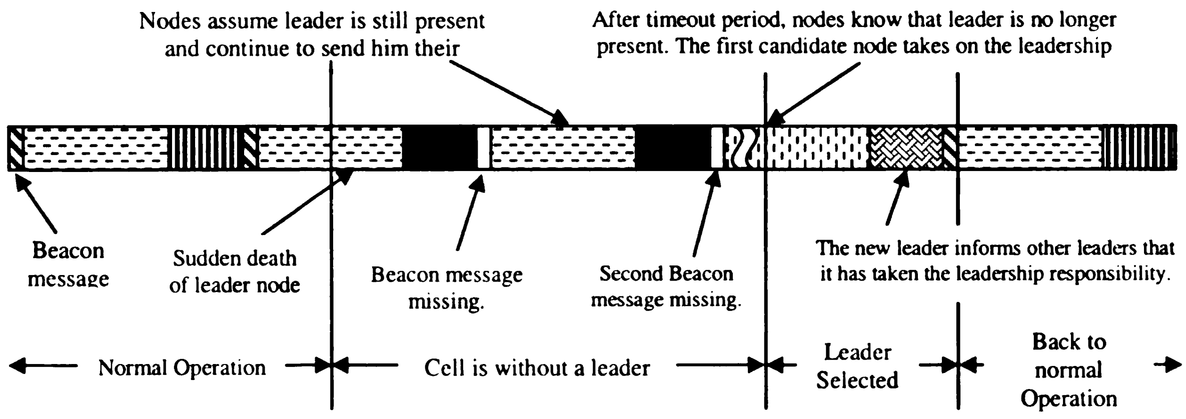

Nodes assume leader is still present and continue to send him their
After timeout period, nodes know that leader is no longer present. The first candidate node takes on the leadership
Beacon message
Sudden death of leader node
Beacon message missing.
Second Beacon message missing.
The new leader informs other leaders that it has taken the leadership responsibility.
Normal Operation
Cell is without a leader
Leader Selected
Back to normal Operation

Figure 51: Communication time-line – failure of a leader and selection of a new leader.

5.3.2.2 Aid in leader selection

The beacon message contains the sorted list of nodes (Figure 50). Leader failures are very rare; however, in case the current leader fails, all the nodes would detect the failure after certain numbers of beacon messages are missed by the leader. The nodes would then examine the sorted (node) list that came in the latest beacon broadcast. The first candidate node has to acknowledge that it is taking leadership before the next beacon period. If it fails to do so, the second candidate node sends a broadcast declaring itself as the leader. The possibility that both nodes become unavailable at the same time is very rare and if it happens, then after another timeout, the next candidate node in the sorted list takes over the leadership. One of the responsibilities of the new leader is to provide its IP address information to the leaders in the immediate neighboring cells.

5.3.2.3 Feedback to each node

The beacon message can serve as a feedback to each node to indicate that its message (containing location and battery information) reached the leader without any error. The beacon message would help identify the IP address of the new leader node when a node crosses over into a new cell. In case of a crowded cell, the beacon may also contain the time slots telling each node when to transmit its information thus to avoid collisions between two (or more) nodes. Implementing slotted transmission in a non-dense cell is optional. Nodes can also use the beacon to synchronize their clock with the leader node.

5.3.3 Hello!! Anybody Home?

This section describes a worst-case scenario. When a node crosses border and enters a neighboring cell, it maintains its association with the old leader and waits for the beacon from the new leader. What if there is no leader in the new cell or if the leader there died? In our design, the node waits a little longer than the timeout period mentioned above. This would give other nodes (originally present in that cell) enough time to take over the leadership. After this long wait, if there is still no sign of a beacon from a leader, the node assumes that the neighboring cell was empty – neither the leader nor member nodes were present. The node now broadcasts a message containing its IP and its desire to become the leader (Figure 52a). Since the cell size is such that any node in that cell will hear this broadcast, the node waits for a small timeout period for response from any other node that might recently enter the cell (Figure 52b). If there were another node that entered the cell at approximated the same time, the node with the lower IP would take over the leadership. After the node has taken the leadership responsibility, it sends a *disconnect* message to the old leader. During this leader election process, the node was still associated with the old leader. Since the node has recently crossed the border, the inaccuracy in its location information would be close to the location inaccuracy for a node in the older cell, which is near the border of the cell.

5.3.4 Initiation of Leader Selection

There are three scenarios when a leader selection is required as described below:

5.3.4.1 When the network first comes online

This scenario is similar to the case where a node has entered a cell with no leader. We follow the same leader selection procedure described in Section 5.3.3 and choose the node with the lowest IP as the local leader. Figure 52 explains this procedure. The first leader may not be the best leader in terms of battery power, location and other factors.

However, one of the responsibilities of the leader node is to find a good replacement leader. Thus, the subsequent leaders would be wisely chosen.

5.3.4.2 Leader wishes to give up leadership

There can be different cases that may cause a leader to give up its leadership – proximity to the cell boundary, running low on battery or the user has gracefully switched off (shut down) the mobile device. After it has decided to quit, the leader sends a broadcast message informing all the nodes. One of the *candidate nodes* takes over the leader responsibilities.

5.3.4.3 Exceptional cases – rare in occurrence

Leader's quit message was lost due to collision or noise or the user switch off the device in an unconventional manner. Both these conditions are seldom possible and would be detected by the absence of beacon messages for a timeout period. In this scenario, one of the candidate nodes takes over the leadership responsibility. The candidate node list will be available to all the nodes from the leader's previous beacon message (Section 5.3.2.2). Figure 51 shows a time of events that happen after a leader's sudden death.

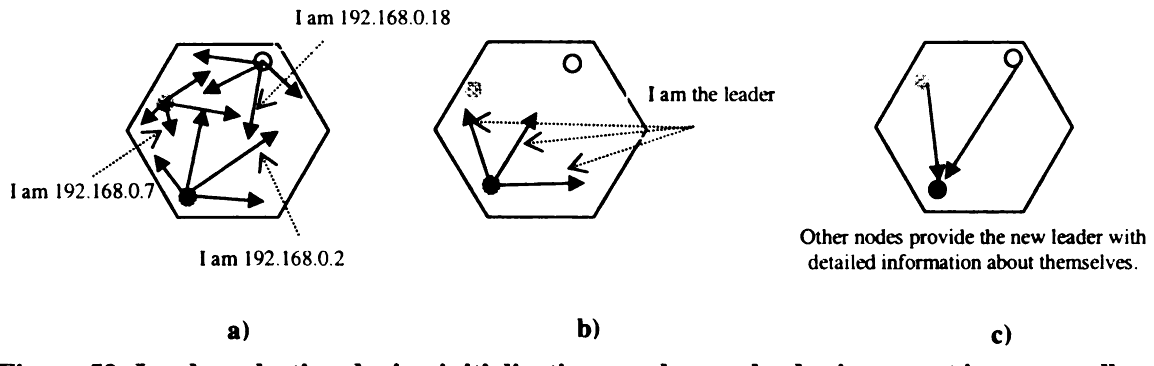

I am 192.168.0.18
I am 192.168.0.7
I am 192.168.0.2
I am the leader
Other nodes provide the new leader with detailed information about themselves.
a)
b)
c)

Figure 52: Leader selection during initialization or when no leader is present in a new cell.

5.4 Simulations

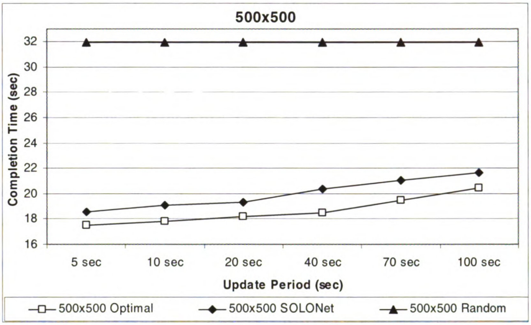
Simulations were carried out using ns2.26 [8]. As of this writing, ns2 doesn't have any extension for simulating overlay multicast in MANETs. With the help of C-programming and bash scripting, the traffic pattern generated by CMU's cbrgen utility was modified to represent a location-aided overlay network. The Prim's algorithm [32] was used to generate the multicast trees. The distance of the nodes was used as the weight in calculating the MSTs. The setdest utility was used to generate different node positions and movement patterns. The nodes in the simulation move according to the 'random waypoint' model [51]. According to the model, when the simulation starts, each node is stationary at a particular location in the specified area for a time equal to the specified pause time. After the pause time expires, the nodes select a random destination within the given area and start to move with the maximum specified speed (during the creating of the scenario file). After reaching the destination, the nodes stay stationary for a time equal to the pause time, select another destination, and proceed towards it. It is easy to see that smaller pause time means higher mobility. For all the simulations, DSR was the underlying unicast protocol. Two-ray ground model is used as the radio propagation model. We use the IEEE 802.11 DCF as the MAC protocol.

The first set of simulations compares an optimal tree (which is built by using precise location information of member nodes) and our proposed sub-optimal (SOLONet) overlay tree. The optimal tree that we generate is similar to the tree generated by the LGT approach (except for some of the packet level optimizations suggested by LGT). This optimal tree utilizes exact location information and builds a Steiner tree where the link cost is the geometric distance between the nodes. In case of SOLONet tree, nodes report the center of their current cell as their location. Therefore, SOLONet's Steiner tree is not built with precise location information. This comparison is done for two different areas: 500x500m^2 and 800x800m^2. The second simulation set aims to show the scalability of our design. We compare the performance for 10, 15, 20 and 30 member nodes. The third simulation set compares the performance for difference choices of cell area and for different number of member nodes. We compare the performance for 25x25, 50x50, 100x100 125x125 and 250x250m^2 for a topology of 500x500 m^2. In all the three scenarios, the file size used for transfer is 50KB and the packet size is 512 bytes. In the all the simulations, the 'Completion Time' is the time it took for all the nodes to receive the 50KB sent by the source node. Note, this time will depend on the delay-distance between the source and the farthest leaf. This is the same as the eccentricity of the source node. (The eccentricity of a node v in a connected graph G is length of the longest of all the shortest paths between v and every other point in G).

5.4.1 Optimal vs Sub-Optimal (vs Random)

Each node updates its local leader with information about their current location. The periodicity of this update determines the accuracy of the location information present at the local leader. This location information will be used during the formation of the location-aware tree. As mentioned earlier, there is a trade off between the frequency of updates and the overhead involved. With lower update times, the node information will be most current at the leader nodes. However if each node were to initiate frequent updates, the network would be swamped with update packets. This section presents results of simulations for different update times. The simulation is performed for an area of 500x500m^2 and 800x800m^2. The cell size in both cases was chosen to be 100x100m^2.

The movement pattern was 5 m/sec with a pause time of 10 sec. The total number of nodes was set to 150 and the number of member nodes was kept at 15. These nodes report the center of their (respective) cells as their location during the formation of the overlay tree.



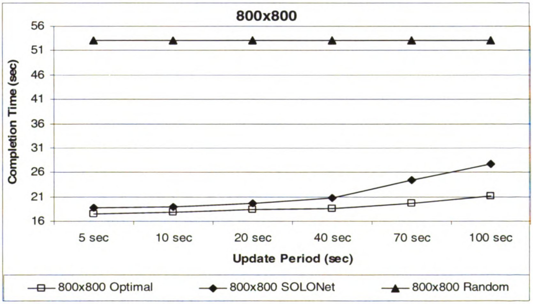


Figure 53: Comparison between accurate location information and location reported as the center of the cell.

In each case, the (tree building) start time was a randomly chosen value between 0 and the update value (chosen for that particular simulation scenario). For example if the update value chosen was 70 sec, then the start time would be randomly distributed between 0 – 70 sec. By having start-up time between 0 and update time, we try to simulate the situation where an overlay tree was built using location information that was updated "*start-time*" sec before. Each simulation result is the average of 50 different scenarios. From Figure 53, it is clear that the performance of a sub-optimal tree closely matches with that of an optimal tree. Figure 53 also shows the performance when no location information is used (i.e. a random overlay tree).

Table 13: t-Test comparison between Optimal and Sub-Optimal

Update time / Simulation Area		5 Sec	10 Sec	20 Sec	40 Sec	70 Sec	100 Sec
	p-value						
500x500	1-tail	7.24 e-06	0.000134	0.003781	0.001651	0.001818	0.023289
	2-tail	1.45 e-05	0.000268	0.007562	0.003301	0.003637	0.046578
800x800	1-tail	0.028161664	0.011959	0.095029	0.053384	0.006982	0.003333
	2-tail	0.056323328	0.023919	0.190058	0.106769	0.013964	0.006665

Table 14: t-Test comparison between Optimal and Random

Update time / Simulation Area		5 Sec	10 Sec	20 Sec	40 Sec	70 Sec	100 Sec
	p-value						
500x500	1-tail	1.19 e-08	1.18 e-08	1.99 e-08	2.7 e-08	1.18 e-07	3.43 e-07
	2-tail	2.38 e-08	2.36 e-08	3.97 e-08	5.4 e-08	2.36 e-07	6.86 e-07
800x800	1-tail	3.25045 e-13	6.28 e-13	4.69 e-13	1.06 e-12	1.07 e-12	6.17 e-13
	2-tail	6.50091 e-13	1.26 e-12	9.37 e-13	2.12 e-12	2.13 e-12	1.23 e-12

We conducted statistical (t-test with significance of 0.005) analysis on the simulation data. The test results are shown in Tables Table 13, Table 14 and Table 15. According to the t-test results, there is no statistically significant performance difference between optimal and sub-optimal at the updated period levels 5 sec through 40 sec. However, the completion times are significantly lower in order optimal < suboptimal < random in case of 70sec and 100sec at the level of 0.05 for both areas.

Table 15: t-Test comparison between Sub-Optimal and Random

Simulation Area \ Update time		5 Sec	10 Sec	20 Sec	40 Sec	70 Sec	100 Sec
	p-value						
500x500	1-tail	3.53 e-08	6.1 e-08	6.17 e-08	2.29 e-07	5.34 e-07	1.23 e-06
	2-tail	7.07 e-08	1.22 e-07	1.23 e-07	4.58 e-07	1.07 e-06	2.45 e-06
800x800	1-tail	1.10139 e-12	1.43 e-12	1.03 e-13	4.15 e-14	3.32 e-13	3.28 e-11
	2-tail	2.20278 e-12	2.86 e-12	2.06 e-13	8.3 e-14	6.64 e-13	6.55 e-11

5.4.2 Scalability Consideration

We repeated the above simulations for an area of 500x500m^2 for different number of member nodes – 10, 15, 20, and 30 nodes. The update time of 20 sec was chosen – which meant that the nodes updated their location information randomly in 0 to 20 sec. The results in Figure 54 show that the completion time increases with the increase in the member nodes. This was expected. The simulation also ascertains the scalability of our design – the performance of optimal and sub-optimal trees is very close.

5.4.3 Effects of Smaller Cell Size

The aim of this set of simulations was to find a relation between the performance and cell size. Cell sizes of 25x25, 50x50, 100x100 125x125 and 250x250m^2 were checked. The topology area was chosen to be 500x500m^2. The simulations also had varying member size – 10, 15, 20 and 30 members. The movement pattern was 1 m/sec (human walk) with a pause time of 10sec.

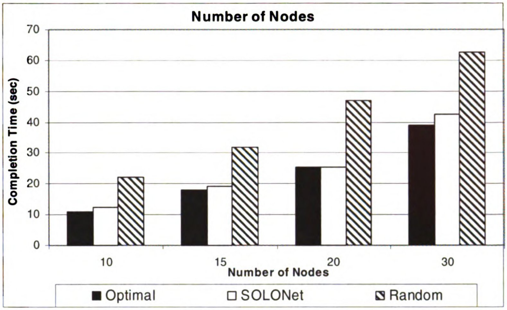


Figure 54: Scalability of the protocol.

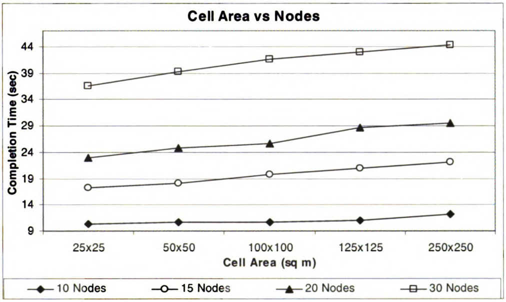


Figure 55: Performance of SOLONET for different cell sizes and member nodes.

It is easy to see from the result in Figure 55 that the performance holds an inverse relation with the cell size. Smaller cell area gives an improvement in the performance. This is because smaller cell areas imply higher accuracy in the location information used for building the location tree.

5.4.3.1 Discussion

This improvement in performance (with smaller cell size) is offset by an increase in the service broadcast overhead. When the topology is divided into large number of small cells, any broadcast (during service discovery – Section 5.2.4) would now pass through more leaders and will take longer to reach the entire leader set. As a result, the service discovery process will be slower and very inefficient. On the other hand, with larger cells, the location accuracy is lowered but the communication overhead and the propagation delays are reduced. Thus, there is a tradeoff between the overhead in service discovery and the performance of the overlay tree. Here's a simple back-of-the-envelope calculation showing the effect of smaller cell size. Consider the easiest case of controlled broadcast where the service discovery broadcast message reaches a leader node not more than one time. In a topology that is partitioned into four cells, there will be at least three broadcast messages generated during any service discovery. If the topology is further divided into smaller cells with each cell having half the earlier length, total cells how increases to sixteen and the broadcast messages increase to fifteen. The number of broadcast keeps increasing exponentially (as seen in Figure 56) every time the cell size is further divided to half its current length. The worst-case scenario is when the cell size is so small that every cell has just one node in it. In such a case, the broadcasts will be between each node and will be exponentially proportional to the number of member nodes. This is the same as the case where no cells are used. The purpose of using cell structure is to limit the broadcast only between the leader nodes.

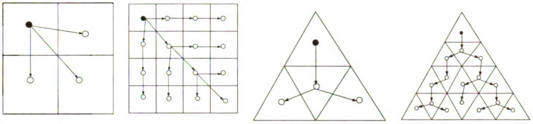

(a) 1/4th area (square) (b) 1/16th area (square) (c) 1/4th area (triangle) (d) 1/16th area (triangle)

Figure 56: Broadcast scenario

5.4.3.2 Reducing broadcast overhead in small cells

Figure 55 showed how decreasing the cell size improves the location accuracy and hence the performance of a location-aided tree. However, this improvement comes at the cost of high overheard during any service discovery broadcast (Figure 57a). In order to solve this problem, we propose some enhancements to our architecture. We suggest the use of two separate cell partitions (each of different sizes). The service discovery mechanism would refer to a larger cell partition called *service discovery* cells. These large *service discovery* cells would house several smaller *regular* cells. Nodes would use *regular* cells during their normal operations (like leader selection, event based update, beacon message etc). Only leader nodes on the other hand will use the larger cells during the service discovery process. Certain cell leaders would be designated as service leaders for that region. The selection of service leaders can be predetermined. For example, if the *service discovery* cell consists of four regular cells, then the service leaders can be chosen as follows:

{2p+2x, 2q+2y} [p=0,n & q=0,m]

where n=1/2 x (no. of longitudinal cells); m=1/2 x (no. of latitudinal cells)

This gives (2,2); (2,4); (4,6); (6,2)... etc as the service leader set (e.g. Figure 57b).

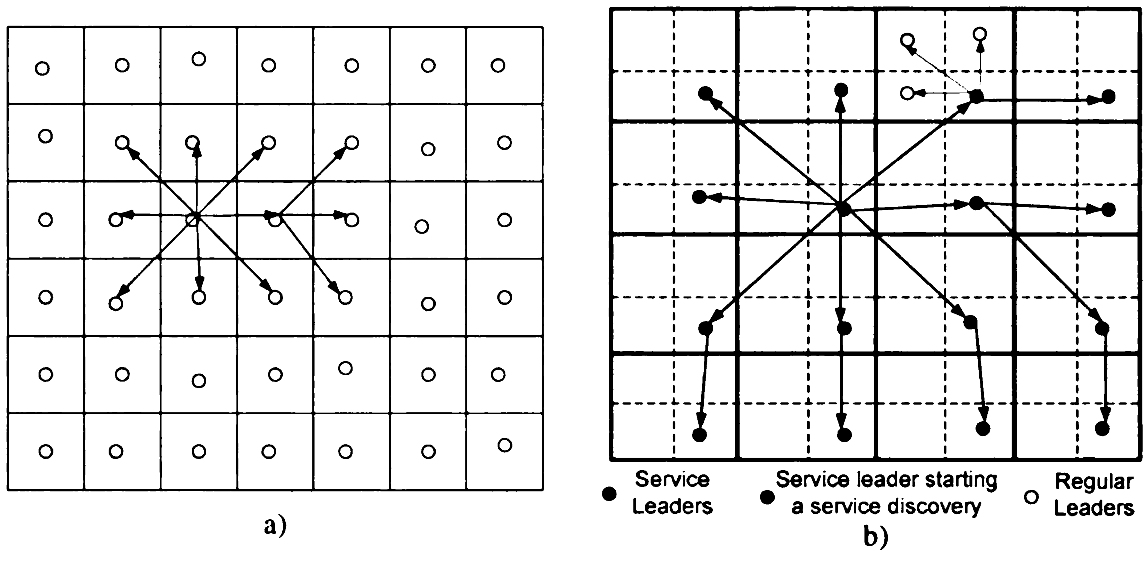
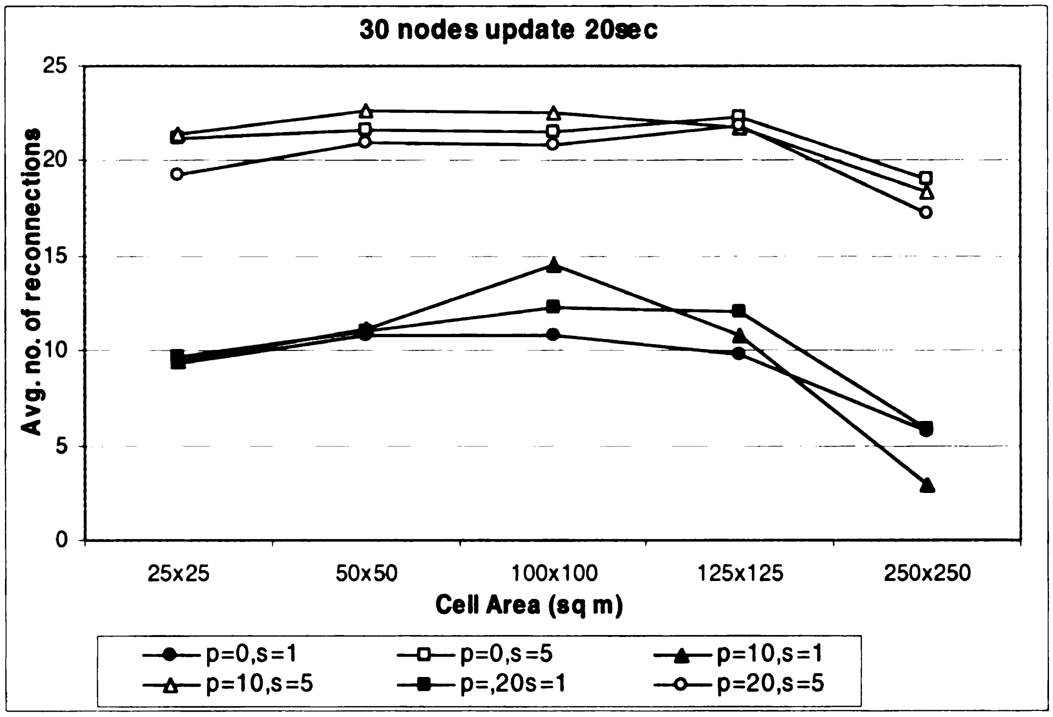


Figure 57: Smaller cell size – more service discovery broadcasts.

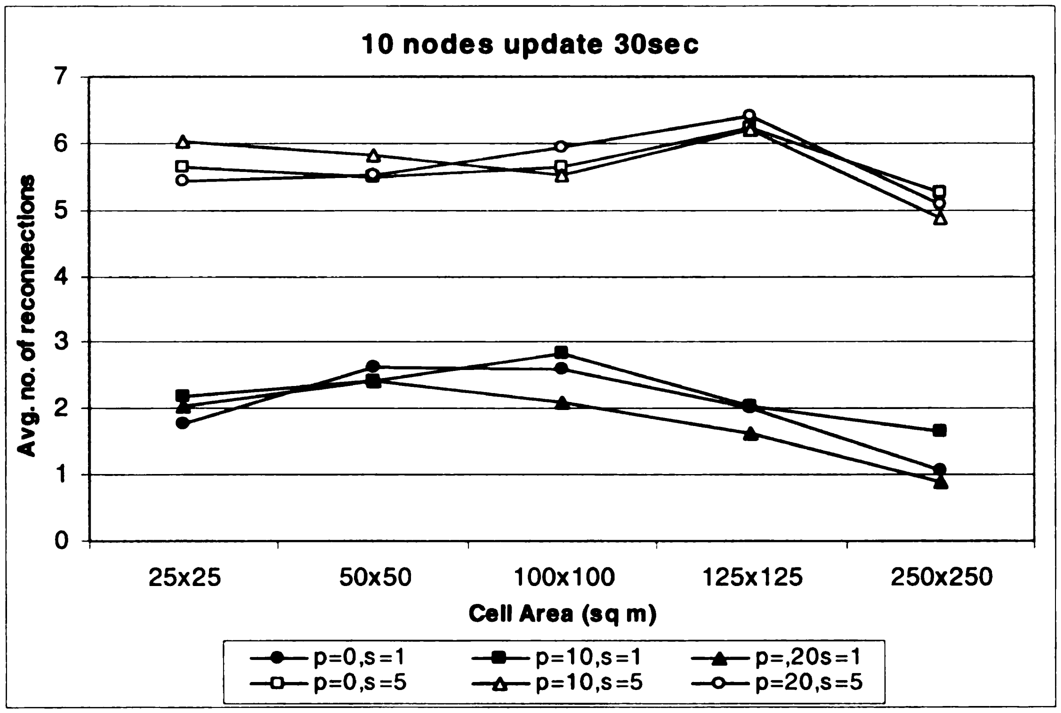
During any service discovery process, a service leader would forward the broadcast message to their neighboring service leaders and to the regular leaders in its *service discovery* cell. By using smaller cell size for regular operations, we can continue to reap the benefit of better location accuracy while the larger service cell limits the broadcast overhead between service leaders only (Figure 57b).

5.4.4 Cell Size and Mobility

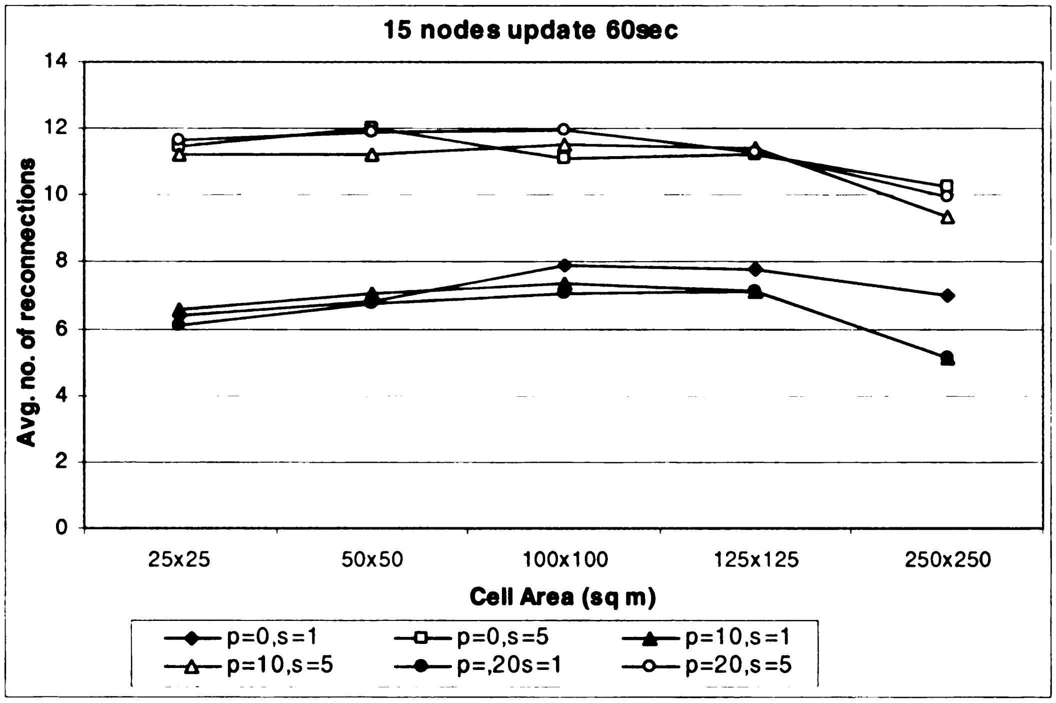
When a node moves from one cell to another cell, it reports a change in its location to the source node. The change is reported as a new CellID (CID). The source node may decide to rearrange the overlay tree to compensate for the new positions. This reshuffling would lead to several new connections (and equal number of disconnections) between member nodes.


30 nodes update 20sec
Avg. no. of reconnections
25
20
15
10
5
0
25x25
50x50
100x100
125x125
250x250
Cell Area (sq m)
p=0,s=1
p=0,s=5
p=10,s=1
p=10,s=5
p=,20s=1
p=20,s=5

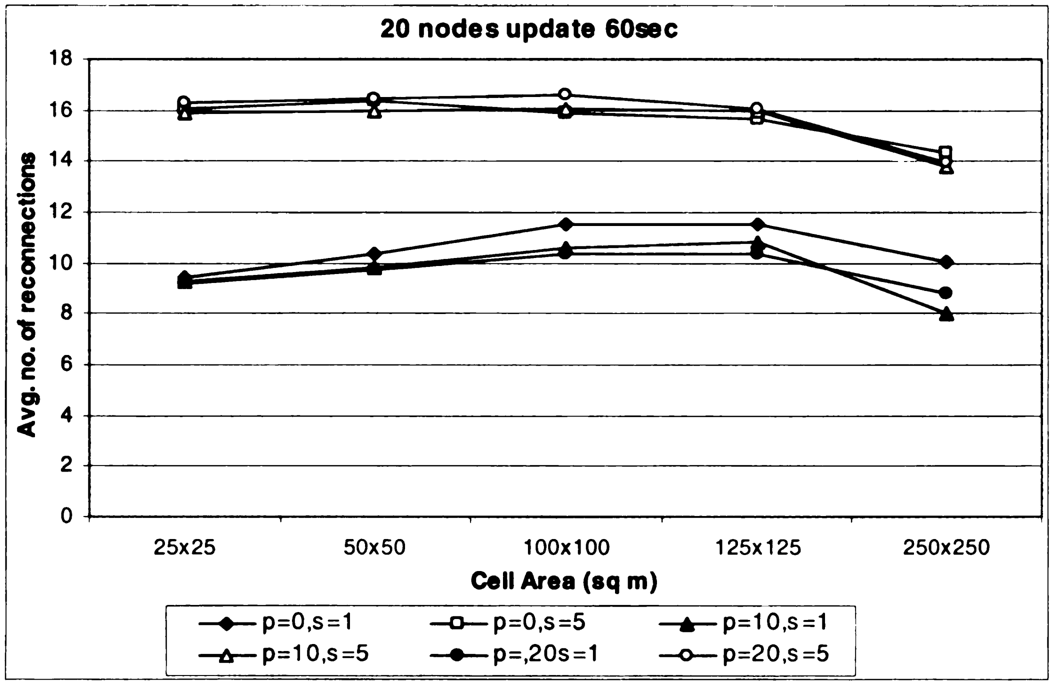
a)


10 nodes update 30sec
Avg. no. of reconnections
7
6
5
4
3
2
1
0
25x25
50x50
100x100
125x125
250x250
Cell Area (sq m)
p=0,s=1
p=10,s=1
p=,20s=1
p=0,s=5
p=10,s=5
p=20,s=5

b)



c)



d)

Figure 58: Cell size vs Node mobility

We defined the term *reconnections* to indicate the new connection number. The cell size should be chosen such that the *reconnections* number is small. *Reconnections* are expensive – every new connection would change the route information at the underlying network layer at all the intermediate non-member nodes. This may initiate route discovery process depending on how the underlying unicast protocol is implemented. In addition, with every *reconnection*, state tables have to be made at the source and destination node for the new connection.

The simulation setup was similar to the one used in Section 5.4.3. The topology area was 500x500m^2. Simulation for cell sizes of 25x25, 50x50, 100x100 125x125 and 250x250m^2 were carried out with varying member size – 10, 15, 20 and 30 members. The movement pattern was 1 m/sec (human walking) and 5m/sec (human running) with a pause time of 0 (continuous movement), 10sec and 20sec. The simulation follows the 'random waypoint' model [51]. Since the performance is sensitive to movement pattern, all the results in this set are average of 30 different movement scenarios (generated by ns2 for the same pause time and speed combination).

The number of reconnections for a speed of 5m/sec was much higher compared to speeds of 1m/sec (Figure 58). Within each speed, higher pause times gave lesser reconnections on an average (clearly seen in Figure 58c & d for p=0,s=1). One would think that smaller cell size should be used only if the node mobility is low. The reason being with high mobility, nodes will constantly cross-cell boundaries triggering frequent location updates and hence higher reconnections (re-organizing the overlay tree). However, our simulation results were a little counter-intuitive. Our simulation results indicate that for lower speeds (1m/s), the number of reconnections starts at a lower values and keeps increasing until the cell area is reaches about 100x100m^2. Beyond 100x100m^2, the average reconnection number starts to fall until it reaches the lowest at 250x250m^2 (as expected). This anomalous behavior can be explained with the help of Figure 59. The source node maintains a CID for every member node in its tree. When a node moves to a new cell, it reports the new CID to the source. In our design, the source assumes the center of the new cell as the new location of the recently moved member node. This is where the problem lies. The source node periodically re-computes the overlay tree in order to compensate for node movement and their new position. With small size cells, the distance between the centers of neighboring cells is very small as compared to the one in a larger size cells. As a result, when the tree is rebuilt, smaller cell will not see many *reconnections*. However, in case of large cells, nodes that have just crossed a cell boundary would report a major change in their location causing several updates. We ran simulations for update time of 20, 30, and 60 sec. For a much larger cell size (beyond 100x100m^2), the shorter update interval and lower speeds is not enough for nodes to travel across boundaries and so the number of reconnections starts to go down. This is clearly seen in case of 20sec and 30sec update (Figure 58a & b) where the no. of reconnections starts to go down beyond 100x100m^2 while in case of 60sec, the curve starts to fall around 125x125m^2

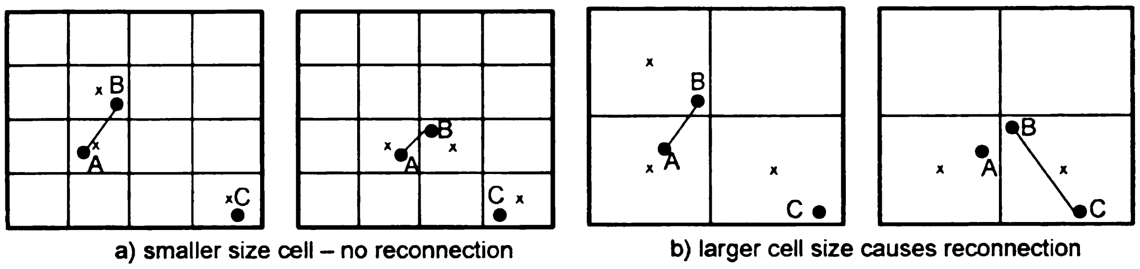

B
A
C
B
A
C
B
A
C
B
A
C
a) smaller size cell – no reconnection
b) larger cell size causes reconnection

Figure 59: Effect of larger cell size on connection pattern.

5.5 Contributions

This chapter presents SOLONet, a design to build sub-optimal overlay multicast trees, which tries to strike a balance between the advantages of using location information for building efficient overlay multicast trees versus the cost of maintaining and distributing location information of every member nodes. SOLONet eliminates the need to do expensive location broadcast for each node and doesn't require each node to know the location of every other node. The SOLONet architecture partitions the physical topology into smaller cells. By having a local leader in each cell, the system is able to localize several tasks and aid in the service discovery mechanism. The dissertation gives a detailed description of the service discovery mechanism, local leader selection process and the use of a beacon message for carrying out various activities.

Simulation results show that SOLONet is scalable and its performance closely matches that of an optimal overlay multicast tree. Detailed analysis of SOLONet's performance (with different cell sizes) was conducted. Effect of smaller cell size was also examined. The current simulations considered square cells; future versions of SOLONet will be tested with other shapes and the effect on performance.

6 Uniform Resource Allocation in Multicast

Implementing multicast in MANETs is a challenging task. A typical multicast network consists of a single tree, in which only a few internal nodes contribute most resources and are involved in performing the multicast functionality. This leads to an uneven utilization of network resources. This problem is more prominent in MANETs where network resources are limited. A possible solution to the problem is to split the multicast content over a number of trees. Multiple trees provide several paths for the multicast content and get more nodes involved in implementing the multicast functionality. However, in this setup, not all the trees get to use the best weight edges, thus the overall multicast latency increases. This chapter presents MEST [70], a distributed algorithm to construct multiple edge-sharing trees for small group multicast. MEST balances the resource allocation and delay constraints by choosing to overlap certain edges that have low weights. The simulation results show that MEST is scalable and can generate multicast networks that have low delay and fair resource utilization.

6.1 Introduction

Similar to other multicast networks, overlay multicast also suffers from the resource utilization problem (also referred to as the fairness issue in some literature). This fairness issue results from the fact that in a typical multicast tree, only a small number of nodes and edges are actively involved in implementing the multicast functionality. For example, there are a few internal nodes that perform the task of duplicating and forwarding packets and a large number of leaf nodes that only act as receivers of packets. These leaf nodes do not contribute any resources to the multicast tree. MANETs are characterized by scarce network resources. Due to this uneven distribution, certain nodes will run out of resources (e.g. battery strength) faster than other nodes, leading to bottleneck nodes in the multicast trees. As explained by Wang and Gupta [90], the lifetime of a multicast tree depends on the lifetime of a bottleneck node. Proper resource allocation can help better utilize the available resource and extend the life of an overlay multicast network. The delay in a multicast tree is equal to the time it takes for all the leaf nodes to receive the packets (data) sent by the source node. There are two main techniques that can help reduce this delay: source eccentricity and tree weight. The first approach tries to reduce the source node's eccentricity. (The eccentricity of a node v in a connected graph G is length of the longest of all the shortest paths between v and every other point in G). A higher fan-out at interior nodes results in a low depth tree (low source eccentricity), thus having a shorter delay. However, such a tree would have a large number of leaf nodes. For example, a binary tree has half the nodes as leaf nodes [60]. A tree with an average fan-out of 16 will be shorter (lower delay) but will have 10% internal nodes that perform the multicast functionality. Thus, it is necessary to have a well constructed overlay multicast tree which would maintain a good balance between the multicast latency and utilization of network resources. There are many distributed algorithms [18, 24, 47, 62] for generating shortest path trees; however, most of them pay little attention to the network utilization issues. The other method for reducing delay is by constructing low weight trees. (The weight of a tree is the sum of weights of its edges.). Low weight trees tend to have lower delays as they try to use low weight edges during their construction. In this dissertation, we concentrate on the second approach to reduce delay: constructing multiple low weight multicast trees.

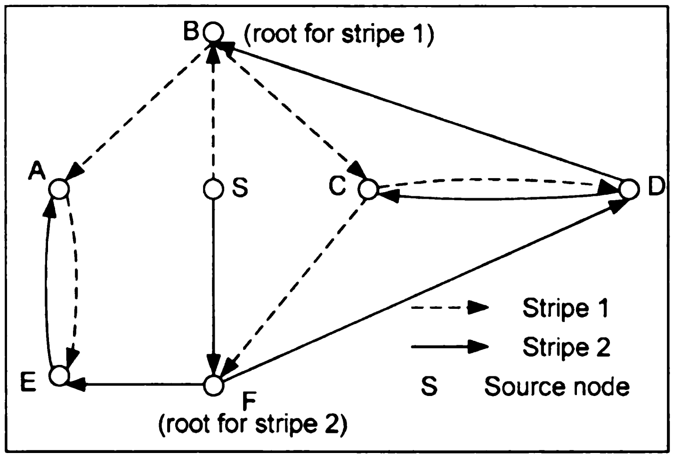


Figure 60: Multicast content split into smaller pieces and sent over multiple trees.

This dissertation, presents MEST – a distributed algorithm for building multiple edge-sharing multicast trees. MEST aims to uniformly distribute the multicast functionality across all the nodes in the group with little change in the multicast delay. The MEST algorithm is designed to run on top of any distributed algorithm for constructing a minimum spanning tree of a given graph (e.g. [17, 37, 84]). MEST is described with respect to the GHS algorithm [37]. An interesting point to note here is that the concept of MEST can be applied to any kind of multicast network. Although this dissertation describes MEST with respect to overlay multicast, MEST algorithm can be used to improve the resource allocation (as well as keep the delay under control) in any wired multicast network or IP-based multicast in MANETS. MEST constructs several multicast trees in the same network. Each tree constitutes a separate sub-group and all the nodes participating in the multicast subscribe these sub-groups. The multicast content is split into smaller stripes and each stripe is then sent over one of the multicast trees (Figure 60). (Note that we chose to use the word stripes instead of fragments, to avoid confusion with 'fragment of a graph'). Several papers propose the use of multiple multicast trees for mesh creation in order to have redundant links to improve reliability. Although improving reliability is not MEST's primary objective, the MEST algorithm can also be used in building a reliable redundant mesh network. In such a network, the multicast content is not split into smaller stripes; instead, each MEST multicast tree will provide an alternative path to the multicast data. If one path fails, the nodes re-route the data over one of the other alternative paths.

Several simulations were carried out to examine the scalability and performance of MEST. We also simulated many variants of MEST in an attempt to find the one that gives the best performance improvement. Overall, our simulation results indicate that MEST is highly scalable and has a lower delay as compared to a single tree multicast. We were also able to determine the best overlap threshold value for a given scenario. The simulation results also show significant reduction in the delay as the number of trees increase. However, we strongly believe that there would be other factors (like file fragmentation overhead) that need close examination in order to check if they offset this performance improvement. It was observed that a variant of MEST – variable file splitting – showed lower multicast latency as compared to basic MEST.

6.2 GHS Algorithm

Gallagher, Humblet, and Spira (GHS) [37] were one of the first to provide a solution to the problem of constructing a minimum spanning tree in a disturbed manner. The following two sub-sections give a brief description of their algorithm and the message exchange between nodes. Interested readers should refer to the GHS paper [37] for further details.

6.2.1 Overview

The GHS protocol works by combining fragments (disconnected components of a graph) along the shortest edge joining them. The protocol maintains a forest of trees, each identified by its fragment id (*fragid*) which consists of the fragment's level number and fragment's core node's id. At start, all the nodes are at level 0 and every node is an individual fragment. Each fragment attempts to asynchronously find a minimum weight out going edge (*mwoe*) into another fragment. When such an edge is found the two fragments exchange '*connect*' messages and attempt to combine to form a larger component. As fragments join, the level of the combined component increases by one (if certain conditions are satisfied). There are a few rules that need to be followed in order for this merger (and level increase) to happen:

1. Suppose a fragment F_l at level m (m>0) encounters a *connect* message from a smaller fragment F_s at level m-p (p<m) at the other side of the *mwoe*, the smaller fragment is absorbed as a part of the larger fragment. The level and *fragid* of the combined larger fragment stays unchanged i.e. F_l at m.

2. If the two fragments are at the same level m and have the same *mwoe*, then they combine to form a larger fragment at level m+1. The *mwoe* edge, which joined the two fragments, is then termed as the 'core' of the fragment.
3. On the other hand, if a fragment at level m encounters a larger fragment (at level x; x>m) along it's *mwoe*, it delays it's *connect* message until it reaches a level at least equal to x.

The rules for combining fragments ensure that a new fragment at level m+1 will be formed by the combination of at least two fragments at level m. Thus, a fragment at level p will contain at least 2^p nodes. Therefore, $\log_2 N$ is an upper bound on the fragment levels. The wait in rule 3 is essential since communication latency required to find it *mwoe* is proportional to the fragment size. If this delay is not implemented, it may result in a loop. An example is shown in Figure 61. When the two fragments combine, the nodes in these fragments (other than the core nodes) are not immediately informed about the identity and level of the new fragment. In this example, let's say F and F' (both at the same level) combine along their *mwoe* (in this case PQ). Now the new fragment is one level higher. The *fragid* has changed to the new *mwoe*. Now, P and Q know about the new level and *fragid*. However, there are several other nodes in each fragment (e.g. A and B) who won't know about this change until they get some message (e.g. *initiate* message) from the P or Q. At some point node A gets an initiate message from P and it updates its level. Node B on the other hand still has the old id and level. If the wait in the third rule is not implemented, node B might send a *connect* message to A thus causing a loop. The wait ensures that B waits till its level increases at least to that of A. When that happens, B realizes that they belong to the same component.

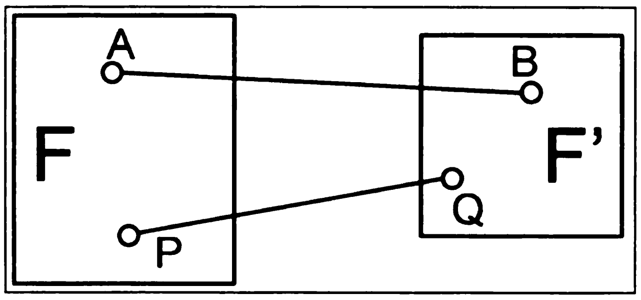


Figure 61: Example of a delayed merge.

6.2.2 GHS Example

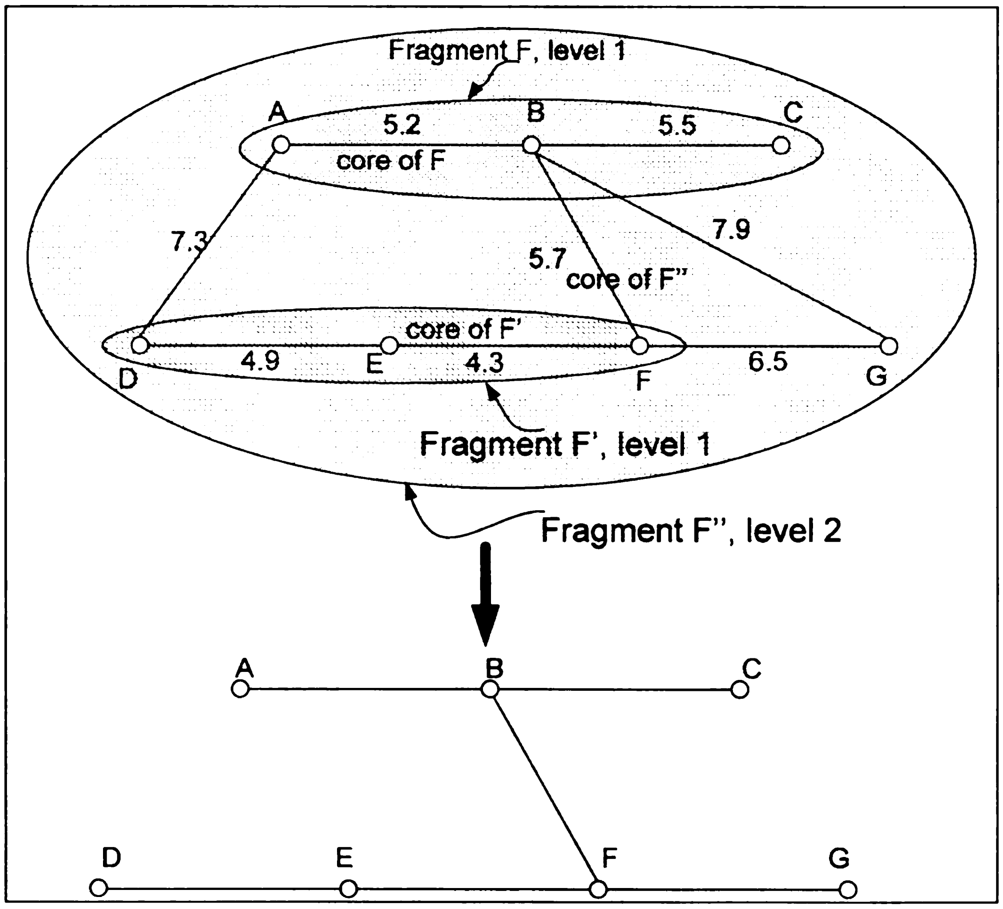


Figure 62: GHS Example.

As an example of the GHS algorithm, consider Figure 62. At t=0, all nodes are at level 0. At some later time, nodes A and B merge to form a larger fragment at level 1. Similarly, nodes E and F combine on their *mwoe* to form F' at level 1. Further ahead, node C and node D get absorbed into F and F' respectively. After some more time, the two fragments merge on the edge BF to form a larger fragment F'' with level 2. At a later point in time, node G gets absorbed into this larger fragment.

6.3 MEST Algorithm

Our MEST algorithm can run on top of any distributed algorithm for constructing a minimum spanning trees of a given graph (e.g. Async [84], Awerbuch [17], GHS [37]). In this dissertation, however, we describe MEST with respect to the GHS algorithm. MEST can be considered as several instances of the GHS algorithm running in a sequence one after the other. At start, all the nodes run the GHS algorithm to produce the first tree (T_1), then the second run gives tree T_2 and so on, till all the desired number of multicast trees have been generated.

6.3.1 Preliminaries

In case of multiple multicast trees, the highest weight multicast tree contributes the most delay and so, this (highest-weight) tree determines the delay of the overall system. Since edge disjoint multicast trees would use different edges in each tree, ideally, they would do a better job in solving the resource allocation problem. However, if edge disjoint trees are not constructed correctly, they may not be the best choice when it comes to delay constraints. An example is shown in Figure 63. For the given graph, it is possible to find many pairs of edge disjoint spanning trees. Note that each multicast tree has to be a spanning tree since it has to cover all the multicast member nodes. Two such pairs are shown (Figure 63b & 4c). In Figure 63b, T1 is a minimum spanning tree. However, other multicast tree (T_2) has a very high latency since it is using all the high weight edges that were rejected by the minimum spanning tree (T_1). With the two trees, the network resources are evenly used; however the overall multicast delay would be determined by the slowest amongst the two trees (T_2). Unlike Figure 63b, in case of Figure 63c, the light weight edges are equally distributed between the two spanning trees and hence the overall delay is lower compared to the earlier case.

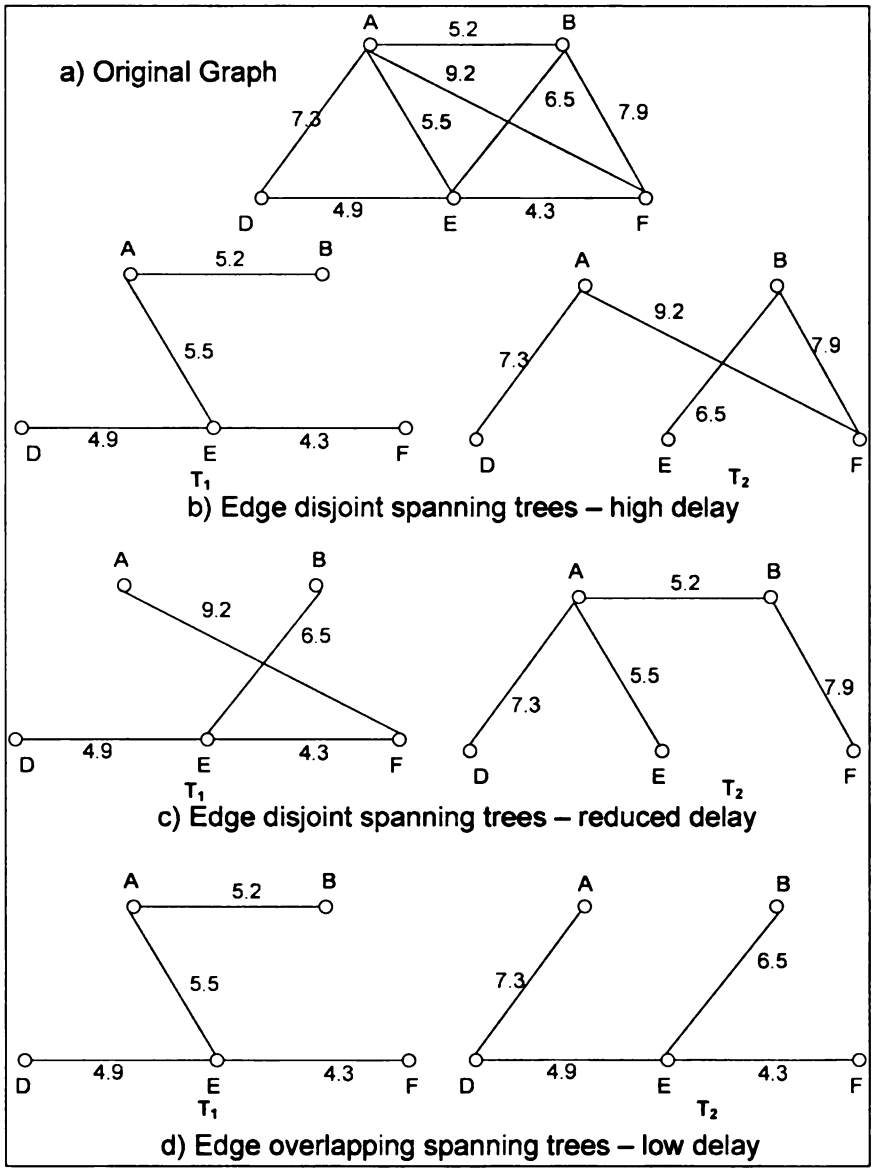

a) Original Graph
A
B
D
E
F
5.2
9.2
7.3
5.5
6.5
7.9
4.9
4.3
T1
T2
b) Edge disjoint spanning trees – high delay
c) Edge disjoint spanning trees – reduced delay
d) Edge overlapping spanning trees – low delay

Figure 63: Edge disjoint vs. edge overlapping trees

Now consider the case where two trees can share an edge. Figure 63d shows two spanning trees (one of them minimum) that have two common edges (DE and EF). Such a multicast tree may be obtained by having a condition that allows the overlap of edges that satisfy a certain condition. For example, edges having weight below a certain threshold or edges connected to nodes that have very low degree. This threshold should be wisely chosen. A value too high would lead to several overlapping edges, decreasing the network utilization and a value too low would increase the multicast latency. In the example, we choose the cut-off threshold for edge sharing as 5. In Section 6.4, we show the best overlapping threshold for a particular scenario.

6.3.1.1 Other factors affecting the delay

In case of edge-sharing trees, a factor that might affect the delay is the number of times an edge is re-used. This is important because a shared edge has to carry packets for different multicast groups since it is part of several multicast trees. For example, in case of Figure 63d, the two common edges (DE and EF) have to carry packets for both the trees. A packet would be queued at a node if that node is currently sending or receiving another packet over the common edge. Depending on the queue length, this queuing might result in additional delay. In case of non-streaming applications, a packet would be queued only if an overlapping edge is at the same delay-distance from the source node in multiple trees. In case of streaming applications, this delay would always be present since there is constant stream of data to be sent by the node. In order to eliminate any further delays due to queuing, we suggest setting an upper bound on the number of trees that can share a particular low-weight edge.

6.3.1.2 Special cases

A bridge is an edge of a graph whose removal will result in a disconnected graph. For graphs that contain bridge edges, it is not possible to have multiple trees that are

completely edge disjoint. Figure 64 shows such an example. In the example, edge BG is a bridge edge and will be present in any spanning tree of this graph. In our MEST algorithm, we mark a bridge edge with the 'S' tag to indicate that this is a special case edge, which has to be included in all the spanning trees.

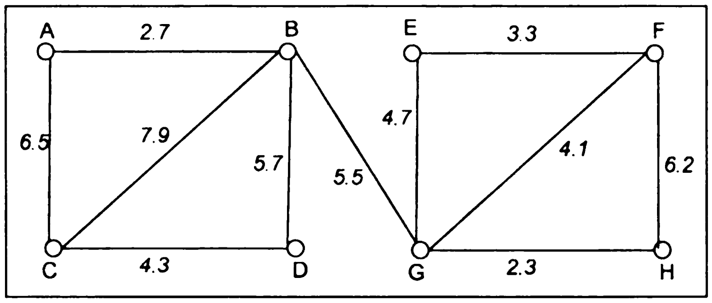


Figure 64: Example of Bridge edge.

6.3.2 Basic Operation

Let's say we want to construct 'm' multicast trees ($T_1, T_2, \ldots T_m$) which may have some overlapping edges. Since each node is running an instance of the GHS algorithm, at t=0 there are 'm x n' fragments. Each node maintains a variable called the *fragid*, which stores the fragment id (*fragid*) of the components that the node is currently associated with. Nodes also maintain separate arrays (of length = 'm') for each of its edges. The 'm' elements of the array correspond to the 'm' trees that the edge can potentially be part of. An element of the edge array can have one the following values:

- *'B' = Edge forms a branch in the current tree.*
- *'R' = Rejected edge - edge was by the current tree since it joins another node of the same fragment.*
- *'U' = Usable edge (Basic edge). This edge will be checked to see if it's the best edge for this node.*

- *'X' = Edge was used by an earlier tree. An edge marked as X will not be used in current tree.*
- *'S' = Usable edge (Special case – bridge edge)*

At start, edge arrays at each node are initialized so that every element in it is marked as 'U'. This allows the first run of GHS to choose any edge it wants. Thus in a way, the first tree is a true MST for that graph. During the formation of the kth tree, a node will mark the kth element of an edge array as 'B' if the corresponding edge is being used (as a branch) in that tree. If certain conditions are met, then the node marks the k+1th and higher element of that edge as 'X'. This is done so that that edge is not reused in later trees. When a node is in the nth (n>k) GHS execution (formation of T_n), it will use an edge only if it finds that the nth element of that edge array marked as 'U'. As explained earlier, certain edges are allowed to overlap, in which case, the higher elements of those edges are not marked as 'X'. As the algorithm progresses more trees are found and some of the previously used edges (elements in the edge array) are marked as 'X', making them unavailable in subsequent trees. There are two rule that determine which used edge not to be marked as 'X':

i. If the edge weight is below the minimum weight threshold, it may be shared by more than one multicast tree. In this case, a node will not mark higher elements of a recently used edge as unavailable ('X') for subsequent trees. Instead, these elements will remain as 'U' thus making them available to the following GHS runs. However, if there is a bound on the number if edges that can be shared (for example a 3-edge overlapping multiple multicast trees), then a node can mark higher elements in the array as 'X' if this bound is reached.

ii. If the edge is a bridge connecting two components of the graph (e.g. Figure 64), this edge will be shared amongst all the multicast trees. A bridge edge is hard to detect. Section 6.3.2.1 explains how this condition is detected and handled.

Figure 65 shows a pseudo code-snippet showing how edges are marked at each node (in each run). In order to lower queuing delay on shared edges (and also to improve resource sharing), nodes in MEST will not re-use an edge if they have the option of choosing an un-used edge that falls below the threshold. This also gives rise to the possibility that the new trees that are generated are edge-disjoint. For example, if the cut-off threshold in Figure 63 was set to 6, MEST could generate edge-disjoint trees similar to Figure 63c.

```
//mark each edge for later trees to use/discard.
void mark_edges(int nd, int curr_tree, int nd_ed_degree){
 //mark the node's currently used edge as a branch!!
 gph_nd[nd].edge_tag[curr_tree][nd_ed_degree] = 'B';
 //check the degree of the current node.
 if( gph_nd[nd].degree > 1 ){
  //check the rest of the edges!!
  for ( int t=curr_tree+1; t<TREES; t++ ){
   //If the edge weight is above the threshold, mark it as un-usable
   if( gph_nd[nd].distance[nd_ed_degree] > THRESHOLD ){
      gph_nd[nd].edge_tag[t][nd_ed_degree] = 'X';
   }
   else {
      //If below the threshold, mark as usable
      gph_nd[nd].edge_tag[t][nd_ed_degree] = 'U';
   }
  }
 }
 else {
  //If it's the only edge, mark its array as special!!
   for ( int t=curr_tree+1; t<TREES; t++ ){
    gph_nd[nd].edge_tag[t][nd_ed_degree] = 'S';
   }
 }
}
```

Figure 65: Pseudo code for edge marking.

There are no bounds on the number of trees that can be generated using MEST for implementing a particular multicast. The algorithm tries its best to select different set of edges in every iteration (each tree). However, it should be noted that beyond a certain point, there is no additional improvement in the resource utilization since all the edges have been used in at least one tree. Beyond a certain point, additional trees will be reusing edges used in at least one tree generated in an earlier iteration. This could possibly increasing the delay due to congestion along the edges that are most used.

6.3.2.1 MEST Messages

Most of the message passing between the nodes is similar to the one used in GHS algorithm except for some minor modifications as explained here. Nodes in GHS belong to one of the three states: *Sleeping* state, *Find* state and *Found* state. A node is in the *Sleeping* state until it starts executing the protocol or is awaken by a message from another node. The two nodes adjacent to the core change their state to *Find* and start a new iteration by broadcasting an *initiate* message along the branches of the fragment. Upon receiving the *initiate* message, a node enters the *Find* state and forwards the *initiate* message to its neighbors (in the fragment). The *initiate* message thus propagates to all the nodes in the fragment. The *initiate* message contains the *fragid* and the fragment level. A node in *Find* mode updates its *fragid* and level number to the one in the *initiate* message. Next, the node picks its minimum weight edge incident on a node in another fragment. To check if an edge is going to a node in a different fragment, a node sends a *test* message (containing the node's *fragid* and level number) across the edge. A node receiving a *test* message responds back with an 'accept' or a 'reject' message depending on whether it belongs to the same or different fragment as the sending node.

Once a node identifies its 'best edge', it sends a *report* message towards the fragment's core. This report message contains the id and weight of its best edge. Interior nodes hold back their *report* message until they receive reports from all of its child nodes. The internal node's report contains the smallest of its best edge or the outbound fragment branch on which has the minimum best edge was found. There are two interesting things to note. If a node is a leaf node, then its *report* message contains the basic edge weight as infinity. If no node has outgoing edge, then the algorithm is complete and the fragment is an MST of the graph. After the two nodes adjacent to the core have exchanged the report messages, the *mwoe* for that iteration is determined. A *Change-core* message is then sent over the branches of the fragment to indicate the new core. A connect message is exchanged by the two fragments over the *mwoe* before moving to the next level.

In the original GHS algorithm, a single degree node responds back with a *report* message containing the 'best edge' weight as infinity. In GHS, the node's parent doesn't do much with this information since its primary interest is in finding the smallest of its base edge or outgoing branch containing the smallest edge. In MEST, however, a node is able to examine only the edges that are marked usable (i.e. 'U'). A node may finds that it is a single degree node for a particular run of GHS. This may be true for other trees (i.e. there might be other edges connecting that node to the rest of the graph). In MEST, a node will send negative infinity in its *report* message if it detects that it truly is a single degree node for the entire graph. Since there is only one edge connecting it to the rest of the graph, this edge has to be included in all the trees. When the node receives a negative infinity in a report from its child node, it will not mark all the elements in the edge array as 'S'; thus making it available to subsequent trees.

6.3.2.2 Bridge Condition

In MEST, we introduce a fourth state called *Completed* state. This state is reached when a GHS run terminates, i.e., when no node has any outgoing edge for that fragment. In plain GHS, this condition is reached when all the nodes have been found (i.e. a multicast tree for the graph has been found). In MEST however, since not all edges get examined, bridge edges might be missed leading to one or more fragments that have reached the *Completed* state. When a fragment attains the *Completed* state, the nodes adjacent to the core broadcast an *override* message along the branches of the fragment. Upon receiving an *override* message, a node checks to see if any of its unavailable edges connect to a different fragment. This check is carried out in the same manner as in GHS - by sending a *test* message (containing the node's *fragid* and level number) across the edge. Upon receiving a *test* message, the node on the other side replies with a reject (same fragment) or accept (different fragment) message. An *override* message from the core is replied back with an *override response* message. When the node receives an accept message from the other side, it sends the weight of the outgoing edge along with the response. In case of a reject message, it replies with infinity as the weight. If the core nodes receive several override response messages, they pick the edge that has the smallest weight.

6.3.2.3 MEST Example

We illustrate the working of MEST by using Figure 66 (slight modification of Figure 63) as an example. The weight threshold for reusing edges is set to 5. At the end of the first GHS run, the edge array entries at various nodes are as shown in Table 16. Edge BC is marked special, while edges DE and EF are marked re-usable for the second tree since they fall below the threshold. Table 17 shows the array entries after the second tree has been constructed. Although this is a simple example showing two multicast trees, in real network scenarios we may have a more complex graph with higher connectivity that gives several multicast trees.

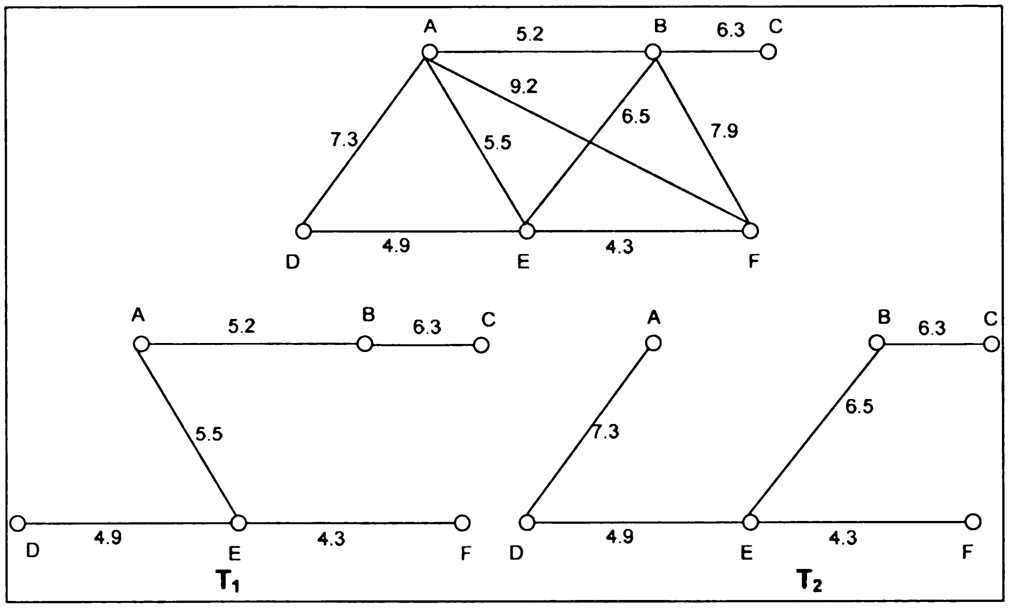


Figure 66: MEST example

Table 16: Edge arrays after T_1

Node	Edge Arrays			
A	E_B[B,X]	E_D[R,U]	E_E[B,X]	E_F[R,U]
B	E_A[B,X]	E_C[B,S]	E_E[R,U]	E_F[R,U]
C	E_B[B,S]			
D	E_A[R,U]	E_E[B,U]		
E	E_A[B,X]	E_B[R,U]	E_D[B,U]	E_F[B,U]
F	E_A[R,U]	E_B[R,U]	E_E[B,U]	

Table 17: Edge arrays after T_2

Node	Edge Arrays			
A	E_B[B,X]	E_D[R,B]	E_E[B,X]	E_F[R,R]
B	E_A[B,X]	E_C[B,B]	E_E[R,B]	E_F[R,R]
C	E_B[B,B]			
D	E_A[R,B]	E_E[B,B]		
E	E_A[B,X]	E_B[R,B]	E_D[B,B]	E_F[B,B]
F	E_A[R,R]	E_B[R,R]	E_E[B,B]	

6.3.2.4 Complexity Analysis

The GHS algorithm has a message complexity of O(E + N log N). Our MEST algorithm is essentially 'm' repetitions of the GHS algorithm except for the last step

(*Completely state*). During a particular GHS run, when each fragment reaches the *Completed state*, the complexity for each fragment is the same as GHS: $O(E_i + N_i \log N_i)$, where E_i (and N_i) are the number of edges (and nodes) in fragment i. For a particular run, these fragments merge along their shortest outgoing edges, ultimately forming one fragment, which is the spanning tree (for the original graph), generated for that run. It should be noted that:

$$\sum E_i = E_{total} - no.\ of fragments - 1$$

During the *Completed state*, the following messages are generated in each fragment: Override, Override Response, Test, Reject, Connect, and Change-core. Therefore, the total number of messages is proportional to *number of fragments – 1* (i.e., one less than the total fragments in the completed state – last fragment does not need to exchange any messages, it is the final tree). Thus, the overall complexity of the MEST algorithm is 'm' times that of GHS. Now, since the number of MEST trees ('m') is much less than the total number of nodes in the topology ('N'), the complexity of MEST can be expressed as O(E + N log N), same as that for GHS. The complexity of MEST can be reduced by choosing to run it over Awerbuch [17], which has a better complexity compared to GHS. Other complexity optimization techniques like the ones mentioned in Michalis [36] or Nancy [59] can be applied to improve the complexity of MEST.

6.4 Simulations

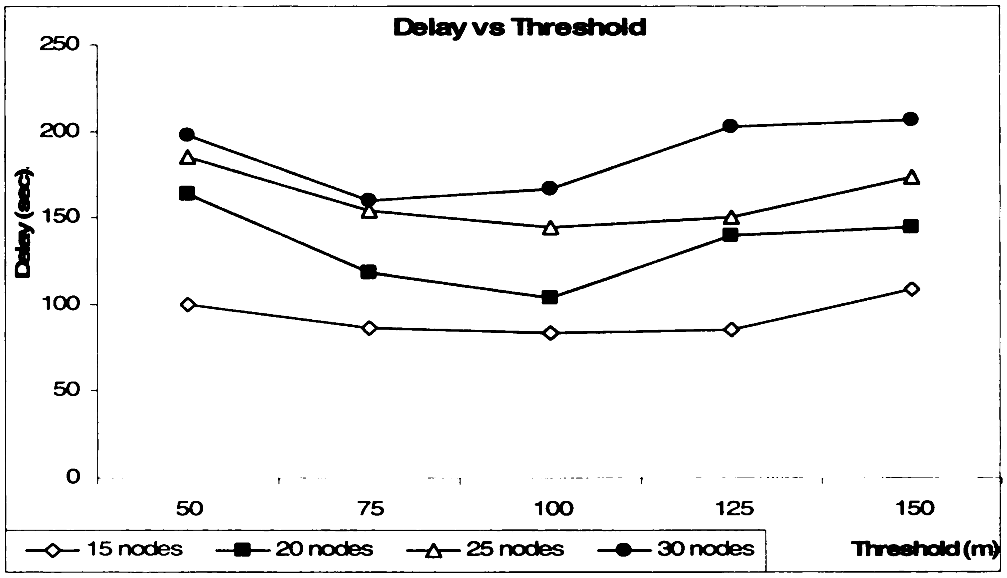
Simulations were carried out using Network Simulation (ns2.26) [8]. As of this writing, ns2 does not have any extension for simulating overlay multicast in MANETs. With the help of C-programming and bash scripting, the traffic pattern generated by CMU's *cbrgen* utility was modified to represent an overlay network. Additional modules were written to simulate our MEST algorithm (for multiple trees) and the Prim's algorithm [32] (for generating a single MST). In all the result graphs, MST corresponds to the results for a single tree multicast in which the multicast spanning tree is a minimum spanning tree built using Prim's algorithm. The *setdest* utility was used to generate different node positions and movement patterns. The nodes in the simulation move according to the 'random waypoint' model [51]. Table 18 shows the default simulation parameters. In the following sub-sections, simulation parameters when not mentioned follow the one given in Table 18. All the simulation results are average of 25 random scenarios.

Table 18: Simulation Parameters

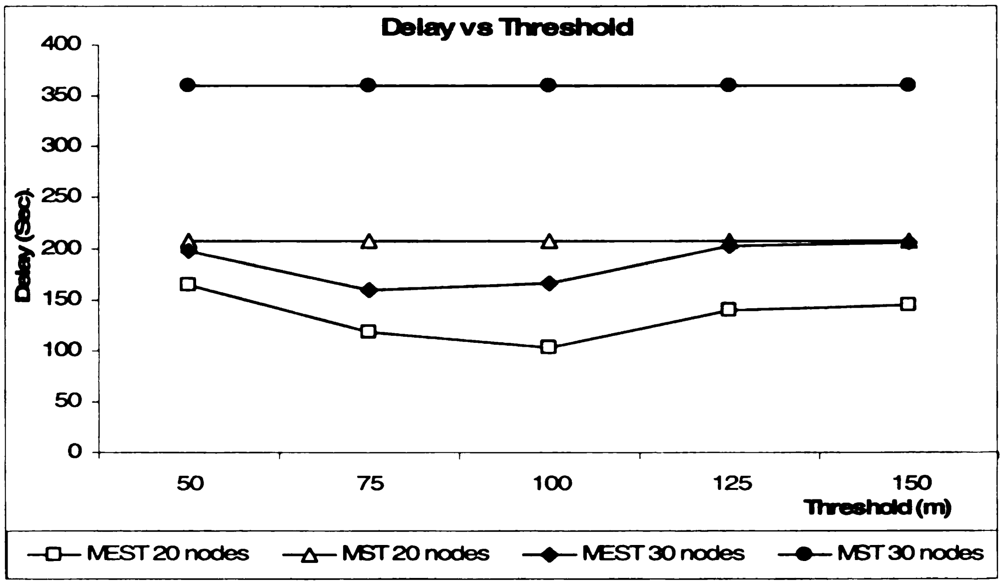
Simulation Area	500x500 m^2
Total number of nodes	150
Number of member nodes	15, 20, 25, 30
Packet size	512 bytes
Total File size	512000 bytes
Ad-hoc Routing Protocol	DSR [51]
Pause Time	10 sec
Max speed	5 m/sec

6.4.1 Threshold vs Latency/Leaves

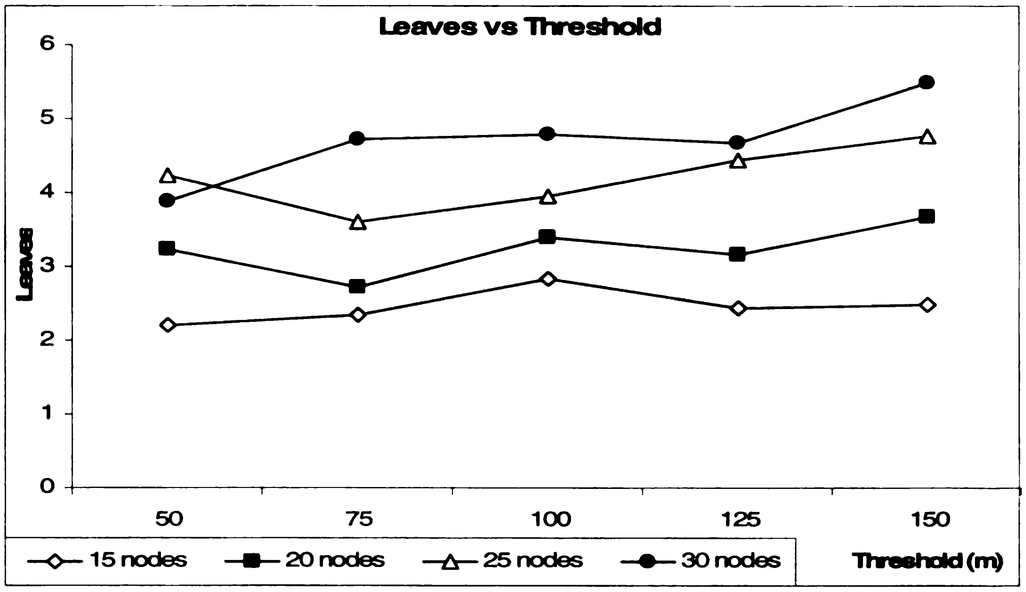
Simulation results from Figure 67 show that there is a significant reduction in the multicast latency along with better utilization (fewer leaves) of the available network resources. Part of this improvement can be attributed to the parallel transmission over multiple trees. A close examination reveals that for this set, the optimal threshold value is 75m. At 75m, the number of leaves in the overall multicast was minimum while the delay was near minimum. Another observation was that the performance curves tend to 'flatten-out' for higher threshold values. This is because at higher thresholds, more edges are allowed to overlap. When an edge is used in multiple trees, it encounters congestion delays since it has to handle packets for multiple trees. This delay is insignificant when the edge weight is small. However, when the threshold is high, longer edges are also allowed to overlap. In such cases, the congestion delays are comparable with the edge delays and hence, for higher threshold, the delay starts to increase.



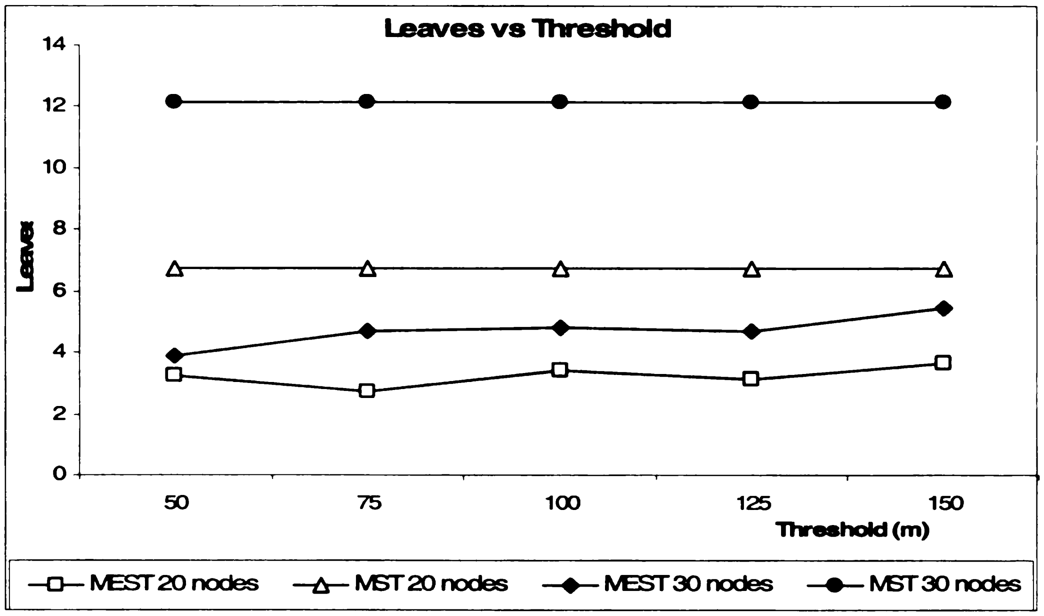
a) Showing latency for all node sizes



b) Comparing latency of MEST with MST



c) Showing no. of leaves for all node sizes



d) Comparison between MEST and MST

Figure 67: Effect of threshold on delay and utilization.

In the worst case, when the delay is set to infinity, all the MEST trees would correspond to the same tree. We conducted statistical (t-test with significance of 0.005) analysis on the simulation data in order to compare the performance of MEST w.r.t a multicast that makes use of a single tree (e.g. MST). Table 19 shows the t-test result (for significance value of 0.005). The tabulated results verify that there is an improvement in the performance of MEST compared to MST.

Table 19: t-Test table for completion time for 3-tree MEST

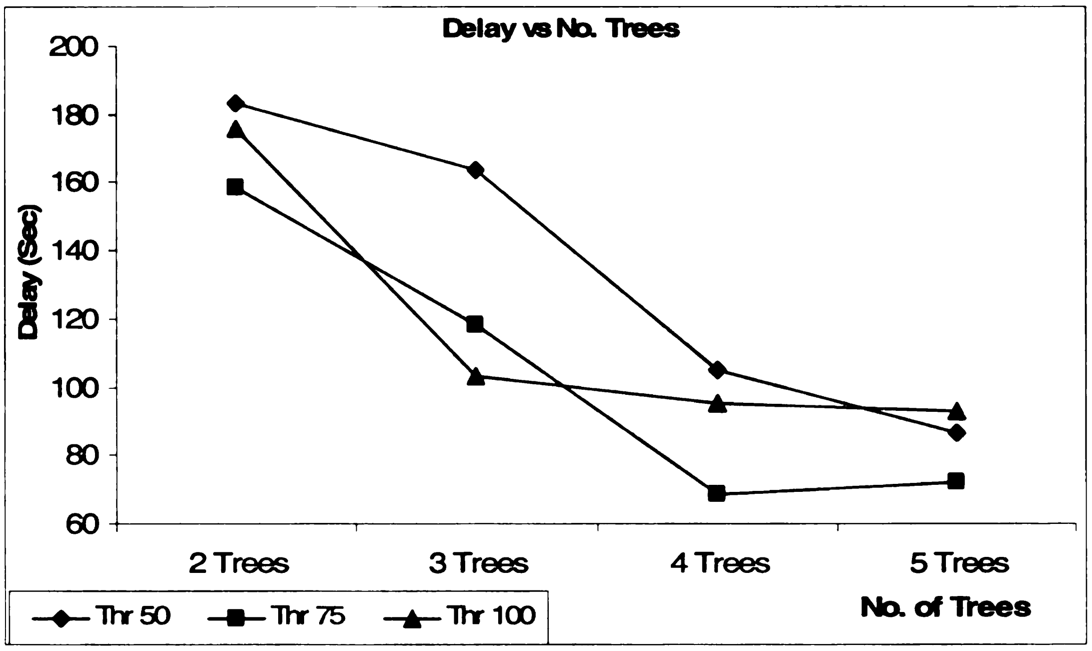
No. of nodes \ Threshold		50	75	100	125	150
	p-value					
15	1-tail	2.61551 e-05	4.32448 e-07	2.77906 e-07	2.50536 e-06	6.52193 e-05
	2-tail	5.23101 e-05	8.64896 e-07	5.55812 e-07	5.01073 e-06	0.000130439
20	1-tail	0.005290438	3.46025 e-07	1.92643 e-08	0.000158717	0.000669583
	2-tail	0.010580875	6.92051 e-07	3.85287 e-08	0.000317433	0.001339165
25	1-tail	0.000710022	2.00695 e-06	8.81065 e-09	3.88618 e-08	2.96496 e-05
	2-tail	0.001420045	4.0139 e-06	1.76213 e-08	7.77237 e-08	5.92991 e-05
30	1-tail	9.96448 e-06	8.07021 e-14	4.27899 e-11	5.76431 e-07	4.55218 e-06
	2-tail	1.9929 e-05	1.61404 e-13	8.55798 e-11	1.15286 e-06	9.10436 e-06

6.4.2 Multiple Trees with Un-even File Splitting

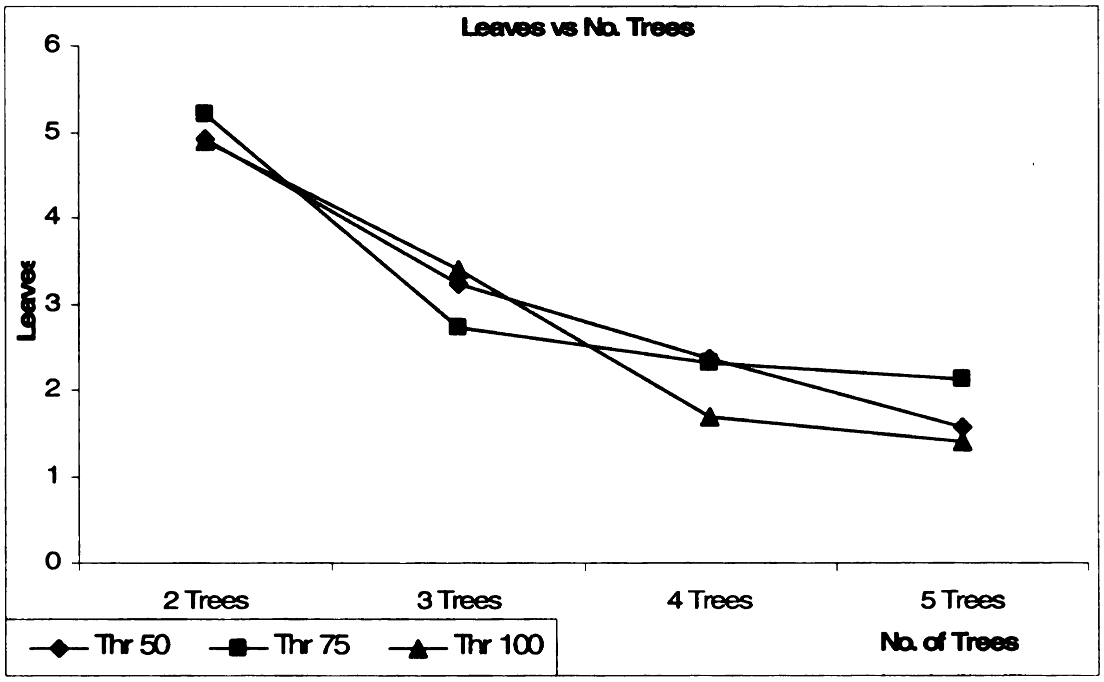
In this round of simulations, we split the multicast content over 2, 3, 4, 5 trees respectively. Figure 68 shows the results for this set of simulations. As the number of trees increased, the multicast latency showed significant improvement along with decreasing number of leaves. We also noticed that the gain in performance is not proportional to the number of trees over which the content is being divided. This is because each tree has a different weight. For example, a lets say we want to transfer 1Mb of data. Using 2 trees, the content would be divided as 500Kb each. For 4 trees its 250Kb each, so on. Since each tree is of a different weight (delay), each tree takes a different time to complete the transfer. The slowest tree decides the total delay. Table 20 shows the t-test (at significance 0.005) result for this set of simulation. The results show an increasing difference in the performance as we move from 2 to 5 trees – $2 < 3 < 4 < 5$.

In order to overcome the problem of variable tree delays, we carried another set of simulations where the file was un-evenly split. Going back to the same example, in case of a two tree multicast let's say for simplicity sake if the tree weights are in the ratio

45:55, we would divide the file as 450Kb and 550Kb each. Similarly in case of 4 trees, if the ratios are: 20:22:26:32, the file would be split as: 200kB, 220Kb, 260Kb and 320Kb. The results of this experiment are showing Figure 69. We saw an improvement in the performance for all the cases. However, this improvement should be treated with caution.



a) Delay Analysis



b) No. of Leaves

Figure 68: Effect of increasing number of trees

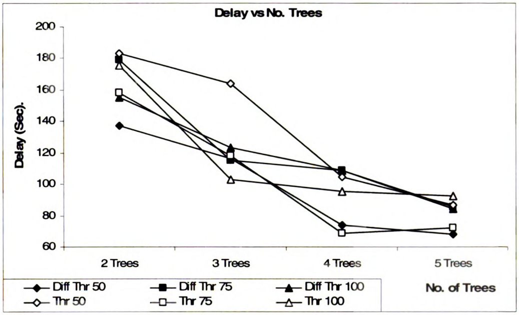


Figure 69: Splitting the multicast content un-evenly.

Table 20: t-Test table for completion time for n-tree MEST

Threshold / No. of nodes		50	75	100
	p-value			
2	1-tail	0.008308716	6.13886 e-05	0.004549551
	2-tail	0.016617431	0.000122777	0.009099101
3	1-tail	0.005290438	3.46025 e-07	1.92643 e-08
	2-tail	0.010580875	6.92051 e-07	3.85287 e-08
4	1-tail	4.94333 e-07	1.20172 e-09	6.47926 e-09
	2-tail	9.88665 e-07	2.40345 e-09	1.29585 e-08
5	1-tail	1.78182 e-09	4.39172 e-10	9.74222 e-09
	2-tail	3.56 e-09	8.78345 e-10	1.94844 e-08

In all our simulations, the file fragmentation and fragment sizes were pre-calculated. We feel that although the simulations showed significant improvement in the performance with increasing trees, in practical scenarios, fragmentation might play a role in determining the overall system performance. Each node has to manage fragmentation and re-assembly of packets. In addition, fragmentation is not a stateless process: nodes also have to store pieces of a file in their memory until they receive all the fragments for that file. As the number of trees increase, the overhead required to manage and maintain fragments increases. Multiple multicast trees would certainly improve latency and resource utilization. However, one should not be overzealous when implementing such a system. Beyond a certain point, the fragmentation overhead could start to overshadow the improvements from multiple trees.

We carried out a set of simulations where the file fragments were proportional to the eccentricity of each sub-tree. Figure 70 and Figure 71 show the delay and no of leaves respectively for such a multicast tree. We did not observe much difference in the performance when eccentricity was used as a parameter for dividing the fragments. This is probably because the sub-trees were built as low weight trees instead of low eccentricity trees. In the future, we plan to closely examine low eccentricity trees.

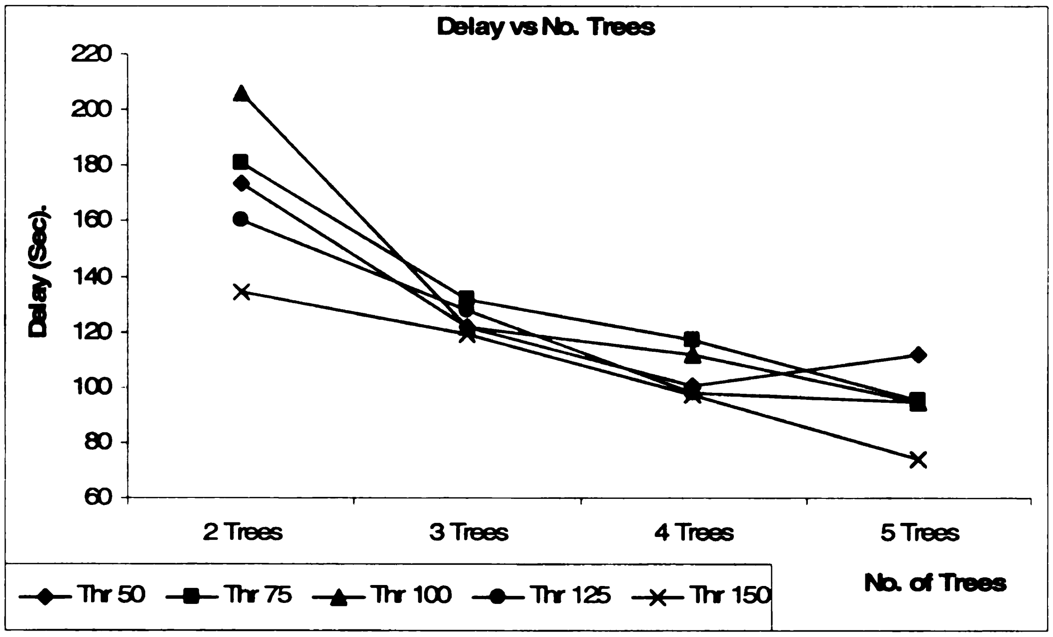


Figure 70: Delay when file fragments were proportional to eccentricity.

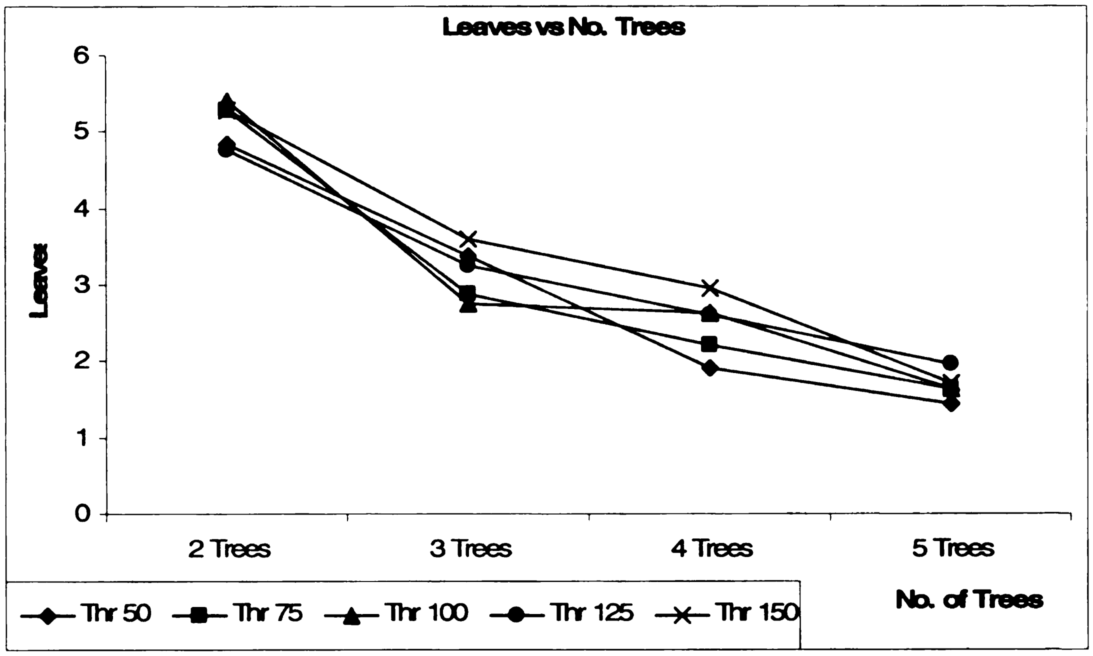


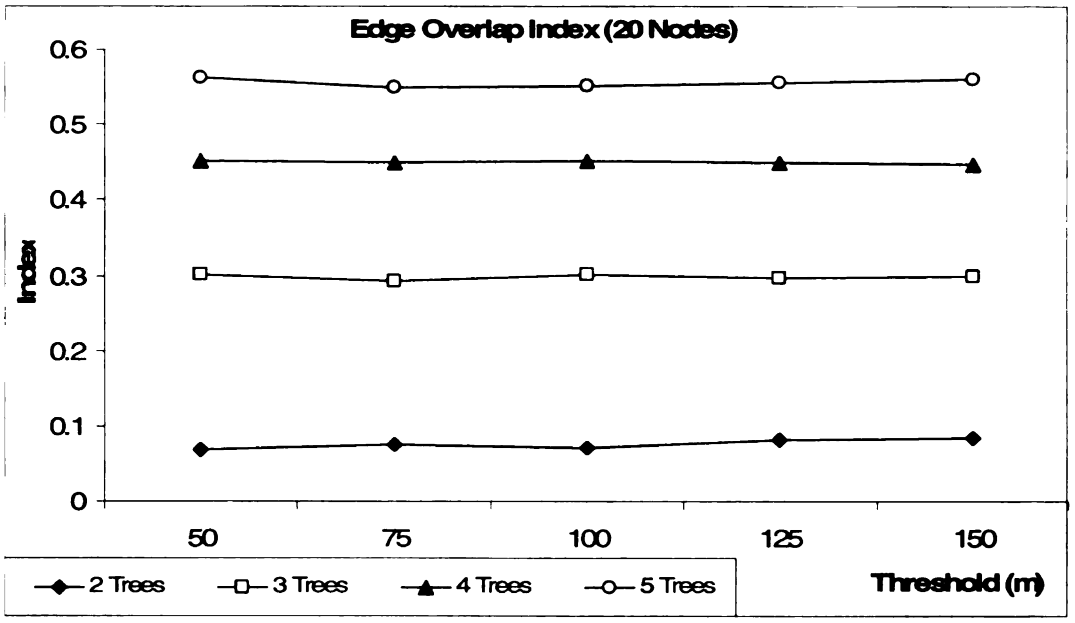
Figure 71: Leaves when file fragments were proportional to eccentricity.

6.4.3 Overlap Index

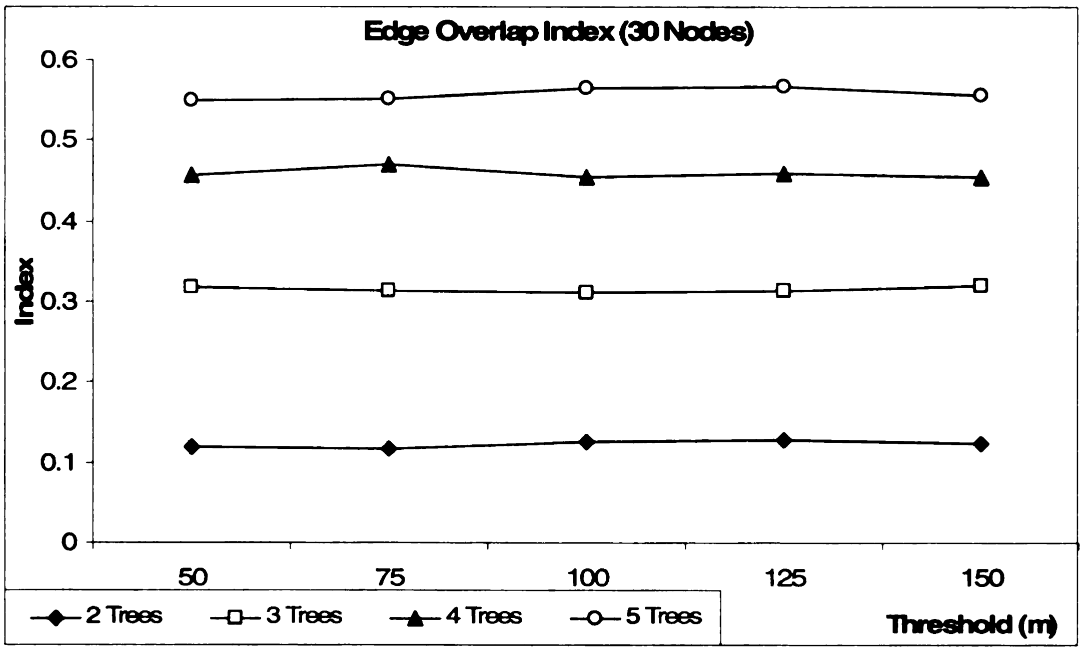
The overlap index is defined as the ratio of sum of overlapping edges in all the multicast trees to the sum of edges in all the multicast trees:

$$OI = \sum OE / \sum E_i$$

where, $\sum$OE gives the number of overlapping edges in all the trees, E_i is the number of edges in the ith multicast tree. As seen from Figure 72, the edge overlap increased with higher number of trees. This means that there would be more edges that are being re-used in multiple trees. This could potentially create congestion issues since the edge has to 'server' several trees. We also observed a narrow increase (barely visible in Figure 72) in the overlapping as the threshold value increases. This is expected as higher thresholds allow more edges to overlap. Also Figure 72a & b show that there is a slight increase in the overlap index as the number of member nodes increases.



a)



b)

Figure 72: Overlap index for multiple trees

6.5 Contributions

This section of the dissertation presents MEST, a distributed algorithm for building multiple multicast trees in ad hoc environment. MEST is easy to implement and can work with any distributed algorithm for generating minimum spanning trees. Simulation results show that MEST can significantly reduce the multicast delay and at the same time improve the network utilization. Although this dissertation describes MEST with reference to overlay multicast in MANETs, it should be noted however, that the concept of MEST can be easily extended to multicast in wired networks by considering stationary nodes or network layer (IP) multicast in MANETs. In addition, even though the design of MEST was not meant to solve reliability issues, it can also be used to build a reliable mesh network - in which case, the multicast data will not be split into smaller fragments.

7 Conclusion and Future Direction

The widespread deployment of IP multicast has been held back by a variety of issues. In recent years, several research groups have proposed the idea of overlay (application layer) multicast for MANETs. In such a multicast, the entire multicast functionality is implemented at the application layer. The application layer relies on the underlying unicast protocols to adapt to the changing network topology. As a result, the application layer has to track only the (multicast) group dynamic.

7.1 Summary

This dissertation proposes POMA – Prioritized Overlay Multicast in Ad-hoc wireless environment, a new infrastructure-less need-based prioritized overlay multicast model. POMA builds priority trees with certain nodes carrying important tasks in overlay networks, and rearranges low priority trees whenever some nodes temporarily move to a high priority network. The idea of building several role-based priority trees in the same environment can find many interesting applications. Simulation results with POMA also showed that low latency overlay networks can be designed by building trees that make use of location information.

One major issue with application layer multicast is that it is not as efficient as IP-based multicast. Data exchange between member nodes requires traversing other member nodes. The latency increases as the number of nodes increases. This delay is greatly reduced when the overlay tree is built by taking into account the member node positions. A location based tree would keep track of member node's movement and would be frequently updated to account for any change in the node positions. The nodes that are physically close to each other would be neighbors in the logical tree and the logical distance of any member node from the source node will be proportional to its actual distance from the source. However, there is a cost involved in using location information: nodes need to constantly update other nodes with their current position, the overlay tree needs to be re-built when location of nodes changes, etc.

This work presents SOLONet, a design to build sub-optimal overlay multicast trees, which tries to strike a balance between the advantages of using location information for building efficient overlay multicast trees versus the cost of maintaining and distributing location information of every member nodes. SOLONet eliminates the need to do expensive location broadcast for each node and doesn't require each node to know the location of every other node. The SOLONet architecture partitions the physical topology into smaller cells. By having a local leader in each cell, the system is able to localize several tasks and aid in the service discovery mechanism. The dissertation gives a detailed description of the service discovery mechanism, local leader selection process and the use of a beacon message for carrying out various activities. The dissertation also presents various methods of location positioning and tracking in indoor environment. It gives detailed description of three systems that differ in technology and method for carrying out location positioning.

Finally, the dissertation examines the issue of resource allocation in multicast networks (esp. in MANETs). In a typical multicast network, a single tree is build with the source node as the root. In such a tree, only a few internal nodes contribute most resources and are involved in performing the multicast functionality. This leads to an un-even utilization of network resources. This problem is more prominent in MANETs where network resources are limited. A possible solution to the problem is to split the multicast content over a number of trees. Multiple trees provide several paths for the multicast content and get more nodes involved in implementing the multicast functionality. However, in this setup, not all the trees get to use the best weight edges, thus the overall multicast latency increases. The dissertation examines MEST, a distributed algorithm to construct multiple edge-sharing trees for small group multicast. MEST balances the resource allocation and delay constraints by choosing to overlap certain edges that have low weights.

7.2 Future Work

In this work, the MEST algorithm was targeted for small group multicast networks. Future, implementations will focus on making modifications to MEST so that it can work with large group multicast too. The current idea proposes to reduce the multicast delay by building low weight spanning trees. Future variants of MEST, we will attempt to build multicast trees where the source node has a low eccentricity. Such trees would have a low multicast delay since the source node would have a shortest path with every node in tree.

Secondly, we are planning to extend our work with location based overlay multicast trees. We are investigating an algorithm that can adaptively divide a cell into smaller sub-cells if the density of nodes increases beyond a certain value. Smaller cell size gives better performance but higher broadcast overhead. We are looking at ways to reduce the service discovery broadcast overhead small size cells. The current SOLONet simulations considered square cells; future versions will be tested with other shapes and the effect on performance.

Finally, we plan to extend our work in the area of indoor location positioning; especially the Bluebot and Bluetooth project. We plan to implement the Bluetooth Location sensing system in our eLANS lab and carry out tests to examine the performance of the system in presence of interference from other RF technologies. We would also like to develop a prototype system (similar to WEBOTS [56]) and measure the positioning accuracy and the feasibility of the system. For the Bluebot project, we are planning to reduce this variation by using a robot that can be controlled to move in a directed pattern. We have considered employing a feedback system so that the direction of the robot is controlled by the RF system. A dual-variable gain antenna system can be used such that the high gain antenna controls the movement of the robot until its low gain partner sees the tag being tracked. In our current implementation, we noticed that signal strength based Wi-Fi systems are easily affected by changes in the surroundings. In the future we plan to make use of systems that are immune to environmental changes (e.g. Time-of-flight, Time-of-arrival and Angle-of-arrival based positioning systems) and analyze the performance of our BlueBot system. We are also looking at various ways to make 3D positioning possible.

8 References

[1] "AeroScout (formally BlueSoft)", http://www.aeroscout.com/

[2] "The Bat Ultrasonic Location System", http://www.uk.research.att.com/bat

[3] "BlueTags", http://www.bluetags.com/

[4] "The Cricket Indoor Location System", http://nms.lcs.mit.edu/projects/cricket/

[5] "Ekahau Positioning System", http://www.ekahau.com/

[6] "HP Cooltown project", http://www.hpl.hp.com/archive/cooltown/

[7] "Intermec UHF PC Reader", http://www.intermec.com/

[8] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/

[9] "The Official Bluetooth Membership Site", http://www.bluetooth.org/

[10] "Radianse Indoor Positioning", http://www.radianse.com/

[11] "Roomba Robotic Floorvac", http://www.roombavac.com/

[12] "Versus Technology", http://www.versustech.com/

[13] "WhereNet location tracking systems", http://www.wherenet.com/

[14] "WiFi (802.11) and Bluetooth - An Examination of Coexistance Approaches," White Paper, Mobilan Corporation 2001.

[15] G. Anastasi, E. Borgia, M. Conti, and E. Gregori, "Wi-Fi in Ad Hoc Mode: A Measurement Study," presented at PerCom 2004, March 2004.

[16] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris, "The case for resilient overlay networks," presented at 8th Annual Workshop on Hot Topics in Operating Systems (HotOS-VIII), May, 2001.

[17] B. Awerbuch, "Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems," presented at STOC, 1987.

[18] B. Awerbuch, "Randomized distributed shortest paths algorithms," presented at Annual ACM Symposium on Theory of Computing, Seattle, Washington, 1989.

[19] P. Bahl and V. N. Padmanabhan, "RADAR: An In-building RF-based User Location and Tracking System," presented at IEEE INFOCOM, Tel-Aviv, Israel, March, 2000.

[20] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," presented at ACM SIGCOMM, August, 2002.

[21] R. Battiti, M. Brunato, and A. Villani, "Statistical Learning Theory for Location Fingerprinting in Wireless LANs," presented at Technical Report DIT-02-0086, 2002.

[22] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, S. Ni, Y. Tseng, Y. Chen, and J. Sheu, "The Broadcast storm problem in a mobile ad hoc network," presented at 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, August 1999.

[23] E. Bommaiah, M. Liu, A. MvAuley, and R. Talpade, "AMRoute: Ad Hoc Multicast Routing Protocol." Internet Draft, draft-manet-amroute-00.txt, March, 2002.

[24] K. M. Chandy and J. Misra, "Distributed computation on graphs: shortest path algorithms," *Communications of the ACM*, vol. 25, no. 11, pp. 833 - 837, Nov 1982.

[25] K. Chen and K. Nahrstedt, "Effective Location - Guided Tree Construction Algorithm for Small Group Multicast in MANET," presented at IEEE INFOCOM, May, 2002.

[26] C. C. Chiang, M. Gerla, and L. Zhang, "Forward Group Multicasting Protocol Of Multihop, Mobile Wireless Networks," *ACM-Baltzer Journal of Cluster Computing: Special Issue on Mobile Computing*, vol. 1, no. 2, pp. 187-96, 1998.

[27] C. F. Chiasserini and R. R. Rao, "Coexistence Mechanism for Interference Mitigation between IEEE 802.11 WLANs and Bluetooth," presented at Infocom, New York, NY, June, 2002.

[28] M. Chiesa, R. Genz, F. Heubler, K. Mingo, C. Noessel, N. Sopieva, D. Slocombe, and J. Tester, "RFID", http://people.interaction-ivrea.it/c.noessel/RFID/RFID_research.pdf (also see /RFID_timeline.pdf)

[29] Y. Chu, S. Rao, and H. Zhang, "A Case of End System Multicast," presented at ACM Sigmetrics, June, 2000.

[30] R. Cohen and G. Kaempfer, "A Unicast-based Approach for Streaming Multicast," presented at IEEE INFOCOM, April, 2001.

[31] C. d. M. Cordeiro, H. Gossain, and D. P. Agrawal, "Multicast over Wireless Mobile Ad Hoc Networks: Present and Future Directions," in *IEEE Networks*, vol. 17, January, 2003.

[32] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, Mass., New York MIT Press, McGraw-Hill, 1990.

[33] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment Issues for the IP Multicast Service and Architecture," in *IEEE Network Magazine*, Jan-Feb, 2000.

[34] R. Dube, C. D. Rais, K. Wang, and S. K. Tripathi, "Signal stability based adaptive routing (SSA) for ad hoc mobile networks," presented at IEEE Personal Communication, February, 1997.

[35] H. Eriksson, "MBone: The Multicast Backbone," *Communications of the ACM*, vol. 37, no. 54-60, August, 1994.

[36] M. Faloutsos and M. Molle, "What features really make distributed algorithms efficient," presented at International Conference on Parallel and Distributed Systems, 1996.

[37] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM TOPLAS*, vol. 5, no. 1, pp. 66-77, January 1983.

[38] J. J. Garcia-Luna-Aceves and E. L. Madruga, "The Core-Assisted Mesh Protocol," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1380-94, August, 1999.

[39] M. S. Gast, *802.11 Wireless Networks, A definitive Guide* O'Reilly Networking, April 2002.

[40] M. Gerla and J. T. Tsai, "Multicluster, mobile multimedia radio network," *ACM-Blatzer Wireless Network*, no. 255-65, 1995.

[41] N. Golmie, N. Chevrollier, and O. Rebala, "Bluetooth and WLAN Coexistence: Challenges and Solutions," in *IEEE Wireless Communications Magazine*, Dec, 2003.

[42] C. Gui and P. Mohapatra, "Efficient Overlay Multicast for Mobile Ad Hoc Networks," presented at Wireless Communications and Networking Conference (WCNC), New Orleans, Louisiana, March, 2003.

[43] A. Haeberlen, E. Flannery, A. Ladd, A. Rudys, D. Wallach, and L. Kavraki, "Practical Robust Localization over Large-Scale 802.11 Wireless Networks," presented at Mobicom 2004, Philadelphia, PA, September, 2004.

[44] J. Hightower and G. Borriello, "A Survey and Taxonomy of Location Sensing Systems for Ubiquitous Computing," University of Washington, Department of Computer Science and Engineering, Seattle, WA, Aug 2001 CSE 01-08-03.

[45] J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian Filtering for Location Estimation," *IEEE Pervasive Computing*, no., July-Sept, 2003.

[46] J. Hightower, R. Want, and G. Borriello, "SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength," presented at UW CSE 00-02-02, University of Washington, Department of Computer Science and Engineering, Seattle, WA, Feburary, 2000.

[47] P. Humblet, "Another adaptive distributed shortest path algorithm," *IEEE/ACM Transactions on Communications*, vol. 39, no. 6, pp. 995-1003, Jun. 1991.

[48] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole, "Overcast: Reliable multicasting with an overlay network," presented at Fourth Symposium on Operating System Design and Implementation (OSDI), San Diego, CA, October 2000.

[49] J. Jetcheva and D. B. Johnson, "Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks," presented at MobiHoc, October, 2001.

[50] L. Ji and M. S. Corson, "Differential Destination Multicast - A MANET Multicast Routing Protocol for Small Groups," presented at IEEE INFOCOM, April, 2001.

[51] D. Johnson and D. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, 1996.

[52] B. Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," presented at ACM/IEEE MOBICOM, August, 2000.

[53] S. Klemmer, S. Waterson, and K. Whitehouse, "Towards a Location-Based Context-Aware Sensor Infrastructure," Dec 2000.

[54] H. Koshima and J. Hoshen, "Personal Locator Services Emerge," in *IEEE Spectrum*, February, 2000.

[55] M. Kwon and S. Fahmy, "Topology-Aware Overlay Networks for Group Communication," presented at ACM NOSSDAV, May, 2002.

[56] A. Lambert, "Thesis: WEBOTS: WEB BASED OBJECT TRACKING SYSTEM," in *Dept of Computer Science and Engineering*: Michigan State University, 2002.

[57] S. Lee, R. Sherwood, and B. Bhattacharjee, "Cooperative Peer Groups in NICE," presented at IEEE INFOCOM 2003, April 2003.

[58] S. J. Lee, M. Gerla, and C. C. Chiang, "On Demand Multicast Routing Protocol," presented at IEEE WCNC, September, 1999.

[59] N. Lynch, *Distributed Algorithms* Morgan Kaufmann Publishers, Inc., 1996.

[60] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," presented at SOSP'03, Lake Bolton, New York, October, 2003.

[61] M. Mauve, H. Füßler, J. Widmer, and T. Lang, "Position-based multicast routing for mobile Ad-hoc networks," presented at Technical Report TR-03-004, Department of Computer Science, University of Mannheim, 2003.

[62] M. F. Mokbel, W. A. Elhaweet, and M. N. Elderini, "An Efficient Algorithm for Shortest Path Multicast Routing Under Delay and Delay Variation constraints," presented at Symposium on Performance Evaluation of Computer and Telecomm. Systems, SPECTS, Vancouver, Canada., 2000.

[63] L. M. Ni, Y. Liu, Y. C. Lau, and A. Patil, "LANDMARC: Indoor Location Sensing Using Active RFID," *Wireless Networks*, vol. 10, no. 4, pp. 701-710, Nov, 2004.

[64] S. Y. Ni, Y. C. Tseng, Y. S. Chen, and J. P. Sheu, "The Broadcast storm problem in a mobile ad hoc network," presented at 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), August, 1999.

[65] D. Niculescu and B. Nath, "VOR Base Stations for Indoor 802.11 Positioning," presented at Mobicom 2004, Philadelphia, PA, September, 2004.

[66] K. Obraczka, G. Tsudik, and K. Viswanath, "Pushing the Limits of Multicast in Ad Hoc Networks," presented at IEEE ICDCS, April, 2001.

[67] P.Antoniadis and C.Courcoubis, "Market Models for P2P Content Distribution," http://citeseer.nj.nec.com/560319.html.

[68] V. D. Park and M. S. Corson, "Temporally-Ordered Routing Algorithm (TORA)," presented at Internet-Draft, draft-ietf-manet-tora-spec-00.txt, Nov 1997.

[69] A. Patil, "Thesis: Performance of Bluetooth technologies and their applications to location sensing," in *Electrical and Computer Engineering*: Michigan State University, Aug 2002.

[70] A. Patil, A.-H. Esfahanian, L. Xiao, and Y. Liu, "Resource Allocation using Multiple Edge-Sharing Multicast Trees", presented at Mobile Ad-hoc and Sensor Systems (IEEE MASS 2005), Washington DC, Nov 2005.

[71] A. Patil, D. Kim, and L. M. Ni, "A Study of Frequency Interference and Indoor Location Sensing with 802.11b and Bluetooth Technologies," presented at Wireless Telecommunications Symposium (WTS 2005), Pomona, CA, April 2005.

[72] A. Patil, Y. Liu, L. M. Ni, L. Xiao, and A.-H. Esfahanian, "POMA: Prioritized Overlay Multicast in Ad Hoc Environments," presented at IEEE International Conference on High Performance Computing (HiPC 2003), Hyderabad, India, Dec 2003.

[73] A. Patil, Y. Liu, L. Xiao, A.-H. Esfahanian, and L. M. Ni, "SOLONet: Sub-Optimal Location-Aided Overlay Network for MANETs," presented at IEEE Mobile Ad hoc and Sensor Systems, Ft Lauderdale, Florida, October, 2004.

[74] A. Patil, J. Munson, D. Wood, and A. Cole, "Bluebot: Asset Tracking Via Robotic Location Crawling," presented at Technical Report RC23510, IBM T. J. Watson Research Center, Aug 2004.

[75] A. Patil, J. Munson, D. Wood, and A. Cole, "Bluebot: Asset Tracking via Robotic Location Crawling," presented at IEEE International Conference on Pervasive Services 2005 (ICPS'05), Santorini, Greece, July 2005.

[76] C. E. Perkins and P. Bhagwat, "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers," presented at SIGCOMM, Aug 1994.

[77] C. E. Perkins, E. M. Royer, and S. R. Das, "Ad Hoc On Demand Distance Vector (AODV) Routing," presented at Internet Draft, draft-ietf-manet-aodv-10.txt, March 2002.

[78] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," presented at MOBICOM 2000, Boston, MA, Aug 2000.

[79] G. Roussos, "Location Sensing Technologies and Application," School of Computer Science and Information Systems, Birkbeck College, University of London, Nov 2002.

[80] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," presented at IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, November, 2001.

[81] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The design of a large-scale event notification infrastructure," presented at NGC2001, UCL, London, November 2001.

[82] E. M. Royer and C. E. Perkings, "Multicast Operation of the Ad-Hoc On-Demand Distance Vector Routing Protocol," presented at ACM MOBICOM, August, 1999.

[83] M. B. Shoemake, "Wi-Fi (IEEE 802.11b) and Bluetooth Coexistence Issues and Solutions for the 2.4 GHz ISM Band," Texas Intruments, White Paper, Feb 2001.

[84] G. Singh and A. J. Bernstein, "A highly asynchronous minimum spanning tree protocol," *Distributed Computing*, vol. 8, no. 3, pp., 1995.

[85] J. Small, A. Smailagic, and D. Siewiorek, "Determining User Location For Context Aware Computing Through the Use of a Wireless LAN Infrastructure," Dec 2000.

[86] P. Steggles and J. Cadman, "A Comparison of RF Tag Location Products for Real-World Applications," Ubisense March 2004.

[87] I. Stoica, T. Ng, and H. Zhang, "Reunite: A recursive unicast approach to multicast," presented at IEEE INFOCOM, March, 2000.

[88] I. Stojmenovic, "Position based routing in ad hoc networks," in *IEEE Commmunications Magazine*, vol. 40, July, 2002,, pp. 128-134.

[89] Y. C. Tseng, S. Y. Ni, and E. Y. Shih, "Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc networks," presented at IEEE 21st International Conference on Distributed Computing Systems, 2001.

[90] B. Wang and S. K. S. Gupta, "On maximizing lifetime of multicast trees in wireless ad hoc networks," presented at International Conference on Parallel Processing, ICPP-03, Kaohsiung, Taiwan, October 2003.

[91] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems*, vol. vol. 10, no. pp. 91--102, Jan. 1992.

[92] G. Welch, G. Bishop, L. Vicci, S. Brumback, K. Keller, and D. Colucci, "The HiBall Tracker: High-Performance Wide-Area Tracking for Virtual and Augmented Environments," presented at ACM Symposium on Virtual Reality Software and Technology (VRST 99), University College London, Dec 1999.

[93] J. Wu, "Dominating-Set-Based Routing in Ad Hoc Wireless Networks," in *Handbook of Wireless Networks and Mobile Computing*, I. Stojmenovic, Ed.: John Wiley & Sons, 2002, pp. 425-450.

[94] L. Xiao, A. Patil, Y. Liu, L. M. Ni, and A.-H. Esfahanian, "Prioritized Overlay Multicast in Ad-hoc Environments," *IEEE Computer Magazine*, no., Feburary, 2004.

[95] Y. Xue and B. Li, "A Location-aided Power-aware Routing Protocol in Mobile Ad Hoc Networks," presented at IEEE GLOBECOM, San Antonio, Texas, November, 2001.

[96] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Y. Wang, "Overlay Mesh Construction Using Interleaved Spanning Trees," presented at INFOCOM 2004, March 2004.

[97] M. Youssef, A. Agrawala, and A. U. Shankar, "WLAN Location Determination via Clustering and Probability Distributions," presented at IEEE PerCom, Fort Worth, Texas, March, 2003.

[98] H. Zhou, L. M. Ni, and M. W. Mutka, "Prophet Address Allocation for Large Scale MANETs," *Ad Hoc Networks Journal*, vol. 1, no. 4, pp. 423-434, November, 2003.

MICHIGAN STATE UNIVERSITY LIBRARIES
3 1293 02736 9531