ROUTING IN LARGE-SCALE AD HOC NETWORKS BASED ON A SELF-ORGANIZING COORDINATE SYSTEM

Shu Du

Thesis: Master of Science
Computer Science
Rice University, HOUSTON, TEXAS (APRIL 2004)

UMI Number: 1419069

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 1419069

Copyright 2004 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

RICE UNIVERSITY

Routing in Large-Scale Ad Hoc Networks Based on a Self-Organizing Coordinate System

by

Shu Du

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

[illegible]

Dr. David B. Johnson, Chair
Associate Professor of Computer Science

[illegible]

Dr. Peter Druschel
Professor of Computer Science

[illegible]

Dr. Rudolf H. Riedi
Assistant Professor of Statistics

HOUSTON, TEXAS

APRIL, 2004

Routing in Large-Scale Ad Hoc Networks Based on a Self-Organizing Coordinate System

Shu Du

Abstract

In this thesis, I present the design and evaluation of new techniques to solve the routing problem in large-scale city-wide wireless ad hoc networks. With an upsurge of wireless technologies, many routing protocols have been proposed for wireless ad hoc networks, but none of them have been proven capable of handling the tasks in a large-scale network that is composed of thousands of nodes. We propose a new routing scheme that is a hybrid of the current proactive and reactive routing mechanisms. This scheme uses proactive beaconing messages to build a virtual hierarchical coordinate system in an ad hoc network and thereafter uses reactive routing maintenance techniques to bypass the stale information between beacons. I implemented this new hybrid scheme in the network simulator *ns-2*. The simulation results show that it performs well in a large-scale ad hoc network of one thousand of nodes with low overhead.

Acknowledgments

I thank my advisor Dr. David B. Johnson for his guidance throughout the process of my thesis work. His advice has been invaluable to get this work done. I also thank my committee members Dr. Peter Druschel and Dr. Rudolf Riedi, not just because of their presence on the committee, but also because of the our close discussions during the past six months in our Safari group meetings.

I am also grateful to our group seniors Amit Saha, Santashil Palchaudhuri for their assists during the thesis work. I would also like to thank Ahamed Khan, Ansley Post and Johnny Ngan for their discussions with me.

Finally, I want to thank my family and my girl friend Qiuyue for their continual encouragement. The support they have given is immeasurable!

Contents

Illustrations

Chapter 1

Introduction

In the past twenty years, an upsurge in networking technologies and the Internet has greatly changed our societies. Within the past ten years, wireless networking technologies and cellular phone systems have taken huge leaps to expand our digital communication significantly. Unfortunately, although we are now in a unique stage in history when information exchanges can be achieved so conveniently, we are actually depending more and more on information exchange infrastructures, such as wired network backbones and cellular base stations. These infrastructure-based networks are vulnerable to natural disasters and malicious attacks. Alleviating our dependence on networking infrastructures has attracted more and more research attention recently, both in academia and in industry.

1.1 Ad Hoc Networking

Ad hoc networking is one of the hot research topics in the last few years because it promises to provide us reliable network services with limited overhead and to eliminate completely the dependence on network infrastructures. An ad hoc network is a group of wireless communication nodes that forward packets for each other so that nodes which are out of direct wireless transmission ranges can still communicate with each other. The big challenge, as well as the research focus, in ad hoc networks is to solve the routing problem. Unlike infrastructure-based networks, a wireless ad hoc network usually has a dynamic network topology. This is not only because of the physical features of the environment-sensitive wireless channels, but also due to the fact that the communication nodes within ad hoc networks are generally able to move around.

In order to resolve the routing problem in ad hoc networks, many routing protocols have been proposed in the past decade. Despite their differences, these routing protocols can be classified into two major categories: proactive routing protocols and reactive routing protocols.

Similar to routing protocols in wired networks, proactive routing protocols, like DSDV [28], use periodically broadcasted maintenance packets in an attempt to make sure that a node always has a route to all the destination nodes in the network. Although proactive routing protocols are usually simpler to implement than reactive routing protocols, they are sensitive to the dynamic topology of network nodes and therefore have difficulties in achieving high performance in the networks composed of nodes with high mobility. On the other hand, reactive routing protocols, such as DSR [16] and AODV [30], do not use periodic broadcasted maintenance packets. For reactive routing protocols, routing information is collected by the communicating nodes only when it is necessary to do so. In other words, only when a node wants to communicate with another node does this node broadcast its routing request packets and collect the path information to the destination. Reactive routing protocols have been shown to perform well in medium- or small-scale ad hoc networks.

Although many routing protocols have been proposed, extensively studied and fine-tuned in the last decade, none of the current routing protocols have shown convincing capabilities of handling the routing task in a large-scale ad hoc network. This is basically due to the unscalable network bandwidth of ad hoc networks [21], as well as due to the lack of scalability in the protocol designs. The scalability problem in ad hoc networks is difficult to solve for the current routing protocols in the sense of both bandwidth overhead and storage requirement.

Given the problems of the current routing protocols and the increasing demand to have a reliable networking system in a large-scale ad hoc network, our research group recently proposed a series of solutions in ad hoc networks to provide dependable connectivities as well as traditional Internet services in a city-wide large-scale ad hoc network. The project is called *Safari*, a scalable architecture for ad hoc networking and services[12, 19, 33]. Safari provides a self-organizing network hierarchy, scalable ad hoc network routing, self-organizing network services and an integrated ad hoc network architecture. Safari is composed of a series of protocols to achieve the above goals, but the basis of all these protocols is a self-organizing proximity-related hierarchical address assignment protocol. Generally speaking, we use proactive, periodic, distance-limited broadcasted packets to help the nodes in an ad hoc network compose clusters with the nodes that are topologically nearby. After the fundamental clusters are generated, the same clustering algorithm is applied to the lead node of each fundamental cell to generate a higher level cell. Continual applications of the same algorithm to the network will generate further higher level cells. Finally, a cell which covers the whole network will be generated. By this proactive address assignment protocol, each node in the ad hoc network will get a unique hierarchical address. More importantly, topologically proximate nodes will be assigned similar addresses. This hierarchical address assignment protocol is called the *buoy* protocol.

Based upon the hierarchical addresses assigned, routing in Safari networks becomes similar to the routing schemes in wired networks. The routing scheme we designed to support Safari is called *SHARP* (Scalable Hierarchical Ad hoc Routing Protocol). Generally speaking, the source node looks at the hierarchical address of the destination node, and then delivers the packet *towards* the lowest common cell in their hierarchical tree. As the packet travels through the network, intermediate forwarding nodes will share a longer and longer common prefix in their hierarchical addresses with the destination node, until finally, the packet reaches a node in the same fundamental cell where the destination node is located. From there, the packet is delivered through DSR, an existing medium-scale efficient reactive routing mechanism, to the destination node.

If we consider the hierarchical address assignment protocol and the above routing mechanism as a integrated new routing mechanism, this new mechanism acts like a hybrid of proactive routing and reactive routing. SHARP uses a proactive periodic routing scheme to set up a virtual coordinate system in the ad hoc network and to provide the high-level, coarse-grained topology information. At the same time, SHARP uses a reactive routing mechanism to handle high mobility and to provide the low-level, fine-grained topology information. In this way, SHARP takes advantages of both proactive and reactive techniques and avoids the disadvantages of both.

1.2 Thesis Contributions

I made the following contributions in this thesis: First, I developed Safari's buoy protocol from its rough design guidelines and proposed several techniques to improve its structural stability. Second, I proposed a reactive local route repair strategy which can significantly improve SHARP's routing performance. Finally, I evaluated SHARP's performance based on my simulation implementation.

1.3 Thesis Organization

This thesis presents the design details of the Safari buoy and SHARP protocols, as well as the simulation results for its routing performance evaluation. In Chapter 2, I introduce the background for this thesis, focusing on the previous work that has been done in this field. In Chapter 3, a general introduction of the Safari architecture is given. In Chapter 4, I describe the design details of the Safari hierarchical address assignment protocol as the background material for Chapter 5, which presents the Safari routing mechanism in detail. In Chapter 6, the evaluation of Safari routing is discussed. Finally, I draw the conclusions and describe my future work in Chapter 7.

Chapter 2

Related Work

Routing protocols in ad hoc networks can be generally divided into two classes: proactive routing protocols and reactive routing protocols. Proactive routing protocols, also called periodic routing protocols, are named because each node tries to maintain routes to all destinations at all times [26, 28, 29]. DSDV [28], a typical example of proactive routing protocols, requires every node to broadcast periodically its own routing table to its direct neighbors. At the same time every node listens to its neighbors' broadcasting and updates its own routing table. One major problem of proactive routing protocols is that they suffer from high overhead of the periodic broadcasting packets. More importantly, for large-scale or highly mobile networks, proactive routing protocols have a slow rate to converge and therefore have not performed well. On the other hand, reactive routing protocols, also called on-demand protocols, do not try to get or maintain routing information until it is necessary [16, 30]. For example, in DSR [16], the first well-known reactive routing protocol, routing information is collected only when a node has a data packet to send. In DSR, broken links are detected and repaired only when a packet needs to be forwarded. Because of this flexibility of reactive routing protocols, they are more adaptive than proactive ones in ad hoc networks with high mobility. However, as the scale of the network grows and the mobility of nodes increases, the performance of reactive routing protocols also drops quickly as their maintenance overhead increases rapidly.

In order to solve the scalability problem in ad hoc networks, many routing protocols have been recently proposed by different research groups. Geographic routing schemes, use the geographic information of the nodes provided by GPS-like devices to guide routing in ad hoc networks[4, 18, 20, 22, 23]. They usually require the node to have the geographic coordinates of the destination and of the node's direct neighbors. Based on the above information, greedy rules or other heuristic rules can be applied to guide the node to forward the packet towards the direction of the destination. Compared with the traditional proactive and reactive routing protocols, geographic routing protocols are much simpler, but the requirement of GPS-like devices for every node and the mandatory service of the geographic information distribution to the whole network greatly limit the scope of their usage.

Recently, new techniques of gathering geographic information without GPS devices have been proposed to empower the geographic routing protocols [31]. The key idea of these techniques is to derive virtual coordinate information based on connectivity information in networks. Therefore, like proactive routing mechanisms, they need extensive exchange of local connectivity information to converge the view of the global topology. Because of this, their ability to solve the problem of mobility in ad hoc networks is still under investigation.

Another popular technique introduced into ad hoc networks is cluster routing[1, 2, 3, 8, 13, 24, 25, 36, 39]. Protocols of cluster routing use periodic broadcasting around the nodes' neighborhood to divide the network into clusters. Some protocols use the concept of the nested clusters to build a cluster hierarchy in ad hoc networks. Similar to proactive routing schemes, these cluster routing protocols suffer from the fast growing overhead of the periodic broadcasting packets as the network size increases. Some of these cluster protocols depend on fixed cluster heads to solve the mobility problem, but these special requirements greatly limit the scope of their deployment[5, 6, 38].

Landmark routing is another proposed routing technique in ad hoc networks to achieve high scalability. Borrowed from the landmark routing protocol in wired networks, landmark routing protocols in ad hoc networks, such as LANMAR [27] and L+ [9], try to forward packets through some key way points. Since they try to keep the way points equally apart and also allow the use of hierarchical landmarks, these routing protocols can greatly reduce the routing overhead. LANMAR specifically assumes their ad hoc networks consist of groups of nodes related in functionality and mobility. L+, on the other hand, uses a purely proactive approach for routing and is in fact based on DSDV [28]. SHARP, on the other hand, is a hybrid of proactive and reactive routing.

Chapter 3

Safari Introduction

In order to eliminate our dependence on wired and wireless infrastructures and to provide connectivity and traditional Internet services, our research group has recently proposed a novel architecture, called Safari, in ad hoc networks. The Safari architecture can not only support scalable routing in city-wide ad hoc networks that may be composed of tens of thousands of nodes, but also it can make the everyday Internet services, such as the DNS, email and web services, available to the participant nodes. Realizing that ad hoc networks and infrastructure networks can be integrated to capture the advantages of both, the Safari group also plans to build an architecture in integrated networks so that infrastructure-based Internet services can be accessed by the ad hoc network users.

3.1 Overview

Safari is composed of the following five inter-dependent modules:

- **Self-organizing network hierarchy:**

  The basis of the Safari architecture is a nested cellular architecture that is self-organized by the nodes in the network. Based on a buoy protocol that allows the nodes to periodically send out hop-limited beaconing packets, the nested cellular architecture can be generated according to the proximity in the topology of the wireless transmission connectivity. With this hierarchical architecture, scalable routing in ad hoc networks can be implemented efficiently and be capable of handling high mobility. Additionally, because this hierarchical architecture is formed based on proximity-based criteria, it can naturally support the proximity-based overlay networks that are usually used to provide efficient and reliable Internet services.

- **Distributed address storage and updating:**

  After the hierarchical architecture is generated for the network, every node will have a hierarchical address, which can locate itself in the nested cellular structure. The hierarchical address of a node needs to be known by other nodes in the network when they want to communicate with that node. Therefore, hierarchical addresses should be stored and updated in the network. They should also be available for retrieval by the other nodes in the network in a reliable and efficient way. Safari uses a distributed hash table (DHT) to provide this service.

- **Scalable ad hoc network routing:**

  SHARP, the routing protocol we have designed to support Safari hierarchical architecture, can be divided into two phases. The goal of the first phase, called inter-cell routing, is to use the hierarchical address of a destination node to deliver the packet to any node in the fundamental cell where the destination node is located. Routing in this phrase is based on the destination's hierarchical address and the records of beacons stored in the routing tables of the intermediate forwarding nodes. The task of the second phase, called intra-cell routing, is to deliver the packet to the specific destination node within that fundamental cell. We use a modified version of the DSR routing protocol [16] for intra-cell routing. Because the size of the inter-cell routing table in each node grows logarithmically with the size of the network, the scalability requirement is achieved. In addition, because the beacon records in the inter-cell routing table are passively received from the buoy protocol, the overhead of the routing protocol itself is limited.

- **Self-organizing network services:**

  One important function of the Safari architecture is to provide the basic Internet services to ad hoc network users. Because ad hoc networks are without infrastructure, traditional client-server service models in the Internet are hard to deploy in ad hoc networks. Recently, peer-to-peer (P2P) technologies have brought a new server-less model into the Internet to provide the traditional Internet services, such as hostname, storage, and instant messages [10, 32, 34, 35, 37, 40]. Because Safari can provide a self-organizing hierarchical structure, it is natural to deploy the P2P service model into Safari. In addition, because Safari's cellular structure is proximity-based, the traditional location-aware Internet services can benefit from it.

- **Integrated ad hoc network architecture:**

  In the situation where wired or wireless infrastructure networks and ad hoc networks are integrally deployed, ad hoc networks can extend digital communication services to areas where the infrastructure networks cannot reach. At the same time, infrastructure networks can support high bandwidth and highly reliable network services that ad hoc networks find difficult to achieve. Moreover, in situations such as disaster recovery or military operations, only isolated infrastructure networks may be available in the area. Therefore, it is beneficial to use ad hoc networks to bridge these infrastructure networks into one unified digital communication platform. Due to the ease in merging and partitioning in Safari's hierarchical structures, the Safari architecture can efficiently support for such integrated networks.

3.2 Hierarchical Address and Nested Cellular Structure

In a Safari network, a distributed self-organizing buoy protocol assigns every node in the network a hierarchical address, with which a nested cellular structure is generated. The unique hierarchical address assigned to each node can locate that node's position in the nested cellular structure. Figure 3.1 illustrates an example cellular structure formed by Safari.

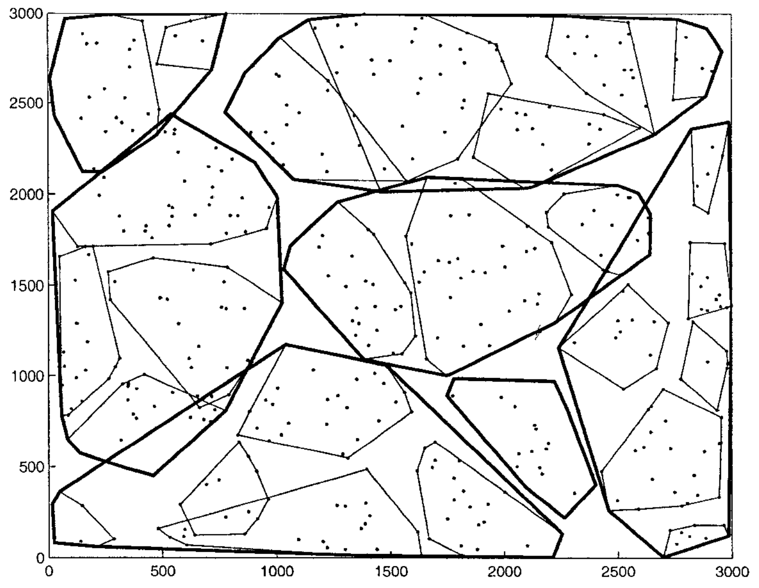

Figure 3.1: Safari hierarchy illustration: a visualization of the hierarchy generated in a network of 500 nodes in a 3000 m $\times$ 3000 m area. The small, thin-edged polygons represent the fundamental (level 1) cells, and the big, thick-edged polygons represent the higher level (level 2) cells. The X and Y axis are the geographic coordinates of the area.

I will present the six definitions of the Safari structure as follows:

1. **Hierarchical level:**

   In a Safari network, a non-negative integer number H_{level} is assigned to every single node to stand for its current hierarchical level. A *buoy* node (also called an *anchor* node) is defined as a node with $H_{level} \geq 1$. A *normal* node is defined as a node with $H_{level} = 0$. The *king* node is the node which has the maximum H_{level} in the network. If a node has H_{level} of n, it is also considered to be at H_{level} of $0, \ldots, n-1$. H_{level} is also simplified as *level* in the rest of the thesis.

2. **Stable state:**

   When a network gets started, every node is a *normal* node. Then a self-organizing distributed algorithm is deployed to organize the system into a dynamically stabilized state and to keep the system in that state thereafter, as for example, node move or wireless propagation conditions change. In an absolute stable state, there should be only one *king* node and its H_{level} should be kept the same. If there are no topology changes in the network, there should be no hierarchical address changes in the stable state.

3. **Direct association:**

   For a node A with H_{level} equal to n, it associates with one single node B with $H_{level} = n+1$ at any given time and considers B as its *parent*. Node A is then called a *child* of B. In this thesis, the association relationship between A and B is denoted as

   $$A : n \longrightarrow B : n+1 \ .$$

   Moreover, we define that $A : n \longrightarrow A : n+1$ if A's $H_{level} \geq n+1$, which means that a buoy always associates with itself since we previously defined that a level $n+1$ node is also a level n node. A level n node can only have at most one level $n+1$ parent at any given time. In a stable state, every node has its parent except for the king node at the highest level.

4. **Indirect association:**

   If $A : n \longrightarrow B : n+1$ and $B : n+1 \longrightarrow C : n+2$, we say node A is a *descendant* of node C, and node C is called an *ancestor* of node A. In this thesis, this indirect association relationship between A and C is denoted as

   $$A : n \Longrightarrow C : n+2 \ .$$

   Also, we define that if $A : n_1 \longrightarrow B : n_2$, $A : n_1 \Longrightarrow B : n_2$, which means that a direct association is always an indirect association. If $A : n_1 \Longrightarrow B : n_2$ and $B : n_2 \Longrightarrow C : n_3$, we have $A : n_1 \Longrightarrow C : n_3$, which means the indirect association relationship is transitive.

   From the above definitions, we also get that $A : n \Longrightarrow A : n+1$ since $A : n \longrightarrow A : n+1$, if A is a level $n+1$ buoy.

5. **Cell:**

   We can define cells in the Safari architecture with the indirect association relationship. A cell is identified by the head of the cell, which is a *buoy* node. In other words, for a *buoy* node A, node A also defines a cell with itself as the cell head. A level n buoy X defines the cell $C_X : n$ as the set of the nodes

   $$\{ \ x \ | \ x : n_x \Longrightarrow X : n, n_x \geq 0 \text{ and } n_x < n \} \ .$$

   Because buoy nodes are associated with each other, cells in Safari are naturally nested within each other. We can derive from the previous definitions that

   - If $A : n_a \Longrightarrow B : n_b$, then $C_A : n_a \subseteq C_B : n_b$
   - In the stable state, the whole network is a cell, and the head of that cell is the *king* node.

We define a *fundamental cell* as a cell with a head node of level 1.

6. **Hierarchical address:**

   A hierarchical address assigned to a Safari node can be defined as a series of the *ID* names, which identifies all the current buoys with which this node is indirectly associated. A node A with the hierarchical address of $A_0A_1A_2 \ldots A_{n-1}A_n$ implys that $A_0 = A$, $A_0 : 0 \longrightarrow A_1 : 1$, $A_1 : 1 \longrightarrow A_2 : 2$, ..., $A_{n-1} : n-1 \longrightarrow A_n : n$. Because a buoy node also defines a cell in Safari architectures, the hierarchical address actually gives the node's position in the Safari structure of the nested cells.

   The *ID* of a node is a number that can distinguish itself from its siblings that have the same parent. Therefore, the *ID* can be the globally unique identifier of the node, such as an IP address, a MAC address, or a product sequence number, but it will be more efficient if it is just a dynamically randomly selected number, which could be much shorter in bits than the node's global identifier. However, in this thesis, I use the *ID* as a global identifier for convenience.

   We have defined that a level $n+1$ buoy is also a buoy with level n, and iteratively, it is also a buoy of level $n-1$, $n-2$, ..., 1. Therefore, we can derive the node's level from its hierarchical address: If a node is of level k, then in its hierarchical address, $A_0 = A_1 = \cdots = A_{k-1} = A_k$.

Chapter 4

Buoy Protocol

After the Safari hierarchical structure is defined, we now need an algorithm to generate the structure in an ad hoc network. Because ad hoc networks are without centralized administration, it is necessary to have a self-organizing, distributed algorithm to finish the task of generating structure. The *buoy* protocol in Safari satisfies the above requirement. Generally speaking, the *buoy* protocol requires every buoy node to periodically broadcast the information of its own existence to the encompassing neighborhood within limited distance. Also, every node in the network keeps a beacon cache to record all the recent beacon packets the node has heard, so that it can choose the nearest buoy node as its parent. The goal of the buoy protocol is not just an algorithm to get "a" structure; instead, it tries to get to an ideal equilibrium state, which means:

- All the buoys of the same level are evenly distributed with an equal distance among them; in other words, the generated cells are of equal size and parents have the same number of children.
- The number of the buoys in the work is as small as possible.
- The hierarchical level of the system is as low as possible.

In summary, the buoy protocol is designed in general to approximate a *perfect* Safari structure.

4.1 Buoy's Proactive Behavior: Sending Beacon Packets

Every buoy node with a level $n \geq 1$ sends out a beacon packet every T_n seconds. The beacon packets are forwarded to all the nodes in its neighborhood within D_n hops. The series of T_n and D_n are given by geometric progression:

$$D_n = \alpha \times D_{n-1} = \alpha^{n-1} \times D_1 \ ,$$

and

$$T_n = \beta \times T_{n-1} = \beta^{n-1} \times T_1 \ ,$$

in which α and β are predefined system parameters. Likewise, D_1 and T_1, the broadcast range and cycle of the fundamental cell, should also be specified in advance according to the requirements of different systems.

Since D_n gives the beacons broadcasting range, it actually determines the maximum distance between a buoy and its children. Therefore, we can derive the maximum radius of a level n cell as

$$\begin{aligned} MAX(R_n) &= D_n + D_{n-1} + \cdots + D_1 \\ &= D_1 \times (\alpha^{n-1} + \alpha^{n-2} + \cdots + 1) \\ &= D_1 \times \frac{h^n - 1}{h - 1} \ . \end{aligned}$$

The size of the cells grows as an exponential function of the level; that is, as the scale of networks grows, the level of the Safari hierarchy will only grow at a logarithmic rate.

The beacon packets broadcasted by buoy nodes include the following fields:

- ***HID*, the hierarchical address of the buoy:**

  The hierarchical address is used to identify the unique hierarchical position of the beaconing source in the Safari nested cellular structure.

- ***N*, the sequence number of the beacon packet:**

  Every buoy node keeps an incremental counter and marks every beacon packet that it originates with a unique sequence number so that a receiving node can identify a duplicate packet. The sequence number in a beacon packet also allows nodes receiving beacons to know which beacon is the freshest one, since complicated wireless channel conditions and jitter timers may cause sequential packets to arrive at a receiving node out of order. The jitter timer is used in wireless communication to reduce packet loss due to collisions. In wireless broadcasting, several nodes may receive the packet simultaneously, and if they broadcast it at the same time, their transmissions will collide. Therefore, instead of sending out the packet immediately after receiving it, a node can add a random delay to the packet processing. This technique is called *jitter*.

- ***L*, the beacon level:**

  A buoy of level $n+1$ is also of a buoy of level $n, n-1, \ldots, 1$, and different levels require different frequency for sending beacon packets. Therefore, the beacons coming from the same buoy may not be of the same level, which means that the beacons may have different transmission ranges even if they are from the same buoy. With the timer controlling the beacon origination, a beacon packet can be marked with the correct level.

- ***D*, the accumulated hop count of the beacon:**

  Receiving nodes need to know the length of the path to determine how far away the originating buoy is in order to choose the nearest higher level buoy as its parent. This hop count number is incremented at every hop during packet forwarding.

- ***R*, the maximum hop count of the beacon:**

  As stated earlier, this value is equal to D_L, where L is the beacon level of the packet.

Every intermediate forwarding node compares the packet's accumulated hop count D with this maximum hop count R. If $R \geq D$, the beacon packet will not be discarded and forwarded further.

4.2 Algorithm to Handle a Beacon Packet

When a node receives a beacon packet, it will do the following things in order:

1. Check to see if the received packet is a duplicate or stale one by its HID and N. If it is, the node will discard the packet. The packet is stale if the node has already received a beacon packet with a higher N from the same buoy.

2. Store the information of the beacon packet into its *beacon cache*, which records the beacon's hierarchical address HID, the beacon's sequence number N, the beacon level L, and the hop count from the beacon's initiator D. The node also calculates the expiration time for the entry of the beacon cache. The higher the beacon level of the packet, the longer its expiration timer will be set, proportional to the beacon initiation timer T_n.

3. Increment L, the accumulated hop count of the beacon packet, and check if $L < R$. If not, the node will discard the beacon packet. Otherwise, the node will rebroadcast the beacon packet.

4.3 Algorithm to Decide HID

After the received beacon packet has been stored into the node's beacon cache, the node can go through its beacon cache to check the recent history of the nearby buoys in order to decide with which buoy it should associate. If there are no appropriate buoys to be selected as a parent, the node will "step up" to become a higher level buoy. Moreover, if a node finds out another buoy is too near to itself, it will decide whether or not it should "step down" to avoid the overhead of having too many buoys.

In the following, I present the algorithm in three inter-dependent parts: the algorithm to decide to step up, the algorithm to decide to step down, and the algorithm to decide the parent.

4.3.1 Stepping up algorithm

In the previous section, I have introduced the maximum hop count broadcast limit D_n of the buoys in level n. A level n buoy uses D_n to decide the maximum hop count it wants to keep away from its direct children, which are level $n-1$ buoys.* Likewise, a level $n-1$ buoy does not want to have a level n buoy more than D_n hops away from itself. Therefore, if a level n buoy cannot hear any higher level buoys within D_{n+1} hops, it will consider stepping up to become a level $n+1$ buoy. However, if all the beacons the node has heard are from lower level buoys, it can identify itself as a *king* buoy; therefore, it will not step up although it can not hear any higher level beacons.

This stepping up checking algorithm will be applied periodically. The checking frequency is in accordance with the origination rate of the level $n+1$ beacons. More specifically, the checking cycle is set as

$$T_{stepUpCheck} = a \times T_{n+1} \quad ,$$

in which T_{n+1} is the originating cycle of level $n+1$ beacons and a is a positive integer. Preferably, a should be bigger than 1 because in unreliable wireless broadcasting transmission, some packets may get lost due to congestion or poor channel conditions.

*In Chapter 5, we will see that in order to support scalable routing, a buoy's actual broadcast range will be expanded beyond D_n.

In addition, in order to avoid several nearby nodes stepping up at the same time, a small random variance σ is added when resetting the $T_{stepUpCheck}$.

Summary: When $T_{stepUpCheck}$ of a level n non-king node expires, if the node finds that no entry in its beacon cache with a higher level $(N > n)$ is within the required limit $(D \leq D_{n+1})$, the node will step up one level. If a node steps up, it will then emit a new beacon packet immediately to notify the neighborhood about this change.

4.3.2 Stepping down algorithm

Because we want buoys to be evenly distributed and as few as possible, we try to avoid a crowd of buoys. Two buoys could be too near to each other either by node mobility or by occasionally dropped packets, which may lead to a wrong decision of stepping up. Therefore, we need to have a mechanism to help buoys decide when to step down in order to keep the dynamic equilibrium.

First we need to define the criteria of *crowd*. A stepping-down distance limit G_n is introduced to judge when two buoys are too close to each other. If a level n buoy hears another buoy with the same or a higher level within the distance G_n, it either thinks that they are too near to each other and will step down, or it assumes the other node will step down. The value of G_n is defined as

$$G_n = \lceil h \times D_n \rceil \quad ,$$

in which D_n is the maximum hop count of the level n buoy, and h is called the *hysteresis coefficient*. The value of h defines the generousness limit of sustaining nearby buoys. If h is smaller, the node is more generous with nearby buoys, and the maximum permitted distance between two buoys becomes smaller.

Theoretically, h could be set as any non-negative real number, but it is only meaningful in practice if $h \in \left[\frac{1}{D_n}, 2\right)$, which sets G_n within $[1, 2 \times D_n)$. G_n should be equal to or greater than one, because the distance between two buoys could never be smaller than one hop. G_n should be smaller than $2 \times D_n$, because when two buoys are at a distance of $2 \times D_n$ hops, the coverage ranges of their transmission circles are cotangent. In that case there is no reason to assume that the buoys are too crowded. We will discuss a more precise range of h later in Section 4.4.

In contrast to the algorithm of stepping up, the algorithm of stepping down does not need to be applied periodically. Instead, only when a node receives a beacon from the same level or a higher level buoy does this stepping down checking function need to be applied.

Summary: If a level n node A receives a beacon from node B which is d hops away and $d < G_n$, then node A steps down a level if

- $Level_A < Level_B$, or
- $Level_A = Level_B$ and $ID_A < ID_B$.

If a node steps down and if its level is still greater than one, it will emit a new beacon packet immediately to notify the neighborhood about this change.

4.3.3 Choose a parent

Most of the time, a node does not need to step up or step down, as it can find an appropriate higher level buoy with which to associate. A node can check all the nearby higher level buoys in its beacon cache to pick the nearest one to be its parent.

The status of a beacon cache gets changed when a new entry is added or when a stale entry is deleted. In either of these two cases, the node needs to check if the change affects its decision on the best appropriate parent. In other words, this algorithm of choosing a parent needs to be applied whenever

- a new beacon packet has just arrived, or
- an entry has just expired and was removed from the beacon cache.

Summary: A level n node goes through all the $n+1$ level beacons it heard recently and then picks out the ones with a distance less than its stepping up limit D_{n+1} and greater than its step down limit $G_n = \lceil h \times D_n \rceil$. Among these qualified potential parents, it chooses the one nearest to itself as its direct associated parent and changes its own hierarchical address accordingly. If a node changes its parent and if its level is still greater than one, it emits a new beacon packet immediately to notify the neighborhood about this change.

4.4 Techniques to Improve Stability

4.4.1 Hysteresis coefficient h requirements

The buoy protocol requires a node of level n to have its parent within the limit of D_{n+1}, and at the same time, the node should be no closer than $h \times D_n$ to any node of the same or higher level. In this way, all the buoys can be evenly distributed in the network and remain in a dynamic equilibrium state even if the wireless transmission is unreliable and nodes are moving.

It is necessary to have the conditions of stepping up and stepping down to be coherent with each other. For example, we do not want a node to step up because it thinks it is too far away from the higher level buoys, and, after stepping up, it immediately has to step down because it finds itself too near to the other buoys of the same level.

In order to satisfy the above requirement, the hysteresis coefficient needs to have a more precise range than the one given in the previous subsection. In the last subsection, we showed that h should be in the range of $\left[\frac{1}{D_n}, 2\right)$. Now we consider the following two cases to further limit its range:

1. **A node A of level n steps up:**

   If node A steps up, it means there is no level $n+1$ node within D_{n+1} hops. Therefore, if d stands for the minimum distance between A and any level $n+1$ node, we must have $d > D_{n+1}$. When A steps up and becomes a level $n+1$ node, we then need to make sure that all the level $n+1$ nodes are beyond G_{n+1} hops away to prevent A from stepping down. Hence, we need to have $d > G_{n+1} = h \times D_{n+1}$. Consequently, in order to satisfy the above conditions, we get

   $$h \times D_{n+1} < D_{n+1} < d \ ,$$

   that is,

   $$h < 1 \ .$$

2. **A node A of level n steps down:**

   If node A steps down, it means the nearest buoy B of the same or a higher level is within G_n hops. Therefore, if d is the distance between A and B, we must have $d < G_n = h \times D_n$. When A steps down and becomes a level $n-1$ node, we then need to make sure that there is at least one level n node within D_n hops to prevent A from stepping up. Hence, we need to have $d < D_n$. Consequently, in order to satisfy the above conditions, we have

   $$d < h \times D_n < D_n \ ,$$

   that is

   $$h < 1 \ .$$

In summary, in order to prevent oscillation of the buoy's stepping up and stepping down, it is necessary to have $h < 1$.

In the original Safari design, the meaning of h slightly differs from the definition here. In the original Safari's proposal [33], the stepping down limit G_n is defined as $G_n = h \times U_n$, in which U_n is the stepping up limit for the level n buoys and equals D_{n+1} in our case. With the original definition, we can derive h's requirement as

$$h < D_n/D_{n+1} = \alpha^{-1} \quad ,$$

if we have the condition that $D_{n+1} = \alpha \times D_n$.

4.4.2 Decreasing the number of beacons

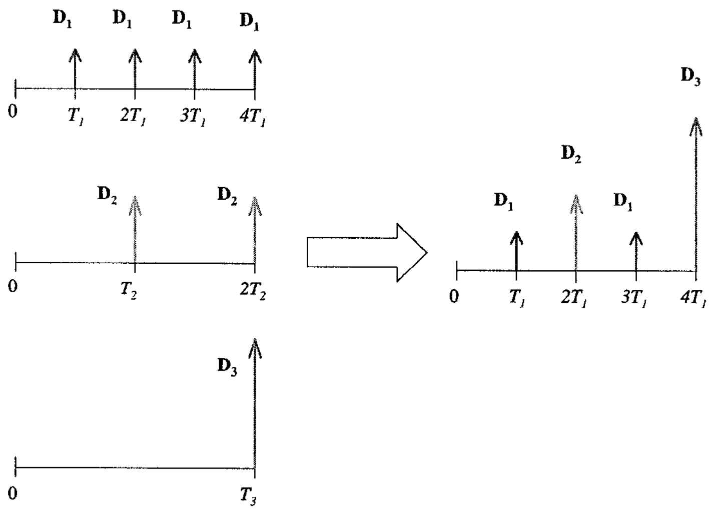

Figure 4.1: Beacon initiating sequences of a three level buoy. Suppose $\beta = 2$.

As mentioned earlier, a level n buoy A needs to broadcast its beacon every T_n seconds, where $T_n = \beta \times T_{n-1}$ for $n > 1$. We also know that if A is of level n, it is also a level $n-1, n-2, \ldots, 1$ buoy. As illustrated in the left column of the Figure 4.1, suppose we start at time 0. At time T_1, a level 1 beacon needs to be initiated by A with the hop limit of D_1. At time $2T_1$, another level 1 beacon is sent out. At time βT_1, a level 1 beacon needs to be sent out as well as the first level 2 beacon. At time $\beta^{n-1}T_1$, beacons need to be sent out for all the levels of $1, 2, \ldots, n-1$, respectively. Furthermore, the first level n beacon should be sent at the same time. Because high level beacons are always required to be transmitted with a larger range, we can suppress the initiation of the lower level beacons when there is a higher level beacon initiated at the same time. As illustrated in the right column of Figure 4.1, at time βT_1, only one beacon of level 2 with the range of D_2 needs to be originated. At time $\beta^{n-1}T_1$, only one beacon of level n with the range of D_n needs to be originated. When a node receives a beacon of level n, it also treats this as if the beacons of the level $2, 3, \ldots, n-1$ are also received from the same buoy. Beacon packets can be marked with the correct level with the same timer which controls the beacon origination. For every T_1 seconds, only one beacon is originated, although the transmission ranges varies according to the level of the beacon.

As a result of this optimization, $\frac{1}{\beta}$ of the level 1 beacons are omitted, and it is the same for every level of the beacons. Therefore, in summary, this strategy can reduce the beacon overhead by $\frac{1}{\beta}$ of all packets.

4.4.3 Get the shortest hop count path

Because simultaneous wireless transmissions may collide with each other, a jitter delay needs to be added on each individual node when a node in a network is forwarding a beacon packet, Although the jitter delay can increase the reliability of the beacon packet distribution, it does introduce a distance estimation error. A node may receive a beacon packet first through a longer path because jitter delays chosen by the intermediate nodes are smaller. When the same beacon packet coming through a shorter path arrives, the node will discard the packet since it is considered as duplicated. Also, if upstream nodes discard beacon packets which have better path information, a further downstream node may never know the shorter path and may make a wrong decision on choosing its associated parent, which is supposed to be the nearest higher level buoy.

To solve this problem, I devised a new rebroadcasting mechanism to let packets with shorter hop count information get distributed. A distance ratio threshold γ is introduced to help make the rebroadcast decision when a node receives a duplicated beacon packet. If $\frac{d_{new}}{d_{old}} \leq \gamma$, then the node rebroadcasts the new beacon packet; otherwise the node discards the new packet. The values d_{new} and d_{old} are hop counts of the paths of the new beacon packet and the old beacon packet, respectively. The value γ is a real number within $[0, 1]$. When γ equals 0, all the duplicate packets will be discarded. The node cannot get better path information, but the overhead of rebroadcasting is low. When γ equals 1, as long as the newly received duplicate packet has a shorter path, it will be broadcasted again. The node can get the best path information, but the overhead of rebroadcasting is higher.

With an appropriately chosen γ, overhead can be limited to a reasonable magnitude. Additionally, the stability of the Safari structure can be greatly improved, because nodes are getting much more accurate distance information.

We use jitter to mitigate packet loss in broadcasting, and by using the above technique, beacon packets usually carry optimized topology information. Nevertheless, sometimes beacon packets can still get dropped due to congestion or to poor channel conditions. Therefore, we need some further mechanisms to bypass the fluctuation of the collected topology information. The easiest way to do this is to add a low frequency filter to the collected information. In my simulation implementation, I used an integer f, called *history factor*, to specify how many continuous packets from the same buoy are needed to be heard before the node decides to choose the buoy as its new parent or decides to step down if the buoy is too close. History factor f alleviates the impact on the structure from fluctuations in the wireless transmission.

Shorter paths recorded in beacon caches can not only increase stability of the Safari architecture, but can also improve routing quality, as it provides shorter, low latency paths. I will discuss the details of routing quality in the following chapters.

Chapter 5

SHARP: Scalable Hierarchical Ad Hoc Routing Protocol

The SHARP protocol can be divided into two phases. Given the hierarchical address of a destination node, the first phase, called *inter-cell routing*, is to deliver the packet to a node in the fundamental cell in which the destination node is located. Once the packet reaches the fundamental cell of the destination, the second phase, called *intra-cell routing*, delivers the packet to the destination node within that fundamental cell.

Inter-cell routing is based on the destination's hierarchical address and on the beacon records stored in the intermediate forwarding nodes. We assume that all wireless links in the network operate bidirectionally. With that assumption, inter-cell routing can be supported by the reverse paths of the beacons that were previously originated by the buoys with which the destination node is associated. For intra-cell routing, any state of the art on-demand routing protocol can be used because the size of a fundamental cell is limited. We choose DSR [15, 16] as the intra-cell routing protocol for its demonstrated stability with high performance in small scale ad hoc networks [7].

When a source node S with HID $[S_nS_{n-1}\cdots S_1]$ has a packet to send to node D, S retrieves the HID $[D_nD_{n-1}\cdots D_1]$ of D using a distributed hash table (DHT) address lookup service*. We claim that there must exist an integer $k, 1 \leq k \leq n$, so that $S_i = D_i$ for all $k \leq i \leq n$, where n is the number of levels in the hierarchy. If $k = 1$, then S and D belong to the same fundamental cell and intra-cell routing is invoked. Otherwise, inter-cell routing

*The Safari architecture uses the DHT to store and look up a node's hierarchical address. Detailed DHT operations are beyond the scope of this thesis.

is invoked. If inter-cell routing is applied, the source S adds the HID of D to the packet header before sending the packet, so that the forwarding intermediate nodes can get the HID of the destination from the packet instead of having to do a separate address lookup. When a node has a packet to forward, it uses the same logic to decide whether to continue inter-cell routing or to apply intra-cell routing.

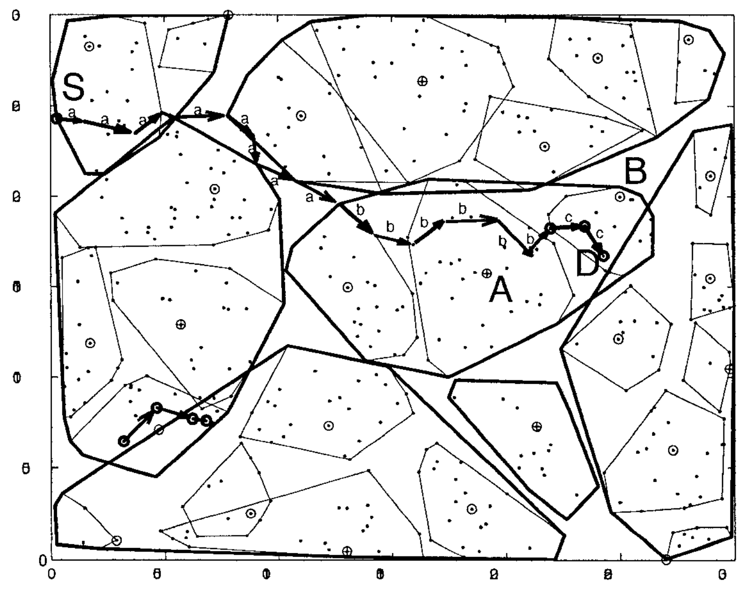

Figure 5.1: SHARP routing overview

Consider the example shown in Figure 5.1, in which a packet is sent from the source node S to the destination node D in SHARP. First, S uses inter-cell routing, along the route labeled with a and b, to reach the fundamental cell of the destination node D. Within the destination fundamental cell, intra-cell routing delivers the packet to the destination, along the route labeled with c. The route marked with a is the reverse path of a beacon originated by the level 2 buoy A, with which D is indirectly associated. The route marked with b is the reverse path of a beacon originated by the level 1 buoy B, with which D is directly associated.

5.1 Modifications to the Buoy Protocol to Support Routing

With the assumption of bidirectional wireless links, a data packet can traverse reverse paths of beacon packets toward the target cells. Some modifications need to be applied to the buoy protocol to support using the reverse paths of beacon packets.

1. **The ID of the previous hop:**

   We need to add to a beacon packet the ID of the immediate previous hop node, because a receiving node needs to know which node is the previous upstream node of the beacon packet. A field called *previous hop* is added to the header of a beacon packet. Before a forwarding node rebroadcasts the beacon packet, the node updates this field with the node's own ID. Hence, the next downstream node of the beacon packet can know from which node the packet directly was received. When the downstream node wants to send a data packet through the reverse path of the beacon packet, it uses the previous hop of the beacon packet as the next direct hop. The *ID* can be an IP or MAC address of the node. If a MAC protocol like IEEE 802.11 is used, the previous hop may not be an extra field of the packet header because 802.11-like MAC protocols require the transmitting node to attach its own MAC address into the packet header during a wireless transmission. In that case, a beacon forwarding node only needs to examine the MAC header of the beacon packet to get the ID of the previous hop.

2. **Expanded beacon broadcast range:**

   In the previously introduced buoy protocol, beacons have a broadcasting range of D_n hops, which is designed to let the buoys cover their own cell. However, SHARP requires that all the nodes in peer cells (the cells which share the same parent, i.e., the cells in the same super cell) should know how to route to each other. Therefore, a node should also know reverse paths to the buoys of its peer cells. In order to

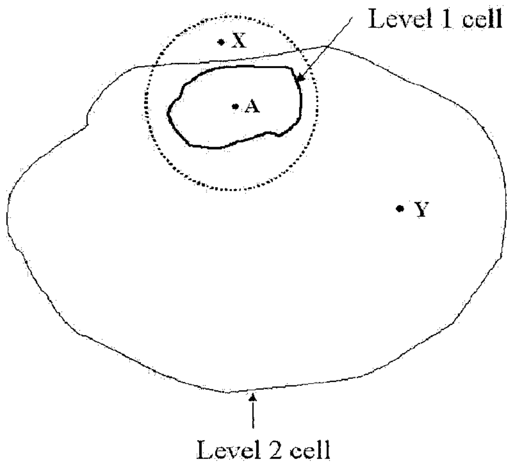

Figure 5.2: Illustration of the scope of a beacon flood

satisfy this requirement, beacon broadcasting ranges need to expand to peer cells. As illustrated in Figure 5.2, the beacon broadcasting range of a buoy A is the union of two sets: the nodes within the circle with a radius of D_n and the nodes within the same super cell. In summary, a node B forwards a level n buoy A's beacon packet as long as either of the following two conditions is satisfied:

(a) The distance between B and A is less than D_n, or

(b) $HID_A[n+1] = HID_B[n+1]$, which means the $(n+1)$th entries in their hierarchical addresses are the same. In other words, A and B have their level $n+1$ cell in common.

If the second condition is satisfied, the beacon packets of A are forwarded all the nodes in peer cells, allowing these nodes to learn routing information toward A's cell.

5.2 Inter-cell Routing Using Beacon's Reverse Paths

Inter-cell routing is the first phase of the SHARP routing protocol. Given the HID of a destination node, the task of inter-cell routing is to deliver a packet to the fundamental cell where the destination node is located. The basic strategy of inter-cell routing is to send packets along the reverse paths of some beacons that were heard previously. These beacons were originated by the buoys with which the destination is associated. Specifically, a packet is first delivered toward a high level buoy with which the destination node is associated, and then it is delivered toward a lower level buoy with which the destination node is associated. Repeating this procedure, traversing toward a lower and lower level buoy, the packet will finally reach the fundamental cell of the destination node.

5.2.1 Buoy ad hoc routing table (BART)

Buoy ad hoc routing table (BART) is the inter-cell routing table that stores the paths to buoys. The data structure of the table entry is the same as the entry in the beacon cache mentioned in Chapter 4. An entry in a BART records

- the buoy's HID,
- the beacon level,
- the sequence number,
- the distance in hop count away from the buoy, and
- the node ID of the previous hop in the path of the beacon packet.

The node ID of the previous hop will be used as the direct next hop if the node has a packet to deliver toward the buoy's cell.

Unlike a beacon cache, which keeps a history of the beacons, a BART keeps only the most recent beacon for any given buoy. This is because the membership decision concerns the hysteresis of the beacons while routing only concerns the freshest information. In my simulation implementation, the beacon cache and BART share the same memory space in a node, and a look up algorithm is used to find the newest entry in a beacon cache for any given buoy.

5.2.2 Inter-cell routing

Packet initialization

During inter-cell routing, a data packet contains the following fields in its header until the packet reaches the fundamental cell of the destination:

- **HID of the destination:**

  The hierarchical address of the destination node is initialized by the packet source.

- **Length (prefix match length):**

  The *prefix match length* is the length of the common IDs between the HIDs of the destination node and of the BART entry that is currently used to forward the packet. We define the *prefix match length* between two HIDs $[B_nB_{n-1}\cdots B_1]$ and $[C_nC_{n-1}\cdots C_1]$ as the largest integer k so that $B_i = C_i, \forall i, n \geq i > n-k$.

- **Sequence number:**

  The sequence number of the packet is initialized to the sequence number from the BART entry that is currently used to forward the packet.

- **Hop count:**

  The hop count records the distance from the BART entry that is currently used to forward the packet. This field, together with the prefix match length and the sequence number, is called the *3-tuple*, and they are updated at every intermediate hop after the node finds a BART entry to route the packet.

The HID of the destination is attached to the packet so that the forwarding nodes do not have to do separate DHT address retrieving operations. The 3-tuple is used by the forwarding nodes to determine the quality of their current routing information for forwarding the packet. The prefix match length determines toward which target cell the packet is currently being forwarded. The sequence number indicates the freshness of the entry, and the hop count describes the distance of the target cell. All the forwarding nodes need to know the quality of the current routing information because they have to use information of equal or higher quality to forward the packet. I will show in the next subsection that this requirement is designed to avoid routing loops.

When a node has a packet to send to a destination, it first uses an address look up protocol to get the destination's hierarchical address. The source node then attaches the destination's hierarchical address to the packet. The source node also needs to set the values of the 3-tuple at zeros. Thereafter, the node applies the following algorithm to deliver the packet.

Packet forwarding algorithm

When a node receives a packet or has a packet to send, it retrieves the destination's hierarchical address from the packet and checks if the destination is within the same fundamental cell. If so, the node invokes the intra-cell routing algorithm to deliver the packet. Otherwise, it applies the following algorithm:

1. The node searches its BART to find the entry with the longest matched prefix with the destination's HID.

2. If the prefix match length of the entry is greater than the *prefix match length* contained in the packet, then the node updates the values of the *length*, *sequence number*, and *hop count* in the packet with the BART entry. The packet is then forwarded to the node indicated by the previous hop in the BART entry.

3. If the maximum prefix match length is equal to the *prefix match length* contained in the packet and if an entry's sequence number is no less than the *sequence number* in the packet, the node uses that BART entry to forward the packet after updating the *sequence number* and *hop count* in the packet.

4. If neither of the above two conditions is satisfied, then the node invokes *local route repair* as described in Section 5.3.

Every forwarding node applies the above algorithm. The value of the prefix match length attached in the packet never decreases as the packet traverses toward the destination, and the HID of the forwarding node also shares longer and longer matched prefix with the destination's HID. Finally, the packet reaches a node that is in the same fundamental cell with the destination. From there, the packet will be applied intra-cell routing to reach the destination, and the extra header carried for inter-cell routing will be removed.

Inter-cell routing is loop free. For any given time a node can only use only one unique BART entry to forward a packet. Moreover, BART entries give the reverse paths recorded from the beacons, which are always trees and hence loop-free. The paths traversed by data packets are composed of branch segments from different trees. When a packet is delivered toward a buoy, the packet always travels upward toward the root along the branches of the corresponding tree, which must be loop free. Finally, when the prefix match length increases during the packet delivery, it means that the packet has jumped to another tree since now the packet is delivered toward a different buoy. Because the forwarding algorithm requires that the packet can traverse from a node with some prefix match length only to a node with greater than or equal to prefix match length, the packet will never jump back to the trees that have been traversed before. Therefore, the entire traversed path of a packet is loop free.

5.2.3 A metaphor of inter-cell routing

Inter-cell routing in SHARP can be considered as water flowing in nature. Water always flows from a place of high altitude to a place of low altitude. For any single place, there is only one altitude. As long as the law of gravity is the only determining factor, water flows do not generate loops: Water flows out of a place and will never come back.

Our inter-cell routing shares the same attribute as water flows. When a packet is routed from the source to the destination, hop by hop, the intermediate forwarding nodes are becoming nearer and nearer to the destination, not just in the sense of geographical or topological distance, but also in the sense of the distance in Safari's hierarchical structure: Forwarding nodes share a longer and longer common prefix with the destination in their hierarchical addresses as the packet moves toward the destination.

We can define this hierarchical distance between a forwarding node and a destination node as the "*altitude*" relative to the destination. Since the altitude only matters in the context of comparison, the altitude should be defined as a comparative order rather than an absolute value. Additionally, a node may have a high altitude relative to a destination node that is far away, but it may have a low altitude relative to a nearby destination node. Finally, because the altitude is defined for routing, it is useful only when the high node knows which nearby node is low, so that it can send the packet to the low node. Therefore, a node should not use its own HID to get its altitude; instead, a node should use its BART entries to get the altitude relative to the destination. For example, if a hierarchically remote node has a BART entry of the fundamental buoy of the destination, this node has a very low altitude relative to the destination, and its routing information is good. In summary, like the 3-tuple, altitude can be conceptually used to measure the quality of the routing information.

Figure 5.3 illustrates the conceptual relationship between altitude and routing in SHARP. Here, a source node S is sending a data packet to a destination node D. The altitude decreases along the path from S to D.

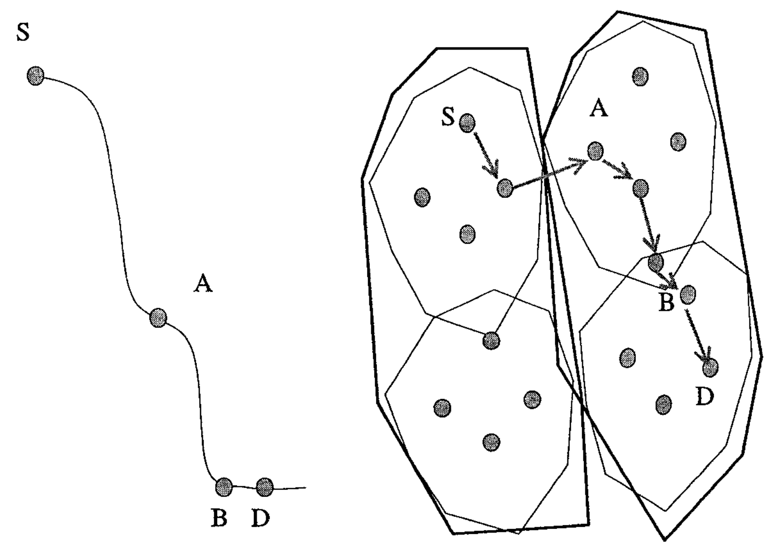

Figure 5.3: A metaphor of inter-cell routing

Given a destination B, we can define the *altitude* as a comparative relationship between two given BART entries x and y:

- When x shares shorter prefix match in the hierarchical address with B than does y, then $Alt_x > Alt_y$.
- When x and y have the same length of prefix match with B and when x's sequence number is smaller than y's, then $Alt_x > Alt_y$.

- When x and y have the same length of prefix match with B, their sequence numbers are equal, and x's distance in hop count from the buoy is greater than y's, $Alt_x > Alt_y$.

The altitude of a node A can be defined as the minimum altitude of its BART entries:

$$Alt_A = min(Alt_x), x \in BART(A) \ .$$

With the above definitions, inter-cell routing behaves like water flowing. Every node has only one unique altitude. A packet is allowed to flow only from a higher altitude node toward a lower altitude node. Therefore, once a packet flows out of a node, it will never come back.

5.3 Local Route Repair to Improve Inter-cell Routing Performance

5.3.1 Overview

The SHARP inter-cell routing scheme requires a node to forward a packet along reverse paths of buoy beacons, and it requires the packet to be forwarded by a node that has a BART entry of an equal or longer matched prefix than the previous hop. Unfortunately, the above requirements cannot always be satisfied during packet deliveries. The following situations may happen to thwart relaying a packet:

1. Because of the asynchrony of the distributed nodes, when a node receives a packet which is following the reverse path of a given buoy's beacon, the corresponding beacon entry may have already expired. It is also possible that the corresponding beacon entry has been deleted because the node found that the next hop moved away. We can say that the routing information has been *forgotten* in this situation.

2. Because of a partition of the network or poor conditions of the wireless channels, some buoy packets may not be able to cover their designed scope. When this happens,

a packet may not be able to be forwarded by a node because the node cannot find a beacon entry with an equal or longer matched prefix. We can say that the routing information is *unavailable* in this situation.

3. Because of the mobility of nodes or poor conditions of the wireless channels, when a forwarding node tries to send a packet to the next hop, the transmission may fail in the MAC or physical layer. This failure can be detected by the MAC layer if the MAC protocol uses an IEEE 802.11-like send-and-acknowledgment mechanism [14]. In this situation, we can say the routing information is *stale*.

When the above situations happen, if the forwarding node does not do anything but discards the packet silently, it will significantly hurt the inter-cell routing performance.

In the traditional DSR scheme, stale routing information can also stop a packet from being relayed to the next hop. A mechanism called *local route repair* helps a DSR forwarding node find an alternate route by initiating a range-limited route discovery packet in the forwarding node.

Although SHARP's inter-cell routing is very different from the purely on-demand DSR routing, we still get inspired by the DSR's local route repair strategy.

5.3.2 The challenge of the local route repair

A naive way to adapt the DSR's local route repair strategy into the SHARP inter-cell routing is to cause the failed node to originate a route request around its neighborhood. If some node has a BART entry to the destination, it can reply the request. Then the failed node records the reply into its BART table and uses the entry to forward the failed packet. However, this may lead to routing loops.

Different from DSR, which mandates the packet initiator to attache the whole path into the packet header to guarantee loop freedom, SHARP inter-cell routing is basically a hop-by-hop forwarding scheme. During the local route repair, it is possible that packets may get trapped in a loop if a repair initiator blindly accepts any routing information around it.

In order to guarantee loop freedom of the packet, we still need the altitude information to guide our local route repair.

5.3.3 Local route repair algorithm

Triggered by a failed search in the BART or by an unsuccessful wireless transmission, *Local Route Repair* is invoked by the forwarding node to find an alternate route for the packet.

Route requesting and forwarding

The repair initiator sends out a LOCAL ROUTE REQUEST packet to its neighborhood after the node buffers the failed packet. A LOCAL ROUTE REQUEST packet contains the following five fields:

- HID of the final destination
- prefix match length of the beacon that is currently being followed
- sequence number of the beacon that is currently being followed
- hop count of the beacon that is currently being followed
- transmission hop count of the LOCAL ROUTE REQUEST packet.

The first four fields are already attached in the data packet during inter-cell routing, and they are updated hop by hop during the delivery. The route repair packet should copy their values from the data packet. If the data packet forwarding fails because the node cannot find a valid entry in its BART, those fields of the data packet were lastly updated by the previous hop. If the data packet forwarding fails due to a MAC transmission failure, those fields of the data packet were lastly updated by the current node.

If a node receiving the LOCAL ROUTE REQUEST can find a BART entry that has better values of the 3-tuple than the values in the request packet, the node replies to the request initiator. Otherwise, the node increases the packet's transmission hop count by one and rebroadcasts the packet. If the transmission hop count is greater than the pre-configured threshold, the node drops the packet.

Similar to DSR, a LOCAL ROUTE REQUEST packet also records its traversed path in the packet header so that a replying node can use source routing to send back its reply along the reverse path of the request packet.

The send buffer in the request initiator stores all the undeliverable packets, and it keeps a timer for every entry. If an entry expires, the node initiates another local route request for the packet. If there is no new routing information to send a packet, the packet will finally get dropped after several times of local route repairs.

There is also a ROUTE REQUEST TABLE in the request initiator recording all the outstanding route request packets. It records when and for which node a route request packet was originated. This table prevents sending route request packets too often for the same target node. If a node wants to send a route request for a target, the node should first check its ROUTE REQUEST TABLE to make sure there are no outstanding request packets for the same target in the recent history.

Route replying and forwarding

After a node receives a local route repair request, it replies to the requester if it can find a BART entry to satisfy one of the following conditions:

1. The maximum prefix match length of the BART entry is greater than the value in the request packet header.

2. The maximum prefix match length of the BART entry is equal to the value in the request packet, but the BART entry has a higher sequence number.

3. Both the prefix match length and the sequence number are equal to the values in the request packet, but the BART entry indicates a smaller hop count away from the referred buoy.

If one of the above conditions is satisfied, the node generates a LOCAL ROUTE REPLY packet and uses the reverse route in the request packet to send the reply packet back to the requester. The node also attaches the 3-tuple of the BART entry to the reply packet so that the intermediate forwarding nodes can know the quality of the replying routing information.

Each node that receives a LOCAL ROUTE REPLY packet builds an entry in its own BART using the values of the 3-tuple (prefix match length, the sequence number, and the hop count) in the corresponding LOCAL ROUTE REQUEST packet. The node also records the node ID of the previous hop into the BART entry. Thus, when a salvaged data packet later is received for forwarding, the node can use this BART entry to send the packet toward the node that replied to the request, and the forwarding functions like the normal inter-cell routing behavior.

A forwarding node of a reply packet may potentially forward multiple reply packets for the same request. The forwarding node should not forward a reply packet if it has already forwarded another reply packet that provides better routing information. Therefore, a forwarding node of the reply packet records the 3-tuple in the LOCAL ROUTE REPLY packet. When another route reply packet is received, the node compares those stored values against the ones in the new request packet. if the new packet provides better information, it updates the stored record and forwards the new reply packet. Otherwise, the new reply packet is dropped.

When an initiator of a LOCAL ROUTE REQUEST packet finally receives a reply packet, it builds a new BART entry using the 3-tuple in the corresponding route request packet. The node also records the node ID of the reply packet's previous hop into that BART entry. Finally, the node examines its send buffer to transmit the corresponding packets using the new updated BART entry. The packet is delivered thereafter by normal inter-cell routing functions since all the nodes along the salvaged path have a valid BART entry to deliver the node.

Optimization to local route repair

The basic scheme described above for local route repair can work well, depending on the pre-configured threshold limit on hop count for forwarding a LOCAL ROUTE REQUEST packet. However, this forwarding of the local REQUEST can be optimized by attempting to forward the REQUEST only in the direction towards the destination cell, avoiding the overhead of searching for a local repair to the route in the wrong direction. In this optimization, we use the concept of altitude as defined in Section 5.2.3. A hop in forwarding the local REQUEST is considered a "downhill" hop if the node receiving it is at a lower altitude than the node from which it received it, and a hop is considered an "uphill" hop if it is at a higher altitude; we currently count hops at equal altitude as downhill hops. Rather than limiting the REQUEST flood to a pre-configured number of hops in all directions, we limit it to a number of downhill hops.

However, there are two reasons not to strictly limit the forwarding of the REQUEST to only downhill hops. The locations of the nodes may be such that no path currently exists composed only of downhill hops, but by allowing a small number of uphill hops, the flood may be able to reach other downhill hops; this is similar to the use of lateral routing in the DARPA PRNET routing protocol [17]. In addition, the altitude of a node is determined by the beacons it has received, but beacons are sent only periodically. It is possible that a node has moved such that it is now entirely surrounded by nodes whose altitude is higher than its own. Without allowing some uphill hops, such a node would be unable to find any path through local repair. In my simulations of SHARP, I limited the local REQUEST flood to no more than 1 uphill and no more than 4 total hops.

5.4 Intra-cell Routing Using DSR

Every time an intermediate forwarding node receives a packet, it checks if it is in the same fundamental cell with the destination. If it is, it will set an intra-cell routing bit in the packet header, marking that from now on, all the forwarding nodes can apply only the intra-cell routing scheme.

5.4.1 DSR overview

Theoretically, any state-of-the-art ad hoc network routing protocol can be used as an intra-cell routing scheme. Once a packet reaches the same fundamental cell as the destination, the distance between the current forwarding node and the destination is less than five or six hops, for example, which can be handled by any of the traditional ad hoc network routing protocols. We chose DSR to be our basic routing scheme as the intra-cell strategy because previous studies [7, 11] show that DSR performs well against some other popular protocols in small-scale ad hoc networks.

DSR is a totally reactive routing protocol for mobile ad hoc networks. It is based on source routing, which means the originator of a packet decides the whole path of the packet and the intermediate forwarding nodes just follow the path in the packet header.

DSR uses *route discovery* to find a path for a packet. When the packet originator finds that it does not have any paths in its route cache to the packet destination, it broadcasts a ROUTE REQUEST packet to its neighboring nodes. The packet header includes the destination's address as well as a list of the addresses which is initialized as the originator. The receiving node rebroadcasts this ROUTE REQUEST packet locally and also adds its own address to the end of the address list in the packet header. Therefore, the address list records the path traversed by the ROUTE REQUEST packet. When the destination node receives this packet, it uses the address list from the ROUTE REQUEST packet to get the path from the originator to the destination. The node then puts the path into a ROUTE REPLY packet. The destination then uses the reverse path if the network has bidirectional links, or it uses a path from its route cache to deliver the ROUTE REPLY packet back to the ROUTE REQUEST originator.

DSR also uses *route maintenance* to maintain the validness of paths that it uses from its route cache. If a forwarding node finds that it cannot successfully deliver a packet to the next hop specified in the source routing header, it sends a ROUTE ERROR packet back to the packet originator to notify it about the broken link. The originator removes the invalid link from the route cache when it receives the ROUTE ERROR packet.

Studies in both simulations and testbeds have shown that DSR can perform well in a small-scale ad hoc network.

5.4.2 Modification of DSR for intra-cell routing

In traditional DSR, the route discovery range is usually predefined as the diameter of the whole network. Because of this, route discovery becomes more and more expensive as the network size scales up. In intra-cell routing, we know that the destination node is located in the same fundamental cell, so it is appropriate to limit the route discovery range to be within the fundamental cell. Moreover, since different fundamental cells may have different shapes and sizes, it is efficient to decide the route discovery range based on the membership of nodes, instead of on a predefined hop count. More specifically, whenever a node receives a DSR ROUTE REQUEST, it compares its own HID against the initiator's HID attached in the request packet's header. If they belong to the same fundamental cell, the node forwards the packet. Otherwise, the node drops the packet. In this way, we can limit the route discovery range only within the fundamental cell where the destination node is located.

Sometimes the packet originator may get the wrong HID of the destination because the destination changed its own membership before this change reached the packet originator. It is highly possible that the destination node is still not far from its previous cell when the packet reaches the destination fundamental cell. In order to improve the possibility of finding the destination, I added a BEYOND_CELL hop count in the DSR ROUTE REQUEST packet, defining how many hops a ROUTE REQUEST packet can go out of the route requestor's fundamental cell. During a route forwarding of a ROUTE REQUEST, every forwarding node increases the BEYOND_CELL hop count by one if the node is in a different cell from the originator. If this BEYOND_CELL hop count is still lower than a pre-defined threshold, it forwards the ROUTE REQUEST packet. Otherwise, it drops the packet. This mechanism can help a DSR route discovery reach the destination node even if the destination node has moved outside its previous fundamental cell.

5.5 Summary

In general, routing in SHARP is divided into two phases: inter-cell routing and intra-cell routing. For inter-cell routing, the data packet follows reverse paths of the beacons. Therefore, its own routing overhead is trivial. Since the number of levels in the Safari hierarchical structure grows logarithmically as the size of the network grows, the number of beacon packets also grows logarithmically. As to intra-cell routing, SHARP uses a modified version of DSR as the basic strategy. Because the average diameter of a fundamental cell is fixed once the network is configured, overhead of intra-cell routing is fixed as the network size grows. Therefore, we claim that our routing protocol is scalable, both in the sense of the number of routing packets and in the sense of storage space of routing tables. The following chapter evaluates the performance of SHARP through detailed simulations.

Chapter 6

Routing Performance Evaluation

In order to evaluate the performance of the SHARP routing protocol, I implemented the buoy protocol and the routing protocol in the *ns-2* network simulator. I used the default parameters in *ns-2* for the wireless MAC and physical layers settings, with a IEEE 802.11-based 2 Mbps data rate and a nominal transmission range of 250 m.

The beacon broadcasting limit was set to be 3 hops (i.e., $D_1 = 3$), thus allowing the diameter of the fundamental cell to be up to 6 hops. The decision is due to the fact that DSR performs well, without significant overhead, in a network with this diameter. The value of α (D_n/D_{n-1}) was chosen to be 2. The lowest level buoys broadcasted their beacon packets every 1 second, i.e., $T_1 = 1$ second. The value of β (T_n/T_{n-1}) was also set as 2.

Simulations were done in various network topologies with size ranging from 50 nodes to 1000 nodes randomly distributed in a two dimensional topology. The X and Y dimensions of the network varied with the number of nodes in the network so that the network density was kept the same as for a network of 50 nodes in a 1000m×1000m area, which gives the average node per transmission area to be $\frac{50}{1000\times1000} \times (\pi \times 250^2) \approx 10$ nodes. Due to the very large memory consumption of *ns-2*, I had to limit the simulations to a maximum of 1000 network nodes.

I used the random waypoint model to simulate the mobility of the network nodes [16]. In this mobility model, a node randomly picks a waypoint within the whole area and moves toward it at a randomly picked speed between 0 m/sec and 10 m/sec. Once the node reaches the waypoint, it repeats the same pattern of motion again toward a new waypoint at a new speed.

The objective of the Safari architecture is to provide a city-wide ad hoc network service platform. Thus it is highly unlikely that all nodes in the network are mobile simultaneously. Nevertheless, I studied the performance of SHARP in the worst case scenarios of all nodes being mobile.

I used constant bit rate (CBR) flows to generate the traffic loads on the networks. Each flow consisted of a randomly chosen pair of source and destination nodes. Each flow lasted 90 seconds and generated 64-byte packets at a constant rate of 4 packets per second. This traffic patterns is more challenging than the typical continuous long lifetime CBR flows, as flows in the latter setting can use the same cache information for a long time. It is also realistic to have multiple short flows instead of a few long flows.

During the simulation, I allowed 350 seconds of simulation time for purely running the buoy protocol in order to get the network stabilized, and thereafter the flows started arriving according to a Poisson distribution. Each simulation ran for 900 seconds of simulation time.

In the simulations in Section 6.1, two different traffic patterns and four different mobility patterns were used to provide an average of eight runs for each data point. In the simulations in Section 6.2 and Section 6.3, two different traffic patterns and two different mobility patterns were used to provide an average of four runs for each data point. The error bar is also shown with the standard deviation of the runs for each point.

6.1 Scalability Evaluation

The most important goal of SHARP is to scale to a large-scale ad hoc network. I show that the design meets this goal by evaluating the Packet Delivery Ratio (PDR) and routing overhead across different network sizes.

PDR is defined as the fraction of application data packets originated that are successfully received by the application layer at the respective destination node. Routing overhead is measured by the number of control (non-data) packets transmitted per node. Overhead packets include the beacon packets, local route request packets, local route reply packets, and the DSR routing packets. Every overhead packet, whether it is generated or forwarded, contributes to this overhead. For example, a single beacon packet forwarded by two nodes is counted as two overhead packets. All the graphs in this section are for networks in which all nodes are mobile and the traffic in the network is 100 CBR flows.

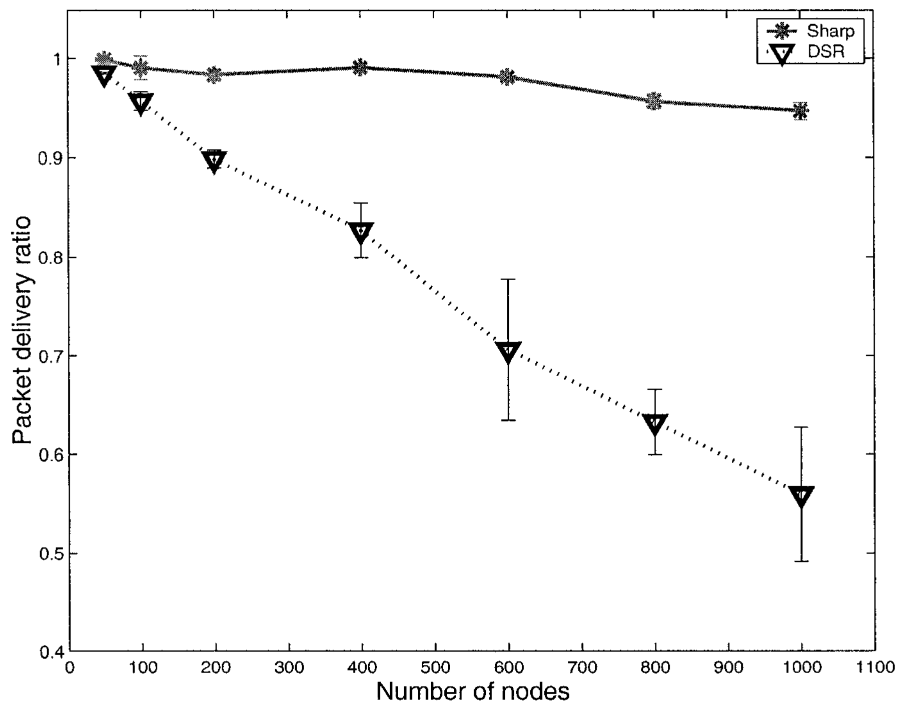

Figure 6.1: Packet delivery ratio vs. network size

Figure 6.1 shows the comparative performance of a popular ad hoc routing protocol, DSR with SHARP. Both routing protocols perform equally well until network size reaches around 100 nodes. After that, the PDR of DSR drops sharply, going down to about 60% for a network consisting of 1000 nodes. SHARP routing, on the other hand, performs very well, with the PDR slowly falling only to about 95% for 1000 nodes. This suggests that SHARP can scale up to much larger networks.

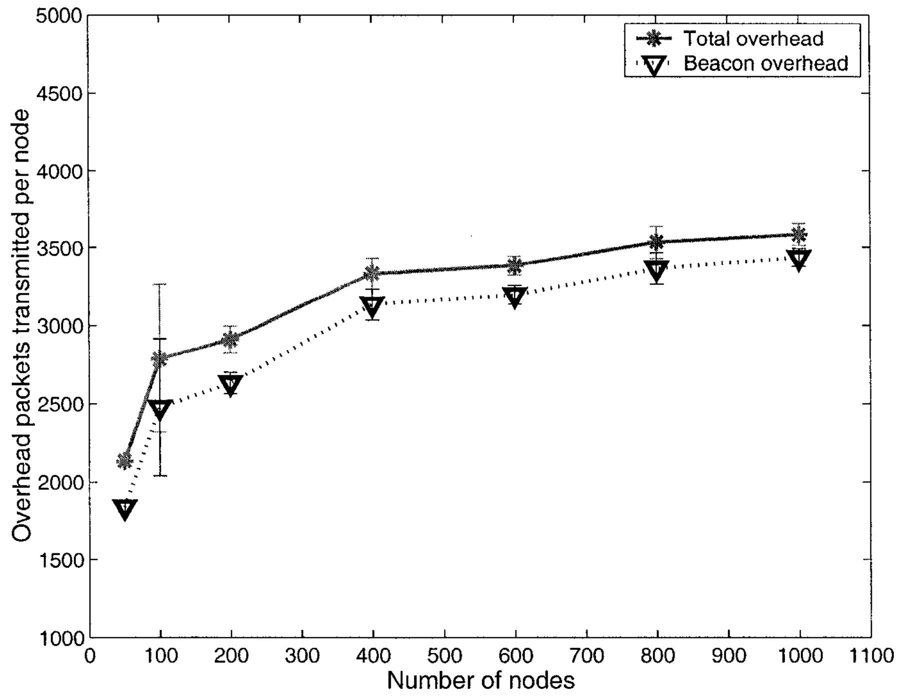

Figure 6.2: Overhead packets per node vs. network size

Figure 6.2 shows the packet overhead versus network size. The total overhead is dominated by the beacon overhead. Local route request, local route reply, and DSR routing packets account for the rest. In the previous work on Safari, Khan et al. [19] proved that in theory the number of beacon packets transmitted per node is constant bounded. In other words, the complexity of beacon overhead is $O(1)$. The simulation results here show that the beacon overhead does appear to approach a constant value, as the curve becomes flatter and flatter when the network size goes to 1000. However, in order to fully confirm our theoretic prediction, simulations in larger networks need to be done, which is now unfortunately limited by the large memory consumption of *ns-2*. Nevertheless, with our current results, it is still encouraging to see that the curve for beacon overhead closely follows the curve for total overhead. In other words, with increasing network size, the overhead per node is likely to be bounded, thus allowing our routing architecture to scale essentially to arbitrary size.

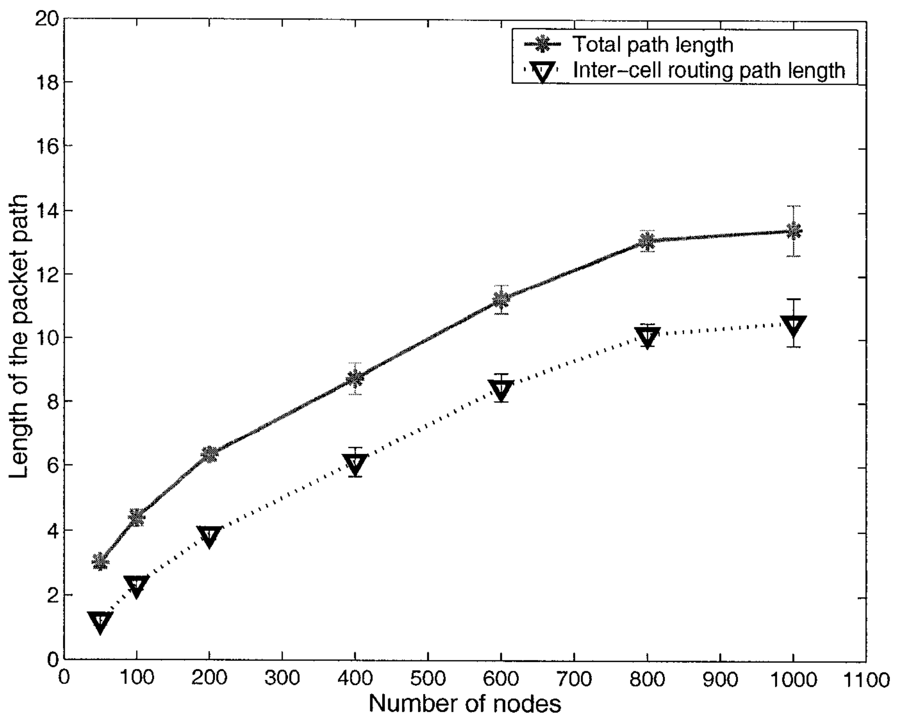

Figure 6.3: Number of hops vs. network size

Figure 6.3 shows the average path length used, in number of hops, with increasing network size. The top line shows the total number of hops to the destination, whereas the bottom line shows only the hops taken by the inter-cell routing. The difference between the two lines shows the number of hops within a fundamental cell; this is almost constant, indicating that the size of the fundamental cell is fixed regardless of the size of the network. The exact value of the difference is 2.9, which is the average number of hops required to traverse a fundamental cell.

6.2 Mobility

In this section, I show that SHARP is able to handle efficiently various degrees of mobility. I fixed the number of nodes in the network at 1000 nodes and the number of CBR flows at 100, and then varied the mobility from 0% nodes being mobile to 100% nodes being mobile.

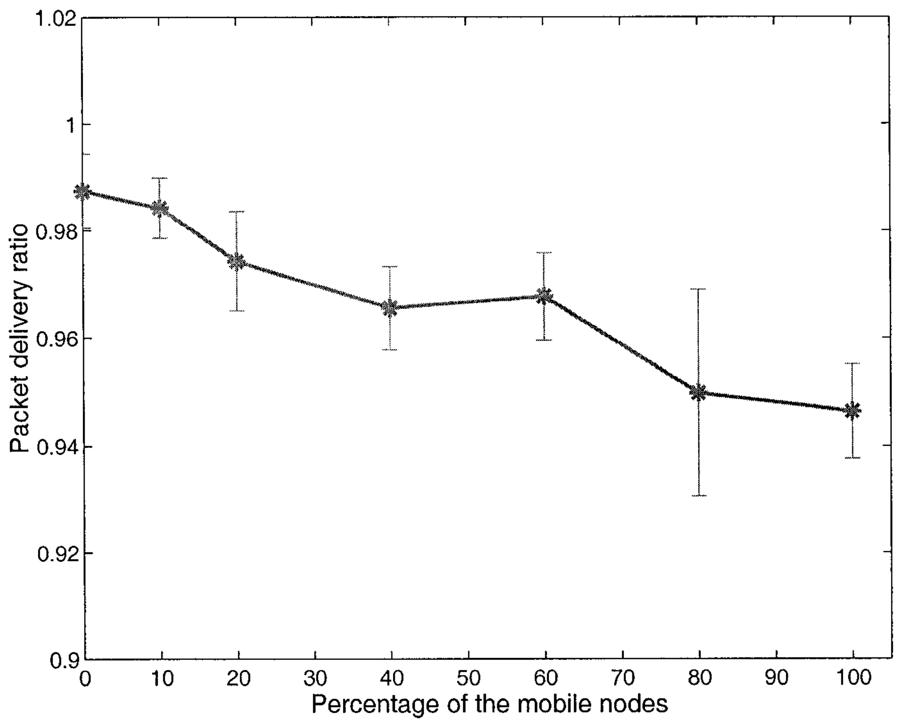

Figure 6.4: Packet delivery ratio vs. percentage of mobile nodes

Figure 6.4 shows the change of PDR with the number of mobile nodes varying from 0% to 100%. The PDR is over 95% even for 100% mobile nodes, showing that the protocol reacts very well to mobility. The packet delivery ratio in the static scenarios (0% mobile nodes) is not 100%, though. This is mainly due to the existence of a permanent network partition in one of the four mobility scenarios I used. In that scenario, three nodes are partitioned from the rest of the network due to the random distribution and stationary position of the nodes. Since there are packets generated from and packets destined to those three nodes, the packet delivery ratio drops due to these undeliverable packets.

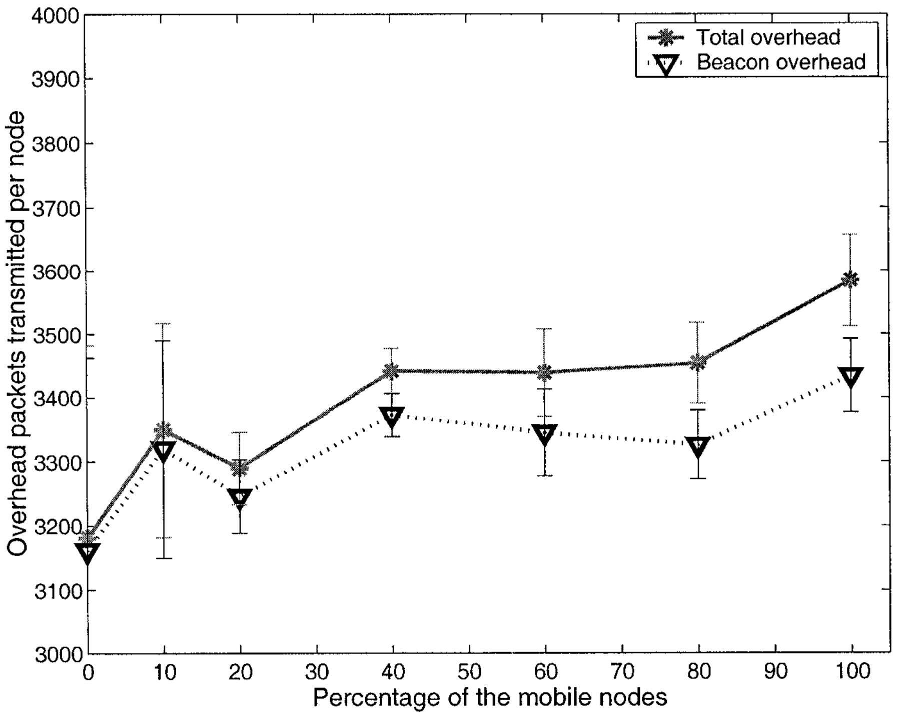

Figure 6.5: Overhead packets per node vs. % mobile nodes

Figure 6.5 shows the packet overhead with the increase of mobility. The beacon overhead shown by the bottom curve varies little. Due to the lack of any reactive overhead, the total overhead is almost the same as beacon overhead in a static network (0% mobile nodes). As the percentage of mobile nodes increases, the reactive component increases, which makes the total overhead diverge from the beacon overhead. For networks with low mobility, there is a small fluctuation with large error bars in the curve of the beacon overhead. This is basically because beacon overhead is much more sensitive to the nodes' initial distribution in a network with low mobility than in a network with high mobility, since mobile nodes can randomly change the topology of networks.

6.3 Traffic Load

I show the SHARP routing performance with varying traffic loads. The number of nodes is constant at 1000, and all the nodes are mobile.

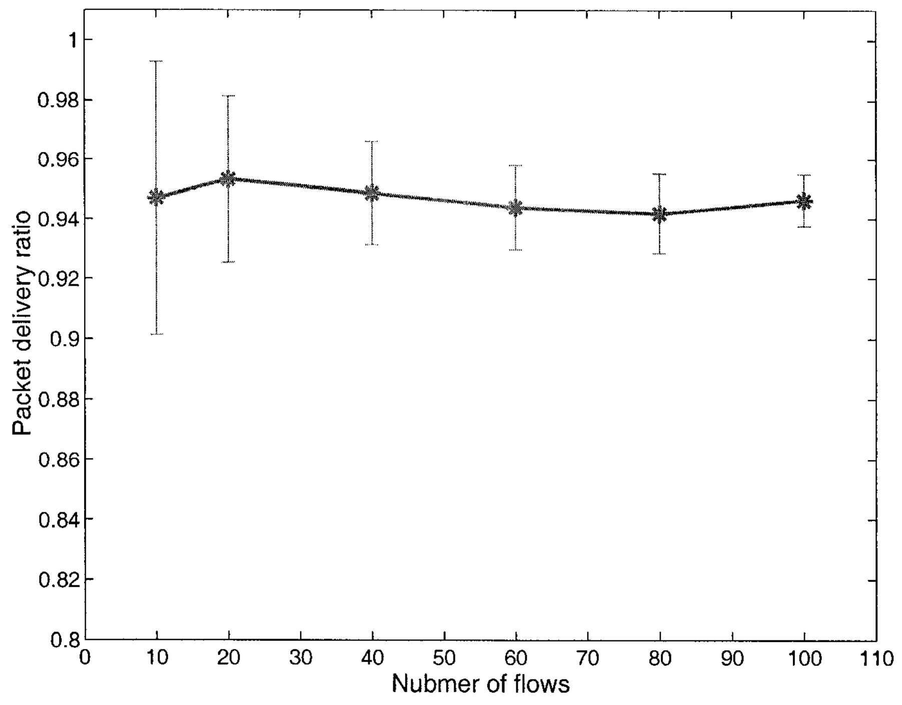

Figure 6.6: Packet delivery ratio vs. traffic load

Figure 6.6 shows the PDR versus number of flows. The PDR changes little with this increase of load. The error bars are smaller for larger number of flows because the total number of packets being sent is more for larger number of flows, thus they smooth the statistical variabilities in the results.

The total overhead increases slightly with the increase of traffic load from 10 to 100 CBR flows, as is shown in Figure 6.7. This increase is from higher reactive component because of the increased number of flows. The beacon overhead remains almost constant. There is a small fluctuation in the curve of beacon overhead. This is because traffic loads

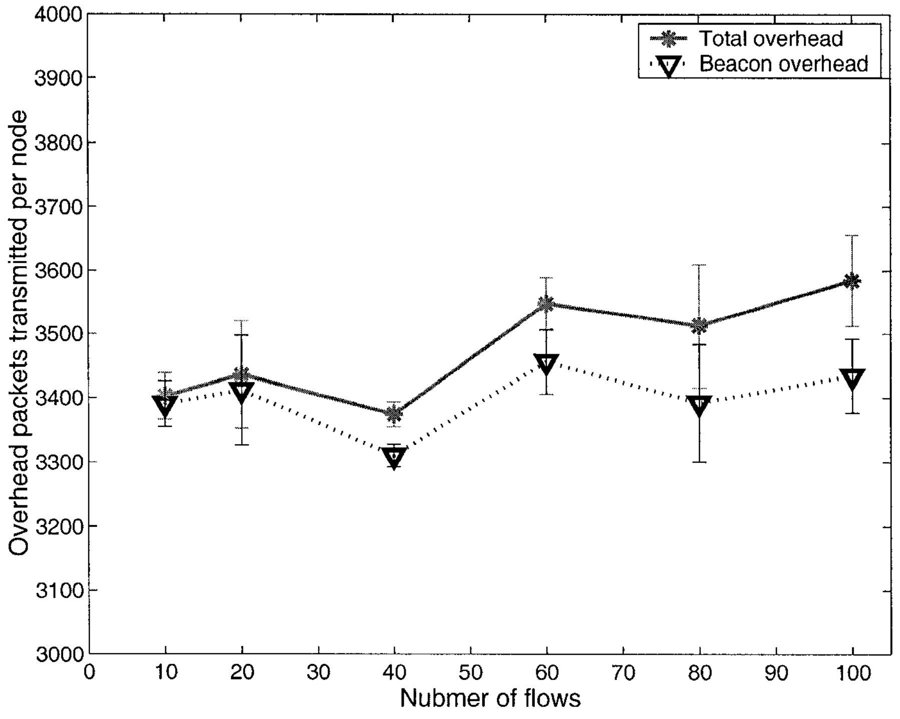

Figure 6.7: Overhead packets per node vs. traffic load

can interact with the forwarding of beacon packets. When the traffic in an area is high, it may cause some beacon packets to traverse longer paths to bypass the area or may cause some other beacon packets to get dropped inside the area.

Chapter 7

Conclusion and Future Work

In this thesis, I presented the design and evaluation of new techniques to solve the routing problem in large-scale city-wide wireless ad hoc networks. Safari [12, 33], composed of a proximity-based hierarchical address allocation protocol and a scalable hybrid routing protocol, can provide a supportive platform in ad hoc networks for Internet services. In order to evaluate the performance of the Safari architecture, I implemented Safari's buoy and SHARP routing protocols in the *ns-2* network simulator. During the process of implementation, I also developed its implementation from rough design guidelines and proposed several techniques to improve its routing performance and the stability of the hierarchical structure. The simulation results show that SHARP's hybrid routing is capable of handling a large-scale ad hoc network of thousands of nodes. In the future, I plan to evaluate SHARP's performance in networks of 10,000 nodes to have a better understanding of SHARP's feature of scalability. Because the large memory consumption of ns-2, I can only simulate up to 1000 nodes in this thesis. Some of the SHARP features, such as the slow overhead growth and limit, still need future investigation to confirm our theoretical analysis. With the simulations of much larger networks, theses features can be better observed. I also plan to try to continue improving SHARP's inter-cell routing performance by delivering packets cell-by-cell, instead of hop-by-hop. Compared to the dynamic topology of nodes, the topology of cells in a Safari architecture is much more stable. Therefore, the cell topology information should provide us more precise information for routing than the currently used node topology information.

Bibliography

[1] Suman Banerjee and Samir Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. *IEEE Infocom 2001*, April 2001.

[2] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *Proceedings of Vehicular Technology Conference*, 1999.

[3] S. Basagni, I. Chlamtac, and A. Farago. A generalized clustering algorithm for peer-to-peer networks. In *Workshop on Algorithmic Aspects of Communication*, 1997.

[4] S. Basagni, I. Chlamtac, V. Syrotiuk, and B. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of the Fourth International Conference on Mobile Computing and Networking* (MobiCom'98), October 1998.

[5] L. Blazevic, S. Giordano, and J.-Y. Le Boudec. Self organized terminode routing. *Cluster Computing*, April 2002.

[6] Ljubica Blazevic, Silvia Giordano, and Jean-Yves Le Boudec. Anchored path discovery in terminode routing. In *Proc. of the 2nd IFIP-TC6 Networking conference (Networking 2002)*, May 2002.

[7] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta G. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual ACM/IEEE International*

Conference on Mobile Computing and Networking (MobiCom'98), pages 85–97. ACM/IEEE, October 1998.

[8] M. Chatterjee, S. Das, and D. Turgut. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 2002.

[9] Benjie Chen and Robert Morris. L+: Scalable landmark routing and address lookup for multi-hop wireless network. Technical Report MIT-LCS-TR-837, Laboratory for Computer Science Massechusetts Institute for Technology, 2002.

[10] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proc. ACM SOSP'01*, Banff, Canada, October 2001.

[11] Samir R. Das, Charles E. Perkins, and Elizabeth M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies* (INFOCOM 2000), pages 3–12, 2000.

[12] Shu Du, Muhammed Khan, Santashil PalChaudhuri, Ansley Post, Amit Saha, Rudolf Riedi, Peter Druschel, and David B. Johnson. Self-organizing hierarchical routing for scalable ad hoc networking. Technical Report TR04-433, Computer Science Department, Rice University, 2004.

[13] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 167–177, August 1998.

[14] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[15] David B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA'94), pages 158–163. IEEE Computer Society, December 1994.

[16] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[17] John Jubin and Janet D. Tornow. The DARPA packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[18] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking*, pages 243–254, 2000.

[19] Muhammed Khan, Rudolf Riedi, David B. Johnson Peter Druschel, and Richard Baraniuk. Analysis of safari: An architecture for scalable ad hoc networking and services. Technical Report Master Thesis, Electrical and Computer Engineering Department, Rice University, December 2003.

[20] Y.-B. Ko and N.H. Vaidya. Location-Aided Routing (LAR) in mobile ad hoc networks. *Proc. of MOBICOM98*, October 1998.

[21] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. In *Proceedings of the 7th Annual ACM/IEEE*

International Conference on Mobile Computing and Networking (MobiCom 01), July 2001.

[22] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographical ad hoc routing. In *Proc. of ACM MOBICOM 2000*, Boston, MA, 2000.

[23] Wen-Hwa Liao, Yu-Chee Tseng, and Jang-Ping Sheu. GRID: A fully location-aware protocol for mobile ad hoc networks. *Telecommunication Systems*, 18(1-3):37–60, 2001.

[24] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.

[25] A. McDonald and T. Znati. A mobility-based framework for adaptive clustering in wireless ad hoc networks. *IEEE Journal of selected areas in communications*, August 1999.

[26] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM'97*, pages 1405–1413. IEEE Computer Society and Communications Society, April 1997.

[27] Guangyu Pei, Mario Gerlan, and Xiaoyan Hong. Lanmar: Landmark routing for large scale wireless ad hoc networks with group mobility. In *SIGCOMM'88*, Stanford, CA, 1988.

[28] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of the SIG-*

COMM '94 Conference on Communications Architectures, Protocols and Applications, pages 234–244. ACM, August 1994.

[29] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of the SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244. ACM, August 1994.

[30] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[31] Ananth Rao, Sylvia Ratnasamy, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *Proceedings of the Ninth International Conference on Mobile Computing and Networking* (MobiCom'03), pages 96–108. ACM, September 2003.

[32] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, San Diego, CA, August 2001.

[33] Rudolf Riedi, Peter Druschel, Y. Charlie Hu, David B. Johnson, and Richard Baraniuk. Safari: A scalable architecture for ad hoc networking and services. Technical Report NSF-ANI-0338856, Computer Science Department, Rice University, 2003.

[34] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.

[35] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. ACM SOSP'01*, Banff, Canada, October 2001.

[36] S. Srivastava and R.K. Ghosh. A cluster based routing using a *k*-tree core backbone for mobile ad hoc networks. In *Proceedings of the 6th international workshop on discrete algorithms and methods for mobile computing and communications*, September 2002.

[37] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIG-COMM'01*, San Diego, CA, August 2001.

[38] Terminode. `http://www.terminodes.org/`. Checked at March 2004.

[39] Hongwei Zhang and Anish Arora. GS3: Scalable self-configuration and self-healing in wireless networks. *21st ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2002.

[40] B.Y. Zhao, J.D. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.