Self-Organizing Hierarchical Routing for Scalable Ad Hoc Networking

Abstract

As devices with wireless networking become more pervasive, mobile ad hoc networks are becoming increasingly important, motivating the development of highly scalable ad hoc networking techniques. In this paper, we present the design and evaluation of a novel protocol for scalable routing in ad hoc networks as part of the Safari project. We develop a probabilistic, self-organizing network hierarchy formation protocol that recursively forms the nodes of the ad hoc network into an adaptive, proximity-based hierarchy of cells. We develop a hybrid routing protocol that uses this hierarchy as well as on-demand reactive routing to scale to large numbers of nodes. The mapping of unique node identifiers to hierarchical addresses is done using a distributed hash table that leverage the hierarchical network structure. We evaluate this scalable routing protocol through analysis and simulations, showing individually as well as together the performance of the hierarchy formation protocol, the overhead of address mapping, and the performance of the routing protocol.

1 Introduction

Ad hoc networking is an attractive technology for providing wireless internet access and other wireless communications services for mobile users, but current ad hoc network routing protocols are limited in scalability. In an ad hoc network, individual mobile nodes cooperate to form a network without the aid of any existing communications infrastructure such as base stations or access points. Instead, each mobile node acts not only as a host but also as a router, forwarding packets for other mobile nodes to allow nodes beyond direct wireless transmission range of each other to communicate.

With the rapid proliferation of wireless devices, the use of ad hoc networking is likely to grow, and with it, the size of ad hoc networks that may be created. At the same time, the field of peer-to-peer, or decentralized, self-organizing distributed systems has seen significant interest and advances in recent years and opens new alternatives in providing services in large-scale networked systems. Leveraging these two areas that have previously worked mostly independently, the *Safari* project is developing protocols and algorithms for providing large-scale mobile wireless network connectivity and basic network services, exploiting a tight integration between ad hoc networking and peer-to-peer networking.

In this paper, we present the design and evaluation of a new self-organizing hierarchical routing protocol for scalable ad hoc networking, as part of the Safari project. The routing protocol is based on a probabilistic, self-organizing network hierarchy formation protocol that recursively forms the nodes of the ad hoc network into an adaptive, proximity-based hierarchy of cells. Routing of packets is guided by this hierarchy and by on-demand reactive routing, forming a *hybrid* routing protocol capable of scaling to large numbers of nodes. The mapping of unique node identifiers to hierarchical addresses is done using a distributed hash table that leverage the hierarchical network structure.

We evaluate our routing protocol through both analysis and simulations, showing the performance of the hierarchy formation protocol, the overhead of address mapping, and the performance of the routing protocol. Our evaluation addresses increasing size of the network in number of nodes, increasing mobility in the fraction of nodes that are mobile vs. stationary, and increasing traffic load in number of data packet flows. Our analysis is well matched by our simulations, and our results demonstrate the protocol's scalability.

In Section 2 of this paper, we describe the details of the Safari protocol architecture, including its self-organization, routing, and address resolution components. Section 3 presents our our modeling and analysis of Safari, and Section 4 presents our performance evaluation of it based on detailed simulation. In Section 5, we discuss related work in the area of scalable ad hoc networking, and in Section 6, we present conclusions.

2 Safari architecture

In this section, we provide a high-level overview of the basic Safari architecture, and then present the various components of Safari routing in more detail.

2.1 Overview

Safari is a hybrid, hierarchical routing protocol, using both proactive and reactive routing components. The hierarchy is formed as a recursive organization of nodes into cells, cells into supercells, and so on, based on an automatic self-selection of a subset of the nodes to operate as *drums*. In general, level k cells are grouped into level $k+1$ cells, and for simplicity, we refer to individual nodes as level 0 cells. We call level 1 cells also *fundamental cells*, as at this level, the cell is composed only of individual nodes.

The drums are organized hierarchically, as well, with a subset of the individual nodes self-selecting to become level 1 drums, and iteratively level k drums self-selecting to become level $k+1$ drums. A level k drum is at the same time also

a level i drum for all $i \leq k$. Each drum has a unique identifier, and each drum at level i identifies a cell at level i. The drum selection is based on a distributed algorithm requiring no coordination. Nodes of the same level are roughly equally spaced (in terms of hop counts) throughout the total ad hoc network. The drum identifiers create a hierarchical address for each node as the identifier of the drum for the cell of which the node is a member at each level.

Each drum disseminates locality, hierarchy, and routing information by transmitting periodic *beacon* packets, which are forwarded by all nodes within a well-defined scope in the network. A beacon hop count and an identifier for the originating drum provides nodes with a sense for their location within the network, which they store in the network using a distributed hash table (DHT); carefully choosing multiple storage nodes improves robustness and efficiency of lookup. The drums do *not* have any special role in data packet forwarding; they simply provide a sense of location and direction that is used in routing.

For sending a data packet, the packet is routed according to the hierarchical address (or *coordinates*) of the packet's source and destination, recursively routing *towards* the drum for the destination node's cell at each level, until reaching *any* node with routing state for a lower level cell of which the destination node is a member based on its hierarchical coordinates. To route towards a drum, Safari normally follows the reverse path of the most recent beacon from that drum. Once the data packet is received by *any* node within the destination's fundamental cell, that node initiates an on demand route discovery for the target node (if it does not already have a previous cached route) to complete the route. Local route repair allows Safari to overcome route failures in an on-demand manner and reduces the need for more frequent beacons; routing state for a drum is followed by data packets until new state is recreated by a later beacon.

We now provide a more detailed description of the components of the Safari architecture The Safari architecture owes its scalability to the following features of the architecture:

Transparency: Beacons maintain hierarchy, keep locality information up to date, and provide crucial routing information matched to the locality information.

Hybrid routing: Proactive, hierarchical routing for the main part of network paths, where route discovery is costly, combines ideally with on-demand routing for the "last mile," i.e., the fundamental cell where maintenance of precise location becomes costly.

Distributed solutions: Shared responsibilities, distribution of locality information, and self-selection reduce stress on individual nodes and avoid overhead from voting, electing, and coordinating.

We now provide the design details left unspecified so far and in particular, we address the issues of efficiency and scalability. The three protocols that provide the basics for the Safari architecture are as follows: *self organization*, *scalable routing*, and *distributed address resolution*. Further elements of the Safari architecture which provide higher level services are beyond the scope of this paper.

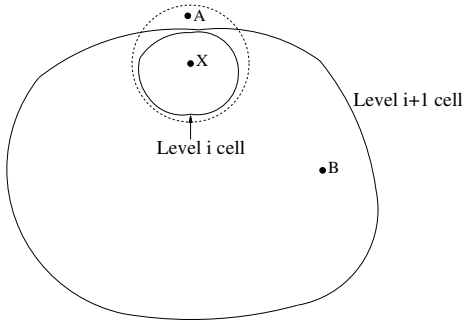

Figure 1: Beaconing scope

2.2 Self organization

In this section, we describe the algorithms that achieve and maintain the desired hierarchical organization of the network, even under node mobility, node failures, and partition and merging of networks. There are three basic mechanisms which allow the Safari architecture to self organize.

2.2.1 Beaconing protocol

Each drum periodically broadcasts an advertisement of its existence along with some useful information. Such a *control* packet from a drum of level n is called a level n *beacon*. Each beacon contains a *beacon sequence number*, a *beacon level and coordinate* which equal those of the emitting drum, and a *hop count* which is set to zero at the emitting drum and incremented by each forwarding node.

A drum at level n transmits a beacon of level n every T_n seconds. These beacons are forwarded by all nodes within D_n number of hops from that drum. This forwarding rule allows beacons to reach all nodes that could potentially associate with the originating drum according to the Membership algorithm described in Section 2.2.3. For example in Figure 1, the dotted circle denotes all nodes within D_i hops of X, a level i drum. Higher level beacons are emitted at a lower frequency than lower level beacons. This is because mobile nodes cross over the regions covered by lower level drums more frequently than they cross over the regions covered by higher level drums. For scalability, T_n and D_n are given by the geometric progressions:

$$D_n = \alpha * D_{n-1} = \alpha^{n-1} * D_1 \quad ; \quad T_n = \beta * T_{n-1} = \beta^{n-1} * T_1 \quad (1)$$

where α and β are system parameters. D_1 is based on the optimal hop count that the on demand routing protocol, used at the level of the fundamental cell, can handle efficiently.

To increase the efficiency of scalable routing (Section 2.3), a beacon of level n is also forwarded by all nodes in the level $n+1$ cell of the originating drum. The exact mechanism by which a cell structure is formed is discussed in Section 2.2.3.

For node X in Figure 1, the beacons are forwarded throughout the level $i+1$ cell of which X is a member. The forwarding of beacons is hence a union of the two forwarding rules mentioned above.

All nodes store the beacons they forward in a cache of beacons, called the Drum Ad Hoc Routing table (DART). This cache is used for self-organization as well as routing. In addition to maintaining all the variables contained in a beacon, a node also stores the *time of reception of the beacon* and the *node identifier from which it received the beacon* in the DART. The latter will allow following of the reverse path of the beacon, while the former allows to keep the cache up to date. Upon receipt of a beacon, a new DART entry is formed and a timer for that entry is started. Whenever a new entry is inserted into the DART or a DART entry expires, the drum level selection and the Membership algorithms are invoked.

The only special function of a drum is to originate beacons. A drum has no special or active role in routing or in the overlay maintenance. In particular, data packets are only routed *towards* but not *through* drums, as will be described in Section 2.3. Thus, all nodes share the beacon transmission workload (origination or forwarding) equally.

2.2.2 Drum level selection algorithm

In order to be efficient under dynamic changes of the network such as mobility and node failures, new drums can form and existing drums can retire. This algorithm runs after each change in the DART and assures that eventually the DART satisfies the following conditions. For a node of level n:

1. the DART contains at least one non-expired beacon of a level $n+1$ drum at most D_{n+1} hops away,
2. there is no non-expired beacon of a level n drum less than $h \times D_n$ hops away ($0 < h < 1$ is a hysteresis factor),
3. there are at least two non-expired beacons of at least level $n-1$.

The way by which the desired state of the DART is achieved is by changing the node's level appropriately such that the invariants are met.

If condition 1 is violated, the level of the node is changed to $n+1$ and the node waits a random back-off time until it announces its new level with its level $n+1$ beacon. This does not apply to the highest level drum; otherwise, the highest level drum would increase its level indefinitely. A drum infers that it is the highest level drum if its DART does not have non-expired beacons from a drum of its level.

If condition 2 is violated, that means that two or more drums of the same level are too close to each other. The drum with the highest node identifier remains at the same level, and all other drums are assumed to reduce their level by 1. The factor h creates a "hysteresis" which prevents oscillations in the drum retirement process. By oscillation, we mean an unstable situation of retirements of drums and formation of new drums to compensate for the retirement, leading to new retirements, and this process continuing unendingly. Indeed, a level n drum retiring could cause condition 1 to be violated for other nearby nodes which then will increase its level. However, these nodes will be at least D_n hops away from any level n drum. Since a conflict between drums requires this distance to reduce to $h * D_n < D_n$ there is no oscillation. For the rest of the paper, we chose $h = 1/2$.

Condition 3 ensures any drum in the Safari tree has more than one branch hanging off it, for efficiency.

2.2.3 Membership algorithm

The presence of drums induces a natural clustering of nodes. Each node *associates* with a drum of level one greater than its own level, and selects this drum according to the contents of its DART. Typically, a node associates with the one higher level drum that is the least number of hops away.

A node's association is made individually and is not communicated back to the drum. A node invokes the membership algorithm after it has run the drum level selection algorithm. In its basic version, this algorithm chooses for a node of level n the closest of all drums of level $n+1$ for which this node has a DART entry that has not expired. To prevent nodes "at the cell border" from switching between drums, the rule to change the associated drum is enhanced by assigning each DART entry a weight calculated from the frequency, the distance, and the number of beacons received. The node associates with the drum corresponding to the DART entry with the highest weight. In our design presented in this paper, we enforce that the new drum is at least 2 hops closer than the current one, and that at least 3 beacons have been received from the new drum. The membership algorithm cannot ensure that the node associates with a drum that is at most D_n hops away. This is the duty of the drum level selection algorithm.

Coordinate selection: The membership algorithm gives a unique ancestry for each node. Using this membership information, each node is assigned a coordinate based on the drum structure. This coordinate plays a vital role in routing.

The Safari hierarchical tree structure fosters a simple scheme to provide coordinates by transferring the locality information encoded in the tree. Every node in the network is assigned a coordinate. The coordinate of a drum at level i is the concatenation of the coordinate of the level $i+1$ drum with which it associates and a randomly generated unique number. Thus, if $COORD(D_i)$ denotes the coordinate of a level i drum, D_i and $PARENT(D_i)$ denotes the level $i+1$ drum with which a node (not necessarily a drum) associates, then we have the following

$$COORD(D_i) = COORD(PARENT(D_i)).RAND(b)$$

where $RAND(b)$ denotes a uniform random number of b bits. With a large value of b the probability that two drums at the same level will chose the same random number can be made

negligible. The coordinate of a leaf node L, is given by

$$COORD(L) = COORD(PARENT(L))$$

When a node powers on, it associates with a drum at level 1 and sets its coordinate to the coordinate of the drum with which it associates. This implies that all nodes in a fundamental cell have the same coordinate.

Suppose a drum D_i at level i has the coordinate $[D_k D_{k-1} \cdots D_i D_{i-1} \cdots D_1]$ and decides to decrease its level from being a drum at level i to a drum at level $i-1$. Suppose after doing so it associates with another drum at level i, B_i, which has the coordinate $[B_k B_{k-1} \cdots B_i B_{i-1} \cdots B_1]$. Then the drum that decreased its level will be a drum at level $i-1$ and its coordinate will be $[B_k B_{k-1} \cdots B_i D_{i-1} \cdots D_1]$.

On the other hand if a drum D_i at level i has the coordinate $[D_k D_{k-1} \cdots D_{i+1} D_i \cdots D_1]$ and decides to increase its level to a drum at level $i+1$ then it chooses a uniform random number D'_{i+1} and sets its coordinate to be $[D_k D_{k-1} \cdots D'_{i+1} D_i \cdots D_1]$. However, before changing its coordinates, the drum listens for beacons from other drums at level $i+1$ in order to check for collisions. If there is a collision then the node picks another uniform random number instead of D'_{i+1} and uses that instead.

The level n beacon scope is extended beyond the TTL D_n to the cell of level $n+1$ in order to improve on the efficiency of routing and self-organization. To this end, a beacon of level n is forwarded by a node unless the $n+1$ coordinate of the node and the beacon are different. This limits the scope of the beacon flood.

2.2.4 Discussion

The beaconing protocol requires a notion of cells and coordinates to determine the beacon scope; the membership algorithm in turn relies on beacons to compute the hierarchy and coordinate. To clarify the situation let us mention that the beacon scope is at least a certain number of hops, the beacon TTL. This will assure proper functioning of the algorithms even with a temporarily dysfunctional cell structure. The beacon scope is widened to the next upper cell for efficiency in routing and not for correctness of the protocol.

At start up, nodes have an empty coordinate. Thus, a node will not stop any beacon but will forward it. If no beacon is received for a timeout period, a node will increase its level to become a drum. The beacons of the first few drums will reach throughout the network. Once at least two of them have increased their levels to become a 2-drum will the 1-beacon scope be confined by the cell structure.

When two Safari ad hoc networks merge, i.e., when nodes of two Safari networks with tree depth n and $m \leq n$ overhear each other's beacons then these beacons are not stopped immediately and penetrate the other network. The level k beacons with $k \geq m$, in particular, are forwarded throughout both Safari networks for the simple reason that either the beacon or the forwarding node has only m coordinates and thus cannot differ in the $k+1$ coordinate. The smaller network quickly learns of the high level drum in the other network, and associates with it, updating its coordinates in the process. This corresponds to merging a smaller tree at the appropriate level into the larger tree. If the depths happen to be equal, one will increase its level to become the new root, as discussed previously.

2.3 Scalable routing protocol

We propose a hybrid routing protocol which utilizes the beacons disseminated by the drum protocol. Our routing protocol is composed of two parts: An inter-cell hybrid routing protocol which deliver packets from the source to the fundamental cell of the destination, and a purely on-demand intra-cellrouting protocol which deliver packets to the final destination after it has reached the fundamental cell. When a node has a packet to forward, it derives the destination's coordinate from the packet and checks to see if the destination is within the same fundamental cell as itself. If so, it will use the intra-cell routing algorithm to deliver the packet. Otherwise, it will keep on forwarding the packet to the next hop using inter-cell routing and attach the destination's coordinate into the header of the packet for the intermediate forwarding nodes.

We assume the existence of bidirectional wireless links, which is true for the commonly used 802.11 protocol in the MAC layer. With this assumption, the inter-cell routing can be supported by following the reverse path of the beacons emitted by the drums where the destination node belong to. As to the intra-cell routing, presumably it can be any state of the art on-demand routing protocol. c

When a source node S with coordinate $[S_n S_{n-1} \cdots S_1]$, has a packet to deliver to node D with coordinate $[D_n D_{n-1} \cdots D_1]$, S retrieves the coordinate of D using the DHT address lookup service described in Section 2.4. Assume 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 is invoked.

In Figure 2, source node S uses inter-cell routing, along the route labeled by a and b, to reach the fundamental cell of destination node D. Within the fundamental cell, intra-cell routing is used to deliver the packet to the destination, along the route labeled by c. The route marked with a is the reverse path of the beacon emitted by the level 2 drum which D is belong to, while the route marked with b is the reverse path of the beacon emitted by the level 1 drum which D is associate with.

2.3.1 Proactive inter-cell routing

The basic strategy of the inter-cell routing is to take the reverse path of the beacons that are heard previously. When a beacon is received, the node store the direct upstream node's identifier to its DART. And when the node has a data packet

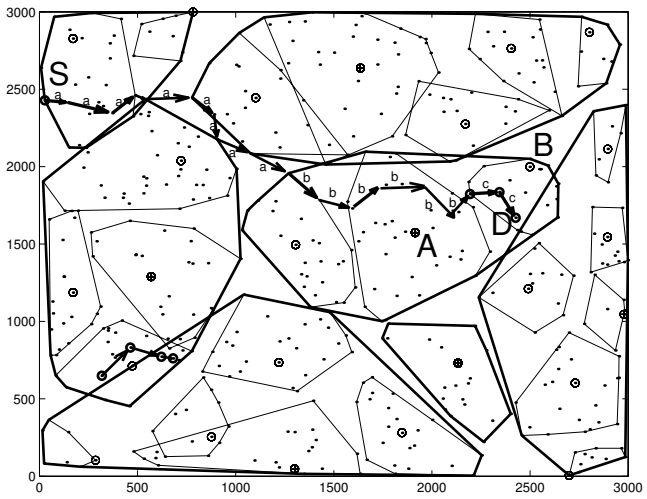

Figure 2: Safari routing overview

to send to the specific cell, it can then use the stored information to get the next hop.

As illustrated in Figure 1, each drum emits its beacon to all the nodes in its own cell as well as the nodes in the cells of its siblings, the same level drums which also share the same higher level drum. This mechanism guarantees that any node that is in the same supercell will have the routing information to all the fundamental cells in this supercell in the sense that it can reverse the beacon path.

Unlike some clustered routing protocols in ad hoc network, which assumes the super powered cluster header, drums in the safari hierarchical structure never need to be part of the path during data transmission. As we can see in the Figure 2, once the packet goes into the level 2 cell of A, any node will have the routing information to the fundamental cell B and the packet will not go though the drum A.

If the node needs to do the inter-cell routing, it searches its own DART to find the best routing information it has to the destination, which means, it needs to find the beacon entry with the longest prefix matched in the coordinate and the newest sequence number. Using this entry, the node then needs to compare it against the routing entry attached in the packet, if its entry is no worse than the one attached in the packet, it then replaces the field of matched prefix length and beacon sequence number in the packet with its own information. After that, the node will deliver the packet to the next hop indicated by the DART entry.

For inter-cell routing, each packet contains the following information in its header, till the packet reaches the fundamental cell of the destination:

- *Length*: The length of the prefix match with the entry in the DART that was used to forward this packet. the packet.
- *Sequence number*: The sequence number of the beacon corresponding to the entry in the DART that was used for forwarding this packet.

We also define *the prefix match length* between two coordinates $[B_nB_{n-1}\cdots B_1]$, and $[C_nC_{n-1}\cdots C_1]$ as the largest integer k such that $B_i = C_i; \forall n \leq i < k$.

In order to forward a data packet for a destination, a node does the following:

1. If the node finds an entry in the DART which has prefix match length $>$ *Length* contained in the packet, then the node updates the values of *Length* and *Sequence number*. The packet is then forwarded to the hop from which the beacon corresponding to the entry in the DART was received.
2. If the maximum prefix length (over all entries in the DART) that the node can find is equal to the *Length* contained in the packet, then if any such entry has a corresponding beacon sequence number which is greater than *Sequence number* carried in the packet, then use that DART entry to forward the packet after updating the *Sequence number* in the packet.
3. If the maximum prefix length (over all entries in the DART) is less than the *Length* contained in the packet, then invoke *local route repair* as described below.

Inter-cell routing is loop free, because for any given time, every node will have only one unique path to the destination. The DART entries give the reverse paths of the beacons, which are always trees and hence loop-free. In any tree, when a packet is travelling upwards, it always travels towards the root of the tree, which should be loop free. And once the packet travels downwards along a tree, it means the prefix match length increases. Since the forwarding algorithm requires that the node traversal can only be from shorter prefix length to longer prefix length, that means the node won't follow the same tree which it used to come upwards with. Therefore, the traverse path is loop free.

2.3.2 Local route repair in reactive inter-cell routing

The inter-cell routing scheme as described above requires the node forwarding the packet to follow the reverse path of the drum beacons. But this might not be possible in the following circumstances:

- When a node received a packet which follows the reverse path of some specific drum's beacon, the corresponding beacon entry at the node may have already expired. Also, due to partitions in the network or due to the unreliable wireless media, some beacons might not reach their intended scope. As a result of these, a node attempting to forward a packet might not have any useful information in its DART.
- When a forwarding node tries to send the packet to the next hop, the transmission could fail due to the mobility of nodes or the flaky conditions of the wireless medium,. This failure can be notified by its own MAC layer if

the MAC protocols uses like request and acknowledge mechanism, similar to the 802.11 MAC protocol. Unlike the previous cases, this situation is very frequent in wireless networks and can potentially degrade routing performance substantially.

If such situations arise, *Local Route Repair* is invoked by the forwarding node to find an alternate route. The node sends out a ROUTE REQUEST packet containing the following additional information:
- *the current maximum prefix length match in the node's DART*,
- *the sequence number for the beacon being followed*, and
- *the coordinate of the unreachable final destination*.
If a node receiving a ROUTE REQUEST, cannot reply to the request and if the request is not a duplicate of an earlier received request, the node rebroadcasts the ROUTE REQUEST. If the node receiving the ROUTE REQUEST searches its DART for the coordinate of the destination. If the nodes finds a longer prefix match for the destination coordinate or if the node finds a same length prefix match but with a higher sequence number, the node send a ROUTE REPLY containing the matching DART entry, back to the originator of the ROUTE REQUEST. A node receiving multiple ROUTE REPLY packets chooses the ROUTE REPLY with the longest prefix match. If two replies have the same length prefix match, the node accepts the reply from the closer responder.

2.3.3 Reactive intra-cell routing

When an intermediate node receives a packet, it checks if it itself is in the same fundamental cell as the destination, i.e. its coordinate matches with the coordinate of the destination. If the coordinates match, then the packet has reached the fundamental cell of the destination and intra-cell routing is used to send the packet to the destination. While any ad hoc network routing protocol can be used within the fundamental cell, we choose to use DSR [12] as it is a fully reactive protocol which has been shown to perform well [3]. DSR is a source routing protocol, i.e. each packet sent using DSR contains a source route. The DSR protocol consists of two mechanisms: Route Discovery and Route Maintenance. To perform a Route Discovery for a destination node D, a source node S broadcasts a ROUTE REQUEST that gets flooded through the network in controlled manner. This request is answered by a ROUTE REPLY from either D or some other node that knows a route to D. To reduce frequency and propagation of ROUTE REQUESTs each node aggressively caches source routes that the node learns or overhears. Route Maintenance detects when some link over which a data packet is being transmitted breaks. When such a route breakage is detected, a ROUTE ERROR is sent to S. Upon receiving a ROUTE ERROR, S can use any other route to D that it has in its route cache, or S can initiate a new Route Discovery for D.

In traditional DSR, a ROUTE REQUEST may be flooded throughout the entire network, thus making Route Discovery increasingly expensive with the increase in network size. Since in our case of intra-cell routing, it is already known that the destination exists in the same fundamental cell, the Route Discovery is limited to the diameter of the fundamental cell. As different fundamental cells may have different sizes, the Route Discovery range is based on the dynamic membership of the nodes, instead of predefined hop count. Specifically, whenever a node received a Route Discovery message, it compares it's own coordinate with the initiator's coordinate. The node then forwards the packet only if the two coordinates match. This technique is scalable, as the fundamental cell size does not increase with the network size.

The originator A may have the wrong coordinate of the destination B, as B might have changed it's membership recently. To take advantage of the high probability of the destination still remaining in the vicinity of it's previous fundamental cell, a hop count threshold is introduced in the Route Discovery phase which allow the Route Request packets to go a few hops beyond the fundamental cell. In this modified Route Discovery, each node forwarding a ROUTE REQUEST checks if its own coordinate is different from that of the originator of the ROUTE REQUEST and if so, increments the hop count field in the packet. If the hop count is lower than a threshold, then the message is forwarded, else dropped. This creates a fuzzy boundary of a cell allowing routing with lesser overhead.

2.4 Address resolution

The routing protocol described in Section 2.3 delivers packets to the intended destination given that destination's current coordinate, which depends on its current location in the network. However, typically senders wish to address packets using the destination's permanent identifier. To enable this, a distributed address resolution service maps a node's identifier to its current coordinate.

Clearly, an appropriate service must be scalable, efficient, and tolerant of node failures and mobility. In addition, to satisfying the needs of ad hoc wireless environments, it must observe locality, such that a lookup of a nearby node does not require communication with a distant node.

The service is implemented in a manner that is similar to a distributed hash table (DHT) but is adapted to the needs of mobile ad hoc environment. A conventional distributed hash table stores $\langle key, value \rangle$ pairs by hashing the key and storing the associated tuple at a set of overlay nodes whose unique identifiers are closest to the resulting hash value.

However, unlike an overlay-based DHT, our service directly exploits the hierarchical structure provided by the buoy protocol. Therefore it does not incur additional maintenance overhead. Also, the space of node coordinates, as opposed to the space of node identifiers as in overlay-based DHTs, is interpreted as the DHT's identifier space. Finally, our service inserts replicas of each tuple in a locality-aware fashion, to satisfy the needs of an ad hoc network environment.

Each node inserts the tuple (IP address, coordinate) at a set of k nodes throughout the network where k is the replication factor. The set k is determined by hashing the node's identifier k times, and choosing the nodes whose present coordinates are "closest" to each of the hash values. The node reinserts the tuple whenever its coordinate changes, or when a given resfresh interval has passed.

Any node can lookup the coordinate of another node by hashing the node's identifier and sending a query towards one of the resulting coordinates. With appropriate choices for k and the refresh interval, this scheme provides reliable address resolution despite the movement and failure of nodes. This basic scheme is similar to the one used in landmark routing [40].

Unfortunately, the basic scheme does not satisfy the locality requirements of an ad hoc network, since looking up a node requires communication with a node that may be distant in the network. Similarly, updating a node's coordinate requires communication with k nodes with arbitrary locations in the entire network.

To preserve locality, we augment the basic scheme as follows. In addition to inserting tuples at the k random coordinates, we insert the tuple also at k nodes closest to the node's present coordinate, with the i least significant coordinate components replaced by the random hash value, for i ranging from 1 to the number of levels in the buoy cell hierarchy minus 1. Thus, we store tuples at nodes that are increasingly closer to the node's present position in the network.

Moreover, when a node A's coordinate changes only in the i least significant components, it needs to reinsert its tuple only at nodes whose distance is proportional to i, thus reducing the cost of mobility. Thus, remote communication is only required when a node moves across a high-level cell boundary. Coordinates stored at greater distance may be inaccurate in the least significant components as a result of this technique, but the correct coordinate can be determined by iteratively checking the closer tuples during a lookup.

When node A looks up node B's coordinates, it proceeds by iteratively querying nodes at greater distance from A, using the same technique of coordinate prefix substitution. The maximal distance that a query has to travel is now related to the present distance between node A and node B in the network, thus reducing the cost of lookups.

In addition, each node in the system caches the coordinate of a node it has recently looked up or otherwise learnt. This caching reduces the load on the lookup service and reduces the total packet transmission latency in the common case. We will present an experimental evaluation of the lookup service in Section 4.2.

2.4.1 Fault tolerance

It is likely that one of the nodes storing a coordinate fails or becomes unreachable. This is counteracted by replicating the tuple at different hash locations as described earlier. The use of a good hash function distributes these replicas widely through the network, thus reducing the likelihood of tuples becoming unavailable. In addition, each node that stores a tuple also broadcasts that tuple for its neighbors to store. This allows a particular tuple to remain available even if the original node storing it has failed or departed.

Despite these measures, it is possible a node's coordinate becomes unavailable due to a series of node failures or movements. To minimize the probability of such a loss, each node periodically refreshes its coordinate. The refresh period should be more frequent for tuples stored close to the node and less frequent for distant tuples and is chosen dynamically according to the observed characteristics and stability (node mobility and churn) of the network.

3 Models and analysis

This section proposes and validates the models that we have used to characterize the drum protocol and the control overhead in the Safari architecture.

3.1 RSA model for drum formation

The Random Sequential Adsorption (RSA) [36] model which is encountered in the description of molecular adsorption processes lets us estimate the number of drums that form, by the time the drum protocol has converged.

A one dimensional version of the RSA model was studied by Renyi [28] and is popularly called the *car parking problem*. Consider a street of length L. Cars of unit length arrive at random points in the street and attempt to park at that point. A car parks if the already parked cars allow enough room at that location for it to park. It leaves otherwise. This process continues until no empty spot of at least unit length remains. The parking process is said to have reached the *jamming limit*. The fraction of street space occupied by the cars is called the *packing density*. Renyi mathematically showed that the average packing density at the jamming limit is 74.75% [28]. Only experimental results are available for higher dimensional cases (e.g. Hyperspheres parking in hypervolumes). The average packing density of the RSA process of circles parking on a plane (2-D RSA) is 54.7% [7]. A limit theorem of Coffman et al. [6] ensures that the average value of the packing density converges when the parking hypervolume becomes large. In the following sections we consider ad hoc networks of nodes in two dimensional space and use the 2-D RSA model of discs.

3.1.1 The instantaneous propagation approximation

Consider two nodes A and B, within D_1 hops of each other, which just powered ON and are waiting to hear from level 1 drums. The nodes wait for a random time, uniformly distributed between $[T_{min}, T_{max}]$, before turning into level 1

drums if necessary. Let T be the time for a packet to traverse D_1 hops. If $T_{max} - T_{min} \gg T$ then, with high probability, one of the nodes will hear a beacon from the other node before deciding to become a drum and so will not become a drum. Therefore with high probability, level 1 drums are separated by at least D_1 hops. The instantaneous propagation assumption states that level 1 drums are separated by at least D_1 hops, with high probability. Extending the argument to all levels, level i drums are separated by at least D_i hops, with high probability.

3.1.2 Drum formation conforms to the RSA model

Under the instantaneous propagation approximation, all level 1 drums are at a distance of at least D_1 hops. Thus, fictitious discs of radii $D_1/2$ hops centered at the drums do not overlap. This can be thought of as nodes attempting to *park*. A node attempts to park a disc of radius $D_1/2$ hops when its waiting timer expires. A parking attempt succeeds if the disc does not collide with an existing disc and fails otherwise. A failure is equivalent to a node finding a level 1 drum within D_1 hops before its waiting timer expires and hence deciding not to become a drum. The drum formation process will proceed as long as a disc can park without colliding and when it completes no more discs can be placed. This is because if it were indeed possible to place a disc at a node without conflicts then that node is at least a distance of D_1 hops from every level 1 drum and hence will have a tendency to become a drum itself.

If n is the average number of level 1 drums that form, then using the fact that the parking density at the jamming limit is 54.7% we find

$$n = 0.547 \left(\frac{N}{\pi \frac{D_1^2}{4} \rho} \right) \tag{2}$$

where N is the number of nodes in the network. ρ is the node density in the hop metric sense, its value depending on the transmission range of the radio signal and the spatial density of nodes. From simulations, we estimated value of ρ to be 1.4 for node density of 10 per radio range and radio range of 250 meters. The above equation expresses that the fictitious discs around the drums cover 54.7% of the nodes.

The plot in Figure 3 was obtained by simulating the drum protocol in ns-2 for 100 seconds with 500 nodes. The mobility model is a modified billiards ball model with random reflection angle upon hitting simulation surface edges. Node speeds are uniformly distributed between 5 and 15m/s. The plot shows the number of level 1 drums per node (n/N) at 100s, at which time the drum formation process is complete. The waiter timer expires uniformly in [0,150s]. Hence the instantaneous approximation holds. The simulation shows that the RSA model successfully predicts the number of drums that form after convergence of the drum protocol.

For higher level drums the parking problem becomes a constrained parking problem (a level 2 drum should also be a level 1 drum). The parking "substrate" (lower level drums) does not look like a continuum to the higher level drum formation process. The appropriate model is the RSA-RS (RSA-Random Sites) [41]. The model effectively reduces the packing density a little. From Equation 2 the following equation holds:

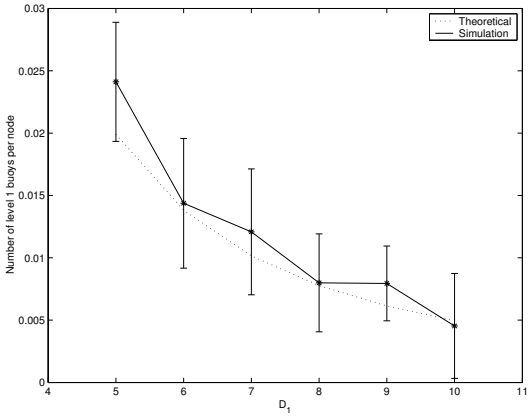

Figure 3: Number of level 1 drums: *ns-2* simulation

$$\frac{n_i}{n_{i+1}} = \alpha_i (\frac{D_{i+1}}{D_i})^2 \tag{3}$$

where n_i is the number of level i drums and α_i is the ratio of the average packing densities of the RSA-RS process associated with level i drums and the level $i+1$ drums. Since the packing density decreases with the level slightly, α_i is larger than but close to unity.

3.2 Collision model for drum retirement

When two level n drums drift to within $h * D_n/2$ hops of each other, one of them will retire. We can think of this as a collision of two fictitious discs of radii $h * D_n/2$ hops. We can use results from statistical physics of molecular collisions to compute the frequency of retirements [33]. Adapting the theory for two dimensional discs we obtain the number of level n drum retirements per second in the network to be

$$\frac{AD_n v_{average} \rho^2}{\sqrt{2}} \tag{4}$$

where $v_{average}$ is the average velocity of the nodes, A is the area covered by the network and ρ is the spatial density of level n drums. Using the RSA model results to substitute for ρ, the above expression simplifies in functional form to (N^2/AD_n^3) which is $\Theta(1/D_n^3)$ level n drum retirements per second per unit area (see Equation 8). Thus the stability of the drums increases with the level of the drums.

Drum formation rate at equilibrium is the same as Equation 4 if we assume a fairly stable population of drums when equilibrium has been reached.

3.3 Overhead characterization

This section characterizes some of the control overhead by using the models previously described.

3.3.1 Drum flooding overhead

Equation 3 says that for every level $i+1$ drum there are $\alpha_i(D_{i+1}/D_i)^2$ level i drums where α_i is a proportionality factor which is close to unity. Consider a network with L levels (L is $O(logN)$) of drum hierarchy. Let the level i drum emit drum packets once every T_i seconds. A level i drum broadcast reaches all the nodes in the next higher level (level $i+1$) cell. So a node will receive drum packets from all level i drums which are within the same level $i+1$ cell which contains this node ($\alpha_i(D_{i+1}/D_i)^2$ of them) for i running from 1 to $L-1$ and from the highest level drum.

Therefore, overhead due to drum floods X_{drum}, defined by the number of packet forwarded per second per node is:

$$X_{drum} \approx \alpha_1 \frac{D_2^2}{D_1^2}\frac{1}{T_1} + \alpha_2 \frac{D_3^2}{D_2^2}\frac{1}{T_2} + ... + \alpha_{L-1}\frac{D_L^2}{D_{L-1}^2}\frac{1}{T_{L-1}} + \frac{1}{T_L} \quad (5)$$

The actual overhead due to drum floods is a little greater than that shown in the above equation because beacon floods propagate a minimum number of hops (D_i). Therefore a level i drum's packets reach some nodes of a neighboring level $i+1$ cell (like A in Figure 1) if the drum is at the border of the level $i+1$ cell. But the error in Equation 5 is negligible. This is because such overhearing nodes live at the border of the cell and hence are far fewer than interior nodes (like B in Figure 1) of a cell. Moreover such a node will hear a "leaked packet" only once for every, order of, $(D_{i+1}/D_i)^2$ broadcasts from its own level $i+1$ cell. D_i's and T_i's are geometrically increasing as shown by Equation 1. Hence, Equation 5 implies that $X_{drum} = \Theta(1)$.

3.3.2 Frequency of cell changes for a node

A node's coordinate could change if the node moves into a new cell. Since the average interspacing (in hops) between two level $i+1$ drums is proportional to D_{i+1}, if the average velocity of the level i drum node is v, it will cross the boundary of its level $i+1$ cell in a time proportional to D_{i+1}/v. In particular the frequency with which a node moves over to an adjacent fundamental cell is proportional to v/D_1.

Figure 4 shows the result of an *ns-2* simulation. The simulation is run for 100 seconds. The data during the first 25 seconds was discarded to remove initial transient behavior. The x-axis is the inverse of D_1 and the y-axis is the total number of coordinate changes for all 500 nodes due to level 1 cell crossings of the nodes. It is seen that the number of coordinate changes due to level 1 cell crossings follows a $1/D_1$ law.

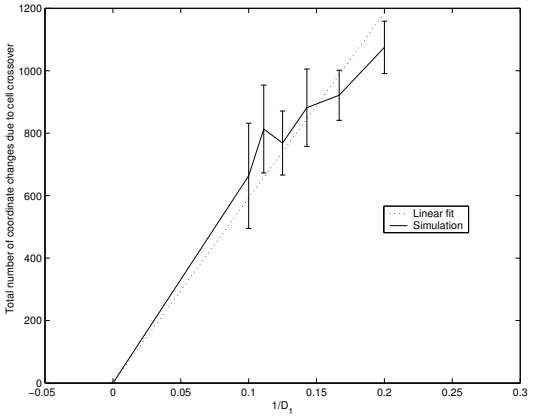

Figure 4: Level 1 cell crossing frequency follows $\frac{1}{D_1}$ law

3.3.3 DHT control overhead characterization

Nodes update the DHT on-demand when their coordinates change. Below we analyze the overhead in the three circumstances that can cause a node to change its coordinate:

Case 1: A level i drum crosses the boundary of its level $i+1$ cell, in which case the coordinates of all the nodes which are associated with this drum change in the $(i+1)$th digit from the last. In particular, if a non-drum node (a level 0 drum) moves into a new fundamental cell within the same level 2 cell, its coordinates last digit will change.

As was concluded in the previous section, a node's coordinates ith digit from the last, changes with a frequency of $\beta(1/D_i)$ times per second for some constant β which is independent of the D_i's.

If a node's coordinate changes beginning at the ith digit from the last, it updates its coordinate at all its *locator nodes* (nodes that store its coordinate) which are in the same level $i+1$ cell as this node is (locality of updates condition). Those are the m locator nodes whose coordinates differ from this node's coordinate in the last digit only, along with the m locators whose coordinates differ from this node's coordinate in the last two digits only and so on until level i (for simplicity we assume $m = 1$ in the remainder of this section). However, the average hop distance of a locator node whose coordinate is obtained by hashing and replacing all but the last j digits with the coordinate digits of the node in question is proportional to D_{j+1}, because such a node could be anywhere in the level $j+1$ cell whose diameter is of the order of D_{j+1} hops. Therefore about D_{j+1} nodes are involved in forwarding the DHT update to that locator node. Therefore the DHT update overhead measured in packets/node/second generated due to nodes changing their i th level drum is proportional to

$$(D_2 + .. + D_{i+1})\frac{1}{D_i} = \frac{1}{D_i}\sum_{j=2}^{i+1} D_j \quad (6)$$

The total overhead is obtained by summing up the above

for all levels. Hence if we denote by X_{DHT1} the overhead per node per second through DHT updates triggered by mechanism I, then

$$X_{DHT1} = \sum_{i=1}^{L-1} \frac{\alpha_i}{D_i} \sum_{j=2}^{i+1} D_j \quad (7)$$

where α_i's are constants independent of the D_i's. If we choose D_i's to be geometrically increasing then the above sum is $0(L)$ where $L = 0(log(N))$.

Case 2: If a drum retires, then nodes in the vicinity will change their coordinate.

Consider two drums of levels i and j. Assume $m = min(i, j)$. If these two drums come within $D_m/2$ hops of each other one of them will retire. This will lead to some nodes changing their coordinates and updating them in the DHT. More specifically the coordinates will change in their mth digit from the last digit because a level m drum has retired.

The number of such nodes changing their coordinates is proportional to the cell size of the level m cell which is in turn proportional to D_m^2, as inferred from Equation 2.

These nodes will update the DHT by updating their locator nodes and will therefore give rise to an update overhead. The frequency with which retirement of level m drums happens is described by the collision model. From Equation 4, the total number of collisions per unit area per second between level m drums is proportional to $D_m \rho_m^2$, where ρ_m is the spatial density of level m drums. By a collision, we mean two drums drifting so close as to lead to retirement of one of them. But $\rho_m = n_m/A$ where A is the area of the network and n_m is the total number of level m drums in the network. But from the RSA model described previously, n_m is proportional to N/D_m^2 where N is the total number of nodes in the network. Therefore, the total number of collisions per second in the network between level m drums is proportional to

$$AD_m \rho_m^2 = AD_m \frac{n_m^2}{A^2} \propto \frac{D_m}{A} \frac{N^2}{D_m^4} = \frac{N^2}{AD_m^3} \quad (8)$$

But each collision is going to lead to order of D_m^2 nodes changing their coordinates. But these coordinate changes are occurring in the mth digit from the last and hence they incur a forwarding overhead proportional to $(D_2 + ... + D_{m+1})$. Therefore the per second *node packet product* associated with the DHT update overhead due to level m drum retirements is proportional to

$$\frac{N^2}{AD_m^3} D_m^2 (D_2 + ... + D_{m+1}) \quad (9)$$

Hence the per node packet overhead per second is $1/N$ th of the above and simplifies to

$$\alpha_m \frac{N}{AD_m} (D_2 + ... + D_{m+1}) \quad (10)$$

where α_m is a constant of proportionality which is independent of the D_i's. We sum up the above expression over the drum levels to obtain the total overhead. If we denote by X_{DHT2} the overhead due to mechanism II in packets per node per second, then

$$X_{DHT2} = \sum_{m=1}^{L-1} \alpha_m \rho \frac{D_2 + ... + D_{m+1}}{D_m} = \sum_{i=1}^{L-1} \frac{\beta_i}{D_i} \sum_{j=2}^{i+1} D_j \quad (11)$$

where ρ is the spatial density of nodes measured in nodes per square meter and α_m's are proportionality constants independent of the D_i's. It is remarkable that X_{DHT2} has the same form as X_{DHT1} and hence is $0(log(N))$. That is, the overhead incurred by drum retirements and cell crossovers are of the same functional form.

Case 3: If a new drum forms, then nodes in the vicinity will change their coordinate.

Under equilibrium conditions the drum population at any level i fluctuates about a mean value. Thus roughly speaking for every level i drum retiring a new level i drum appears. Thus the retirement rate is equal to the formation rate at equilibrium. But for every level i drum appearing the number of nodes that will experience a change in the coordinate in the ith digit is proportional to D_i^2, the order of the size of a level i cell. These nodes will update the locator nodes. Therefore by arguing exactly as we argued for the previous case, we arrive at the same expression for the per node per second packet overhead as for X_{DHT2}. Therefore the overhead due to mechanism 3, X_{DHT3} is also $0(log(N))$.

In addition to above three mechanisms by which nodes update the DHT on-demand (only upon a coordinate change) they also periodically reinsert their co-ordinates in the DHT. To quantify the overhead due to this mechanism we assume that a node reinserts its coordinate at all locator nodes whose coordinates are obtained by replacing the random hash of its identifier with the node's own coordinate except at the last i digits, once every τ_i seconds. These locator node's coordinates differ from the coordinate of the inserting node in the last i digits and are therefore at a distance proportional to D_{j+1} hops. These updates should be forwarded by the nodes in the network. The per node per second packet overhead due to this is

$$\sum_{i=1}^{L} (1/\tau_i) D_{i+1} \quad (12)$$

where we define $D_{L+1} = D_L$ for the above equation. If τ_i were chosen proportional to D_i then the above summation is $0(log(N))$

3.3.4 DHT storage overhead

Assume a safari network of L levels. Each node inserts its coordinate m times for each level (locality condition). Therefore the total network storage requirement is $NmL <$ $key, value >$ pairs. Since every node is equivalent to others in terms of the the amount of storage it contributes to the DHT, the per node storage overhead is $mL = 0(mlog(N))$.

4 Simulation evaluation

In this section we present the evaluation of our architecture. We evaluate the scalability of the Safari architecture with increasing network size as well as the performance of the architecture under varying node mobility and offered traffic load. In order to get a clearer picture of how Safari performs, we evaluate the routing performance separately from the performance of the distributed hash table based address resolution service.

4.1 Routing performance

We present the evaluation of the safari routing protocol using the *ns-2* network simulator. We used the default parameters in the *ns-2* for the wireless MAC and physical layers settings, thereby having a 802.11 based 2 Mbps data rate and a nominal transmission range of 250 m.

We set the beacon broadcasting limit to 3 wireless hops (i.e. $D_n = 3$), thus allowing the fundamental cell to be up to 6 wireless hops in diameter. We made this decision based on the fact that DSR is shown to perform well, without significant overhead, up to around 5-6 wireless hops. The lowest level drum broadcast its beacon packet every 2 seconds, i.e $T_1 = 1$ second in Equation 1. The value of α in Equation 1 was chosen to be 2 so that the are encompassed by a cell at a particular level contains $2^2 = 4$ cells of the next lower level. The value of β in Equation 1 is also chosen to be 2.

We did our simulations in various network topologies with size ranging from 50 nodes to 1000 nodes randomly distributed in a two dimensional topology. However, the X and Y dimensions of the network were modified so that the network density was kept at a constant 50 nodes in a 1000 m$\times$1000 m 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 huge memory consumption of *ns-2* we had to limit out simulations to a maximum of 1000 network nodes. Nevertheless, as our results show, we were successful in demonstrating the scalability of Safari.

We used the random waypoint model to simulate the mobility of the network nodes. Mobile nodes have a maximum speed of 10 m/sec, and an average speed of 5 m/sec, and zero pause time. The objective of Safari is to provide a city-wide ad hoc network environment. Thus it is highly unlikely that all nodes in the network will be mobile simultaneously. Nevertheless, we studied the performance of Safari in the worst case scenario of all nodes being mobile. The knowledge of whether a node is mobile or static is totally transparent to the Safari architecture, which implies that Safari does not take any advantage of static nodes. For example, if the static nodes are known, choosing them as drums will provide much better performance due to lesser number of coordinate changes.

We used constant bit rate(CBR) flows to generate the offered traffic load on the network. Each flow consists of a randomly chosen pair of source and destination nodes. Each flow lasts 90 seconds and generates 64 bytes packets at a constant rate of 4 packets per second. We allow 350 secs of simulation time for the network to stabilize and after that the flows start arriving according to a Poisson distribution. Each simulation runs for 900 sec. This traffic patterns is more challenging than the typical continuous long life-time 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.

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

4.1.1 Scalability

The most important goal of Safari is to scale to a large-scale ad hoc network. We show that we meet 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 the number of control (non-data) packets transmitted by the protocol. Each overhead packet whether it is generated or forwarded contributes to this overhead. For example, a single beacon packet that gets forwarded by two nodes will count 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.

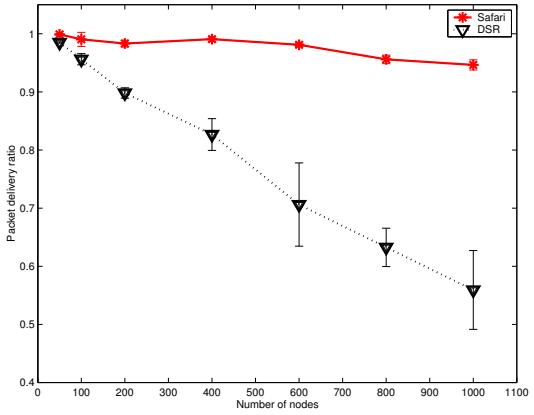

Figure 5: Packet Delivery Ratio vs. network size

Figure 5 shows the comparative performance of a popular ad hoc routing protocol, DSR, with Safari. Both the routing protocols perform equally well until network size of around 100 nodes. After that the PDR of DSR drops sharply, going down to about 60% for a network consisting of 1000 nodes. Safari routing on the other hand performs very well, with the PDR slowly falling to about 95% for 1000 nodes. This shows that Safari can scale up to much larger numbers.

Figure 6 shows the packet overhead versus network size. The total overhead is dominated by the beacon overhead,

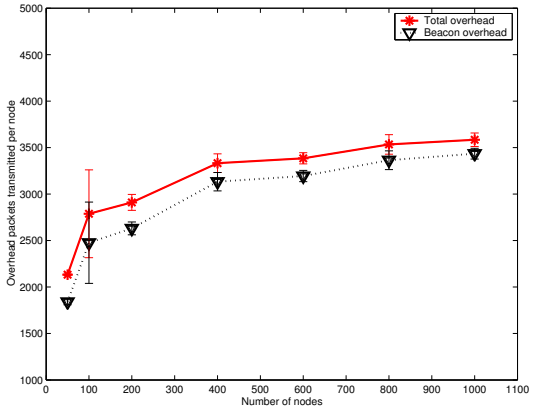

Figure 6: Overhead packets per node vs. network size

with Route Request, Route Reply, and Route Error for both DSR and inter-cell routing accounting for the rest. The beacon overhead is characterized in Equation 5, which shows that the overhead asymptotically approaches a constant value. This characteristic is validated in our simulations, as shown in Figure 6. Thus, the curve for beacon overhead closely follows the curve for total overhead. In other words, with increasing network size, the overhead per node is bounded, thus allowing our architecture to scale essentially to arbitrary size.

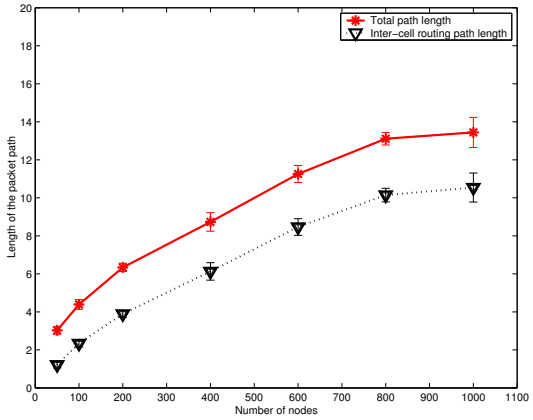

Figure 7: Number of hops vs. network size

Figure 7 shows the average number of hops taken for a random source to a random destination, with the increase of network size. The line on top 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, showing the number of hops within a fundamental cell is almost constant, showing that the size of the fundamental cell is fixed regardless 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.

4.1.2 Mobility

In this section, we show that Safari architecture is able to efficiently handle various degrees of mobility. We fix the number of nodes in the network at 1000 and the number of CBR flows at 100 and then , vary the mobility from 0% nodes being mobile to to 100% nodes being mobile.

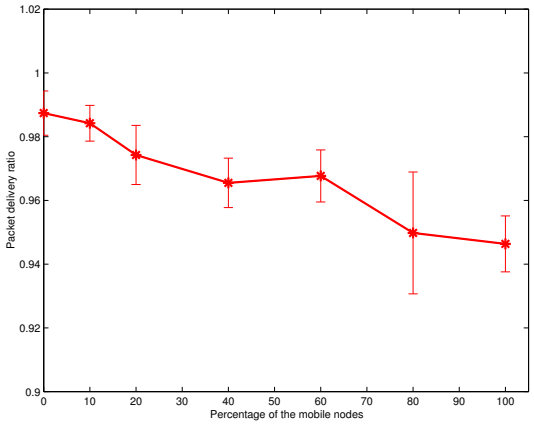

Figure 8: Packet Delivery Ratio vs. % mobile nodes

Figure 8 shows the change of PDR with the number of mobile nodes varying from 0% to 100%t. The PDR is over 95% for even 100% mobile nodes, showing that the protocol reacts very well to mobility.

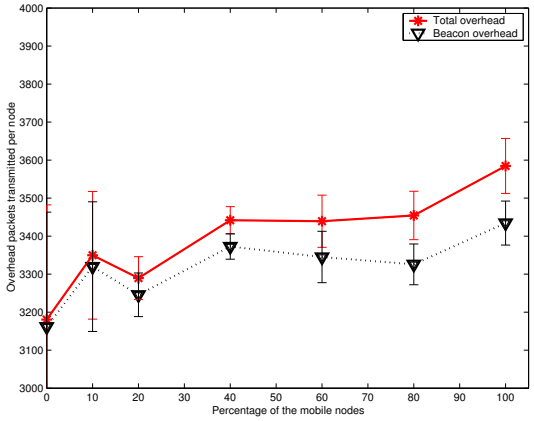

Figure 9: Overhead packets per node vs. % mobile nodes

Figure 9 shows the increase of overhead with mobility. The beacon overhead is almost constant with very little increase, as it does not vary with mobility. The total overhead is almost same as beacon overhead in a static network due to the lack of any reactive overhead. As the percentage of mobile nodes increase, the reactive component increases which makes the total overhead diverge from the beacon overhead.

4.1.3 Traffic load

Next, we show the Safari routing performance with varying traffic load. The number of nodes is constant at 1000, and all the nodes are mobile.

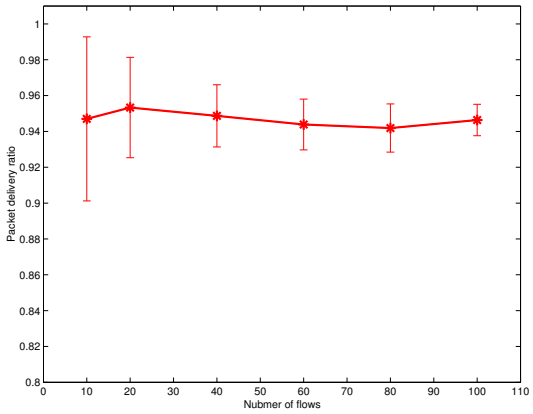

Figure 10: Packet Delivery Ratio vs. traffic load

Figure 10 shows that the PDR changes little with the increase of load. The error bars are smaller for larger number of flows since the total number of packets being sent is more for larger number of flows, thus smoothing statistical inaccuracies in the results.

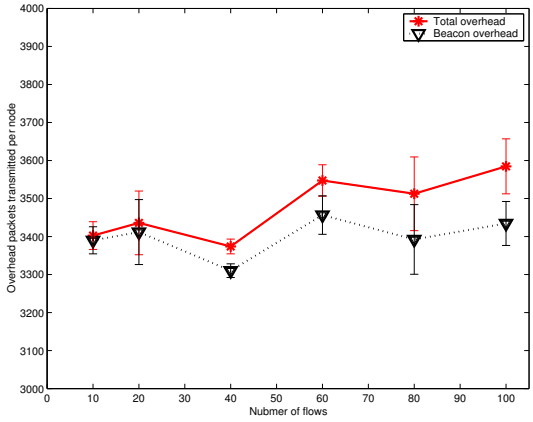

Figure 11: Overhead packets per node vs. traffic load

The overhead increases slightly with the increase of traffic load from 10 to 100 CBR flows, as is seen in Figure 11. All the increase is from higher reactive component, because of the increased number of flows. Th beacon overhead remains almost constant, because it does not vary with the number of flows.

4.2 DHT performance

In this section, we present a scalability evaluation of the DHT component of the system. We use a high level event based simulator written specifically for this evaluation. This simulator simulates the cell formation and inter-cell routing but does not implement the DSR based intra-cell routing. The parameters chosen are the same as those described in Section 4.1 unless otherwise stated. We use the values established from the *ns-2* experiments in the previous section (Figure 7) to place 10 nodes per fundamental cell and to make the cost of traveling across a cell 2.9 hops.

4.2.1 Evaluation of DHT scalability

The DHT component of the system must scale effectively in order to allow the system as a whole to scale. We evaluate the scalability of these operations in the steady state as they do not depend on mobility, i.e. the cost does not change with the mobility of nodes. The following experiments we show that all quantities scale as expected and grow logarithmically.

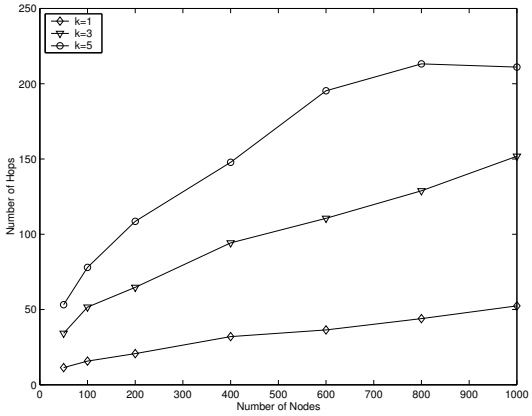

Figure 12: Average network hops to store a coordinate

In Figure 12 we show the average number of hops for all packets required to store a coordinate in the DHT. Node mobility will lead to an increase in the number of coordinate changes and consequent updates to the DHT but the cost for a single update does not depend on the mobility of nodes. Each line shows the value for a different replication factor. Expectedly, storing more replicas increases the cost of storing a coordinate. In this experiment we allow all nodes in the system to store their coordinate and then calculate the average number of hops needed, this value is also the expected value for a new node entering the system.

Figure 13 shows the number of hops required to satisfy a coordinate query to the DHT. In this experiment each node chooses a random node in the network to lookup. We again vary the replication factor, in this case more replicas help us, as the likelihood of one of the replicas being close to the node doing the lookup increases. These values agree with the expected cost to route to a random node in the network. This cost is relatively small and only occurs when you begin a connection or the destination node moves.

Figure 14 shows how the number of $\langle key, value \rangle$ pairs stored at each node varies with the number of nodes for various values of the replication factor. As shown previously this value is about kL where k is the replication factor and L

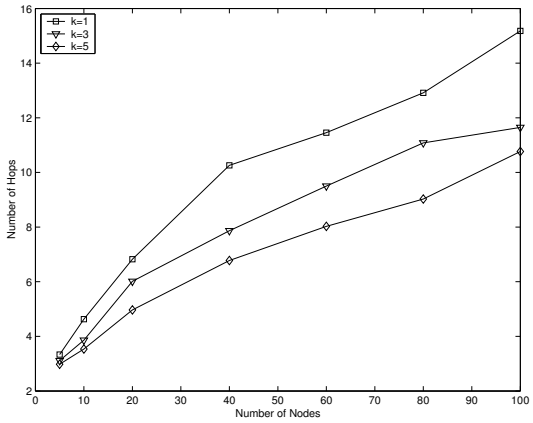

Figure 13: Average network hops to lookup a coordinate

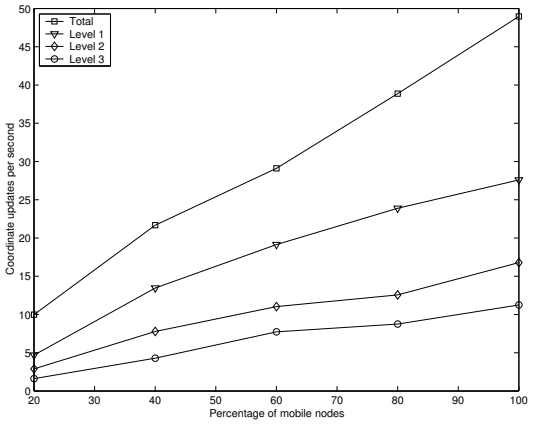

Figure 15: Number of updates

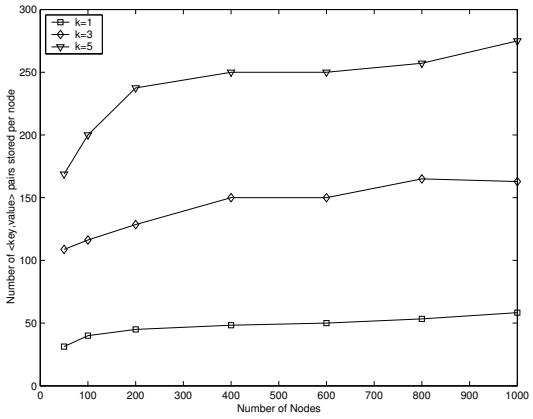

Figure 14: Per Node storage requirements

is the number of levels of hierarchy. The load of storage is distributed through the network and the requirement is quite modest as each record is small.

Figure 15 shows the number of updates per second required when nodes are mobile. In this scenario there are 500 nodes and the k is 3. We use random way point model with no pause time and a maximum velocity of 10 m/s. Each node moves and reinserts its coordinate as appropriate. The cost to reinsert is proportional to the distance traveled, as the graph shows level 1 stores make up the majority of the traffic meaning that only a node in the same super cell must be contacted to update the coordinate.

5 Related work

In this section, we describe the related work in scalable ad hoc networks and how Safari is different from existing approaches.

Ad hoc network routing protocols can generally be classified as either *proactive* (periodic) or *reactive* (on-demand). Proactive protocols (e.g., [25, 22]) try to maintain routes to all possible destinations at all times, whereas reactive protocols (e.g., [12, 26]) attempt to discover or maintain routes only when needed to destinations for current communication. Reactive routing protocols have been shown to have generally lower overhead than proactive protocols, and they can react much more quickly as routes in the network change. However, for very large networks or very high rates of mobility, the overhead of current reactive protocols can grow quickly.

A number of approaches to scalable ad hoc network routing have been proposed. Geographical routing techniques (e.g., [13, 14, 18, 32]) allow routing with state proportional only to the number of neighbors at each node, but they require GPS or other location techniques. Moreover, the sources need to know the location of destinations before sending messages thus requiring a location distribution and maintenance service. DREAM [32] and LAR [14] employ proactively or reactively flooding the network and hence do not scale with the size of the network. GLS [17] proposes a scalable location service for geographical routing in ad hoc networks. Even if the location service for geographic routing can be made scalable, yet GPS devices are expensive and more importantly, do not function indoors, thus making location based routing impractical for general purpose ad hoc networks. Even though Safari is not as scalable as geographic forwarding yet Safari provides a practical, self-organizing hierarchy and is not dependent on expensive GPS devices.

Clustering techniques (e.g., [1, 2, 11, 21, 19, 20, 34, 31, 42]) can increase scalability, but existing active clustering mechanisms require periodic refreshing of neighborhood information and introduce significant maintenance overhead due to global query flooding. Another class of protocols use the idea of routing via dedicated fixed anchors (e.g., [8, 15, 37]), but such techniques depend on deployment of fixed anchor nodes.

Techniques based on *landmark routing*, first proposed by Tsuchia [40, 38, 39], have also been proposed for scalable routing in ad hoc networks. Similar to our drum hierarchy in Safari, landmark nodes self-organize themselves into a hierarchy, such that landmarks at a given level in the hierarchy are

an approximately equal number of network hops apart. The address of a node consists of the sequence of identifiers of the nearest landmarks, from highest to lowest level. During routing, a node extracts from the destination address the highest level landmark identifier that differs from its own node address, and forwards the packet towards the landmark with that identifier. The mapping from node identifiers to their current address is maintained in a distributed fashion. Landmark routing achieves scalability by dramatically reducing the size of per-node routing tables at the expense of somewhat longer routes. The original landmark scheme, designed for large wired networks such as the internet, had only routers as landmarks. End nodes did not participate in the hierarchy. LANMAR [24] attempts to scale mobile ad hoc networks by combining ideas from landmark routing and from Fisheye State Routing [23]. It specifically targets ad hoc networks consisting of groups of nodes related in functionality and mobility.

The routing protocol that is most similar to Safari is L+ [4]. L+ modifies the lookup service of landmark routing to make it more scalable and modifies the routing to be more reactive in order to handle mobile nodes. It has the concept of of multi-level landmarks, with one highest level landmark. The landmarks exchange advertisement messages with the radius of propagation proportional to the level. The landmarks have the concepts of promotion and demotion of it's level depending on distance to other landmarks. The routing follows the Distance Vector (DV) table, constructed by exchanging these advertisement messages. To forward a packet, the DV table is searched for the longest prefix match with the destination of the packet. The location service, which converts a given identifier (say, IP address) to the hierarchical address is quite similar to our distributed DHT lookup service. The distance the lookup has to traverse is proportional to the distance between the source and destination. So, a local communication will not entail a global communication due to lookup.

Even though L+ has a number of similarities to Safari, the fundamental difference between the two lies in how the two systems perform routing. L+ uses a purely proactive approach to routing and is in fact based on DSDV. However, Safari employs a hybrid of proactive and reactive approaches. L+ keeps a list of routes to any destination and switches to the next route when the current best route breaks. It also needs a trigger DV update whenever there is any change in connectivity due to mobility or channel loss. As the proactive part of routing, Safari uses reverse beacon paths, avoiding all per-destination overhead. However, when a route breaks, we perform route maintenance, thereby switching to reactive routing to repair the routing state. In addition, unlike L+, the hierarchy in Safari does not extend down to the lowest level but stops at fundamental cells. Within each fundamental cell, Safari uses a purely reactive protocol, thus reducing overhead and improving scalability. In summary, L+ inherits all the problems associated with proactive ad hoc routing protocols, which Safari avoids. With increasing mobility, the frequency of the proactive updates in L+ (or any proactive protocol) must increase proportionally, significantly increasing its overhead, a problem avoided in Safari.

Moreover, the authors of L+ have done only preliminary evaluation of it. The evaluation uses a channel bandwidth of 100 Mbps, whereas the traffic load on the network is very low, with each node sending just 15 packets over a 5-second interval. Even though L+ is a proactive protocol, the control overhead of the protocol has not been evaluated.

Structured P2P networks like CAN [27], Chord [35], Pastry [29] and Tapestry [43] can scale to very large number of nodes and are highly resilient to node failures, same properties that we want in our Safari architecture. However, all of the existing structured overlay networks were designed for the Internet, with its mostly stationary hosts and its high-capacity backbone. Non-local communication is the norm in these overlays, nodes are assumed to be stationary, overlay routing is completely separate from network routing, and overlay maintenance overhead can be substantial. As a result, these systems are not directly applicable to ad hoc wireless networks, where cross-section bandwidth is scarce and the network topology may change rapidly [10, 16, 9]. Nevertheless, one of the abstractions one can implement upon a structured overlay is a distributed hash table (DHT). Like a conventional hash table, a DHT stores ⟨key,value⟩ tuples, but does so in a highly scalable, decentralized, and fault-tolerant manner. DHTs can be used, for instance, to provide decentralized storage and distributed lookup services [5, 30]. We use a similar notion of DHT to store and lookup hierarchical addresses of nodes, adapting it to preserve locality in ad hoc networks.

6 Conclusions

We have presented the design and evaluation of a novel protocol for scalable routing in ad hoc networks. We have developed a probabilistic, self-organizing network hierarchy formation protocol and complemented it with our hybrid routing protocol that uses this hierarchy. The hybrid protocol consists of both proactive as well as reactive components, helping it to scale. Nodes in the Safari architecture are given coordinates, and each node's unique node identifier is mapped to its coordinate using a distributed hash table that leverage the hierarchical network structure. We have evaluated our protocol, through analysis and simulations, under increasing network size, increasing fraction of mobile nodes, and increasing offered traffic load. Our evaluation demonstrates that the Safari architecture succeeds in being able to scale to large networks.

References

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

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

[3] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. 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, Oct. 1998.

[4] B. Chen and R. 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.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. ACM SOSP'01*, Banff, Canada, Oct. 2001.

[6] P. J. E. G. Coffman, L. Flatto and B. Poonen. Packing random intervals on-line. *Algorithmica*, 22(4):448–476, December 1998.

[7] J. F. E. L. Hinrichsen and T. Jossang. Geometry of random sequential adsorption. *Journal of Statistical Physics*, 44(5/6), 1986.

[8] s. Giordano and L. Boudec. Anchored path discovery in terminode routing. *Proc. of the 2nd IFIP-TC6 Networking conference (Networking 2002)*, May 2002.

[9] N. Gupta and S. R. Das. A capacity and utilization study of mobile ad hoc networks. In *Proceedings of the Wireless Local Networks (WLN) Workshop*, November 2001.

[10] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 40(2):388–404, March 2000.

[11] Z. J. Haas and M. 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, Aug. 1998.

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

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

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

[15] S. G. "L. Blazevic and J.-Y. L. Boudec". "self organized terminode routing". *"Cluster Computing"*, "April 2002".

[16] J. Li, C. Blake, D. S. J. D. Couto, H. I. Lee, and R. 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.

[17] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris. A scalable location service for geographical ad hoc routing. In *Proc. of ACM MOBICOM 2000*, Boston, MA, 2000.

[18] W.-H. Liao, Y.-C. Tseng, and J.-P. Sheu. GRID: A fully location-aware protocol for mobile ad hoc networks. *Telecommunication Systems*, 18(1-3):37–60, 2001.

[19] C. R. Lin and M. Gerla". "adaptive clustering for mobile wireless networks". *"IEEE Journal of Selected Areas in Communications"*, "15"("7"):"1265–1275", "1997".

[20] S. D. "M. Chatterjee 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".

[21] A. MCDonald and T. Znati". "a mobility-based framework for adaptive clustering in wireless ad hoc networks". *"IEEE Journal of selected areas in communications"*, "Aug., 1999".

[22] V. D. Park and M. S. 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, Apr. 1997.

[23] G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing in mobile ad hoc networks. In *ICDCS Workshop on Wireless Networks and Mobile Computing*, pages D71–D78, 2000.

[24] G. Pei, M. Gerlan, and X. Hong. Lanmar: Landmark routing for large scale wireless ad hoc networks with group mobility. In *SIGCOMM'88*, Stanford, CA, 1988.

[25] C. E. Perkins and P. 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, Aug. 1994.

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

[27] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, San Diego, CA, Aug. 2001.

[28] A. Renyi. On a one-dimensional problem concerning random space-filling. *Publ. Math. Inst. Hung. Acad. Sci.*, 3:109–127, 1958.

[29] 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, Nov. 2001.

[30] 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, Oct. 2001.

[31] I. C. S. Basagni and A. Farago. "a generalized clustering algorithm for peer-to-peer networks". *"Workshop on Algorithmic Aspects of Communication"*, "1997".

[32] V. S. S. Basagni, I. Chlamtac 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). ACM, Oct. 1998.

[33] W. R. Salzman. Kinetic molecular theory. http://www.chem.arizona.edu/ salzmanr/480a/480ants/knumolco/knumolco.html.

[34] S. Srivastava and R. Ghosh". A Cluster Based Routing Using a k-Tree Core Backbone for Mobile Ad Hoc Networks. *"Proceedings DIAL M 2002"*, "Sept., 2002".

[35] 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 SIGCOMM'01*, San Diego, CA, Aug. 2001.

[36] R. Swendsen. Dynamics of random sequential adsorption. *Phys. Rev. Lett. A*, 24:504, July 1981.

[37] Terminode. `www.terminodes.org`.

[38] P. F. Tsuchiya. The landmark hierarchy: description and analysis. Technical Report Technical Report MTR-87W00152, MITRE corporation, June 1987.

[39] P. F. Tsuchiya. Landmark routing: architecture, algorithms, and issues. Technical Report Technical Report MTR-87W00174, MITRE corporation, May 1988.

[40] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *Proc. of MobiHOC*, Stanford, CA, Aug. 2000.

[41] G. T. Xuezhi Jin, N. H. Linda Wang and J. Talbot. Irreversible adsorption on nonuniform surfaces: the random site model. *Journal of Physical Chemistry*, 97(17):4256 – 4258, 1993.

[42] H. Zhang and A. Arora". "gs3: Scalable self-configuration and self-healing in wireless networks". *"21st ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing"*, "2002".

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