The Clade Vertebrata: Spines and Routing in Ad Hoc Networks

Raghupathy Sivakumar Bevan Das Vaduvur Bharghavan
Coordinated Sciences Laboratory
University of Illinois at Urbana-Champaign
Email: {sivakumr, das, bharghav}@timely.crhc.uiuc.edu

Abstract

This work proposes a self-organizing, dynamic infrastructure called a spine for efficient routing in ad hoc networks. We present a scalable framework for routing that encompasses a range of knowledge at each spine node, and identify the trade-offs involved for routing at different points in this range. Our routing algorithm requires only partial topology information at each spine node, consisting of the spine structure, dependents of each spine node, propagation of long-lived links, and snooped routing information from ongoing flows. Through worst-case theoretical bounds and simulation of typical scenarios, we show that the spine-based routing with only partial topology information provides good routes at low overhead.

1. Introduction

An *ad hoc network* is a multihop wireless network in which mobile hosts communicate over a shared, scarce wireless channel. Ad hoc networks lack a wired backbone to maintain routes as hosts move or turn off or on. Instead, the hosts in ad hoc network must cooperate with each other to determine routes in a distributed manner. Because of the high mobility of hosts, the frequency of power conservation, and the limited bandwidth available, a routing algorithm for an ad hoc network must have low overhead as well as select good routes.

In particular, the number of nodes maintaining any topology information should be minimal in order to reduce the update overhead on a topology change. At the same time, the overhead involved in accessing this information should also be minimized. In this paper, we introduce a network structure called a *spine* that helps combine low write and read costs, i.e. it minimizes the "write overhead" involved in topology updates while at the same time reducing the "read overhead" for accessing route information.

A *spine* is a self-organizing network structure similar to ones used in some previous packet radio protocols [1, 2, 3]. Unlike the wired backbone of a cellular network, the spine typically does not carry data traffic; the spine is primarily used to compute and update routes. The routes for data packets are the computed shortest paths. However, the spine can provide backup routes immediately to handle interim traffic in case of a link or node failure.

We present a *scalable* framework for routing in ad hoc networks that encompasses a range of knowledge at each spine node, and the trade-offs involved for routing at different points in this range. We describe in detail the use of the spine for computing routes. Our routing algorithm requires only partial topology information at each spine node, consisting of the spine structure, dependents of each spine node, propagation, and snooped routing information from ongoing flows.

Through worst-case theoretical bounds and simulation of typical scenarios, we show that the spine-based routing with only partial topology information provides the low-overhead benefits of routing on demand and the short-route benefits of shortest-path routing.

In Section 2, we present the model and notation for our work. In Section 3, we present a framework for routing using spines and elaborate on one example. The framework encompasses a range of knowledge at each node, from complete global knowledge (ideal link state) to only local knowledge (dynamic source routing). In Section 4, we describe the routing algorithm in detail for the midpoint of this range, when each node in the spine knows the structure of the spine plus some additional global knowledge. In Sections 5, we present our simulation results. Finally, in Section 6, we present our concluding remarks.

2 Network Model

We assume that all the hosts use a common wireless channel for communication. Each host has a fixed transmission range called its *neighborhood*. Two hosts within range of each other can communicate, and are said to be *neighbors*. We assume that communication is bidirectional, though this may not be always true in practice due to interference, fades, and hidden stations.

Let Δ be the maximum number of neighbors a host can have,due to the scheduling assumption, a spine routing instruction such as "Each host locally broadcasts one message" takes $O(\Delta)$ time. We assume the size of each message to be bounded by $O(\Delta \log n)$. In other words, each message can contain update information for $O(\Delta)$ edges or nodes, the size of one neighborhood. But in practice, multiple messages can be coalesced into a single message.

3 The Spine: Infrastructure for Routing

In this section, we discuss the use of a spine as an infrastructure for routing. The spine construction algorithm and the basic spine routing algorithm can be found in [4, 5].

In the basic routing algorithm, each spine node is required to keep a complete topology of G. This condition necessitates update messages for each change in G. As an alternative, we explore ways to reduce the amount of knowledge maintained at each spine node.

There are many useful pieces of information that the spine nodes can keep if they do not maintain complete global knowledge of G. The *spine structure* is the set of spine nodes and spine edges where spine edges do not include non-spine edges between spine nodes. The *domain membership table* lists $dom(v)$ for each v in V. The *local domain topology* of a spine node v consists of v, the set of non-spine nodes $\{x \notin C | v = dom(x)\}$, and all edges incident on v and its dominated nodes. Spine nodes can also learn of routes by either *snooping*, that is, processing headers of packets intended for other destinations, and by exchanging information about *long-lived edges*, that is, edges that have not been deleted or inserted for some period of time. Recall that the $(v, dom(v))$ pairs form a spanning tree, of which C is the interior; a non-spine edge added to this spanning tree forms a unique cycle. Spine nodes can learn of *long-cycle completions*, which are non-spine edges with cycles longer than $diam(C)/k$, for some k.

As a result, we have a range of different spine routing algorithms, depending on the amount of knowledge maintained at each spine node. At one extreme is ideal link-state routing, in which each node maintains a global topology of G. At the other extreme are dynamic source routing [7] and LMR [8], in which no global knowledge is kept except for some cached routes. We list some possibilities in this range in terms of the amount of knowledge maintained at each node:

1. Maintain global knowledge of G at each node: ideal link state.
2. Maintain global knowledge of G at each spine node and up-to-date routing tables at all nodes: basic spine routing.
3. Maintain global knowledge of G at each spine node: basic spine routing as a background process.
4. Maintain the spine structure, domain membership table, and local domain topology at each spine node, plus knowledge gained from snooping, exchange of long-lived edges, and long-cycle completions.
5. Maintain the spine structure, domain membership table, and local domain topology at each spine node.
6. Maintain the spine structure and domain membership table at each spine node, plus knowledge gained from snooping, exchange of long-lived edges, and long-cycle completions.
7. Maintain the spine structure and domain membership table at each spine node.
8. Maintain the spine structure at each spine node, plus knowledge gained from snooping and exchange of long-lived edges.
9. Maintain the spine structure at each spine node.
10. No spine: dynamic source routing [7], DSDV [9] etc.

The advantages to reducing the amount of knowledge at each spine node are a proportional reduction in the amount of topology updates required and in the size of the topology copy sent to each new spine node. For example, an edge insertion or deletion between two non-spine nodes does not trigger a global update when only the spine structure is maintained. For another example, the spine structure is $O(C)$-size, and the domain membership table is $O(n)$-size; these are significantly smaller than the $O(m)$-size topology of G.

On the other hand, when only partial global knowledge is kept at each spine node, route discovery becomes more complicated and path lengths may increase. Route selection is no longer a local decision: each source must send query packets to discover routes. To keep from flooding the network with these queries, we send the queries along paths in C. As a result, the routing becomes similar to hierarchical routing. Hence, the selected routes are not necessarily the shortest paths. As noted in [10], the path lengths increase because routes are constrained to pass through specific nodes.

Additional knowledge, gained from local domain topologies, snooping, exchanges of long-lived edges, and long-cycle completions, can be used to counteract the increase in path length. This extra knowledge allows spine nodes to select non-spine routes.

There are also other options such as maintaining a non-connected spine, either by keeping a dominating set, a multicast group of spine nodes, or increasing the coverage radius from 1 to some small constant number of hops.

4 Spine Routing Algorithms for Ad hoc Networks

As described in the previous section, there can be a range of algorithms based on the amount of knowledge maintained in the spine nodes. In particular, in [4] we propose the basic spine routing algorithm that collects global knowledge of G in the spine nodes and provides optimal routes. But this algorithm does not scale well as ad hoc networks grow beyond small sizes. The amount of overhead caused by propagating information about each individual host becomes prohibitively large. There are two approaches to solve this problem. The first approach that we propose in [5, 6] involves clustering of the network. The algorithm, clustered spine routing, employs a simple clustering structure - a two level hierarchy - with large clusters of 30-50 nodes. Within each cluster, we use our spine routing algorithms. For intercluster routes, we use dynamic source routing [7], with global knowledge of the cluster level maintained at the root of each cluster.

The second approach, which is the focus of this paper, is orthogonal to the first approach and involves reducing the amount of information maintained in each spine node. Our new spine routing algorithm called partial knowledge spine routing makes use of only partial topology information at each spine node. More generally, we advocate the use of the partial knowledge spine routing algorithm in tandem with our clustered spine routing for larger networks. The subsequent sections describe in detail the partial knowledge spine routing.

4.1 Overview of Partial knowledge Spine Routing (PSR)

We describe a spine routing algorithm called PSR that does not maintain complete global knowledge of G at each spine node. Instead, PSR keeps a minimal amount of information to reduce the overhead with only a small increase in path length.

Specifically, each spine node must keep at least the spine structure and the domain membership tables. Without this information, route queries would have to flood the network each time a new route is sought. In addition, the local domain topologies, snooping, long-lived edges, and long-cycle completions are also maintained at each spine node to help select non-spine routes. Without this extra informaion, spine nodes would choose only routes that pass through the spine.

The high-level overview of PSR is as follows.

1. Approximate the Minimum Connected Dominating Set (MCDS) to construct the spine C. Note that the MCDS for a graph $G = (V, E)$ is a minimal connected subset of vertices such that any node v in V is either in this subset or is adjacent to a vertex in this subset.
2. Gather knowledge into the spine.
   (a) For each v not in C, transmit v's adjacency list to $dom(v)$.
   (b) For each u in C, send u's adjacency list to all other nodes in C. In addition, for each v in u's adjacency list, mark (u, v) as a spine edge if (u, v) is a spine edge, and set a flag for v if $u = dom(v)$.
3. Periodically, each u in C exchanges with its spine neighbors the list of long-lived edges and long-cyle completions that u has heard of in previous exchanges.

Lemma: To establish the spine structure, the domain membership table, and the local domain topology at each spine node, the new spine routing algorithm takes $O(n\Delta)$ time using $O(nC + m + n \log n)$ messages [4].

The periodic exchange of long-lived edges and long-cycle completions does not affect the initial time and messages needed. Anyways, this periodic exchange of information requires $O(n)$ information to be exchanged, either taking $O(\Delta(C))$ time and $O(C)$ messages with large messages of $O(n)$ size, or $O(n)$ time and $O(nC)$ messages with smaller messages of $O(\Delta)$ size.

Finally, we note that the overall size of the required knowledge at each node in C is $O(C) = O(n)$, instead of the $O(m)$ required for the basic MCDS routing algorithm. $\square$

4.2 Route discovery with partial global knowledge

Intuitively, the route discovery mechanism is like LMR [8] directed along the spine. First, we use a query packet to alert spine nodes about the route discovery; then, we use a route-building packet to construct the route.

When a source s wants to route to a destination t, s sends a query to $dom(s)$. Given the ID of t, $dom(s)$ looks up $dom(t)$ in the domain membership table and forwards the query to $dom(t)$, along a path in the spine, called a *spine path*. This spine path from $dom(s)$ to $dom(t)$ is unique, since the spine is a tree. At this point, the route from s to t consists of the edge $(s, dom(s))$, the spine path from $dom(s)$ to $dom(t)$, and the edge $(dom(t), t)$.

For the final route from s to t, we wish to avoid the spine as much as possible. Therefore, at every intermediate node in the spine path from $dom(s)$ to $dom(t)$, we seek edges that bypass the spine. Let i be an intermediate node in the spine path. After i receives the query packet sent from $dom(s)$ to

$dom(t)$, i looks at the edges to domains of $i + q$, for $q > 0$. An edge from the domain of i to the domain of $i + q$ is an *outgoing* edge. Likewise, a previous spine node $i - p$, for some $p > 0$, may have selected an edge into the domain of i. An edge from the domain of $i - p$ to the domain of i is an *incoming* edge.

For connecting an incoming edge to an outgoing edge, there are three possibilities:

1. The endpoint in i's domain of both the incoming edge and outgoing edge are the same node. See Figure 1(a).

2. The endpoints are not the same, but there is an edge between the endpoints. See Figure 1(b).

3. There is no direct edge between the two endpoints. However, since both are in i's domain, there is an edge from the incoming edge to i and another edge from i to the outgoing edge. See Figure 1(c).

The next step is to construct the route. First, each spine node in the spine path from $dom(s)$ to $dom(t)$ selects one or more *candidate* outgoing edges, that is, outgoing edges that i nominates to be used in the final route from s to t. Then, $dom(s)$ sends a route-building packet along each of its selected candidate edges. For each node v that receives a copy of this route-building packet, if v is not in the spine, then v contacts $dom(v)$ to find the corresponding candidate outgoing edge for $dom(v)$'s domain. If v is in the spine, then v uses its own knowledge to look up the corresponding candidate outgoing edge. Node v then forwards the route-building packet over the candidate outgoing edge. This process repeats until the packet reaches $dom(t)$.

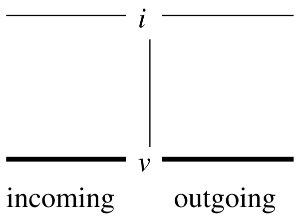
When an edge is deleted, the routes are easily updated. Let u be the upstream node of the deleted edge, that is, the endpoint closer to s. Node u alerts $dom(u)$, and $dom(u)$ triggers another route-discovery from u to t (not from s to t).

4.3 Discovering better bypass routes

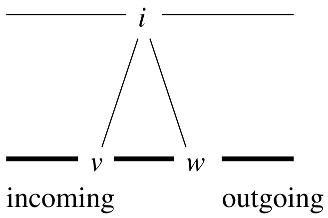
Even with the best selection of candidates for the route-building phase of the route discovery of Section 4.2, the bypass routes still follow the structure of the spine. Hence shorter routes may be overlooked, as in Figure 2.

To discover routes such as $\langle s, a, b, t\rangle$ in Figure 2, we use snooping, exchange of long-lived edges, and long-cycle completions. First, $dom(s)$ may overhear another packet that uses edge (a, b). From the domain membership tables, $dom(s)$ can look up $dom(a)$ and $dom(b)$ and determine that (a, b) contributes to the shorter route $\langle s, a, b, t\rangle$.

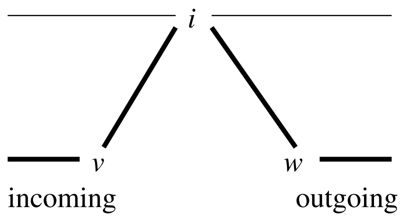
Second, let τ be a threshold for long-lived edges; if an edge is unchanged for more than τ seconds, then its endpoints mark it as long-lived. So, at τ seconds after (a, b)



(a) incoming and outgoing edges share endpoint in domain of node i



(b) one intra-domain edge (v,w) connects incoming edge to v with outgoing edge from w



(c) two intra-domain edges (v,i) and (i,w) connect incoming edge to v with outgoing edge from w

Figure 1. Cases of incoming and outgoing edges with respect to i's domain. In all three cases, $i \in C$, and the two top edges connect i to $i-1$ and $i+1$ in the C-path from $dom(s)$ to $dom(t)$. The thick edges denote the edges that i suggests be part of the route from s to t; thin edges not in the C-path represent intradomain edges not in the suggested route.

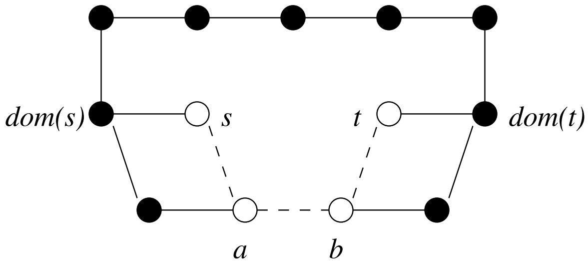


Figure 2. The spine nodes are in black, the non-spine nodes in white. The edge (a, b) is a "long-cycle completion," that is, $\langle s, a, b, t\rangle$ is a much shorter route than the route constructed along the spine path from $dom(s)$ to $dom(t)$.

is inserted into G, a informs $dom(a)$ of (a, b) as long-lived, likewise for b to $dom(b)$. At 2τ seconds, $dom(a)$ informs $dom(s)$, and $dom(b)$ informs $dom(t)$ about (a, b) as a long-lived edge. Hence, after 2τ seconds, $dom(s)$ can use $\langle s, a, b, t\rangle$ based on information from the exchange of long-lived edges.

Third, the length of (a, b)'s cycle is 10 edges. If $10 > diam(C)/2$, then (a, b) is a long-cycle completion with respect to the spine. Hence, in this case, $dom(s)$ would know about (a, b) from $dom(a)$'s broadcast of long-cycle completions. Similarly, $dom(t)$ would know about (a, b) from $dom(b)$.

4.4 Updates

To maintain the spine structure, domain membership table, and local domain topology at each spine node, our new spine routing algorithm sends global update messages when the spine structure or domain membership changes and local update messages when a domain topology changes. The specific messages and processes depend on the node that moves, whether the node is a non-spine node, a leaf in the spine, or an interior node in the spine. In this section, we briefly summarize how these cases are treated.

- Non-spine node: let v be a non-spine node and let $u = dom(v)$.
  - Within domain. v has some incident edges deleted and some incident edges inserted. v reports these changes to u. In addition, the other endpoints of the changed edges report the changes to their dominators. (These changes do <u>not</u> have to be announced to the rest of C.)
  - Into new domain. v moves from the domain of u to another spine node u'. In addition to the edge changes reported locally in the previous case, u' reports to the rest of C that u' is now $dom(v)$.
  - *Orphan* case. v moves so that none of its neighbors is in C. v selects its maximum-degree neighbor w to join C, and w asks $dom(w)$ to send a copy of G to w. In addition, $dom(w)$ reports to the rest of C that w has joined C and that $dom(v) = w$.
  - Isolation. v moves away from the rest of the nodes. After $O(diam(C)\Delta(C))$ time, u knows that v has not moved into another node's domain. Thus, u deduces that v has moved away and reports to the rest of C that v is now isolated.
- Leaf spine node. Let v be a leaf spine node. The movement of v generates update messages as in the case of a non-spine node. In addition, the nodes formerly dominated by v must find new dominators. Let w be such a non-spine node. If w is adjacent to another spine node u, then u becomes $dom(w)$. Otherwise, if w is orphaned, then w forces its maximum-degree neighbor to join the spine. Finally, for those non-spine nodes w isolated from the spine by the movement of v, these nodes construct a spine among themselves.
- Interior spine node. Let v be an interior spine node. The movement of v generates update messages as in the case of a non-spine node, and the non-spine nodes formerly dominated by v generate messages as in the case of the leaf spine node. In addition, the movement of the interior spine node partitions C into at most $\delta_C(v)$ components. After the updates about incident spine edges on v are broadcast through C, the remaining nodes in C do not know whether G is also partitioned. To identify potential edges to connect the components of C, the remaining nodes in C can use information from snooping, exchanges of long-lived edges, and long-cycle completions. However, in the worst case, the components of C have to redo the MST algorithm of Gallager, Humblet, and Spira [11] to reconnect.

Lemma: To update the global knowledge at each spine node, the new spine routing algorithm takes the following time and messages, using $O(\Delta)$-size messages:

- non-spine node
  - within domain: $O(\Delta)$ time, $O(\Delta)$ messages
  - into new domain: $O(\Delta + diam(C)\Delta(C))$ time, $O(\Delta + C)$ messages
  - orphan: $O(\Delta + (n/\Delta) + diam(C)\Delta(C))$ time, $O(\Delta + (n/\Delta) + C)$ messages

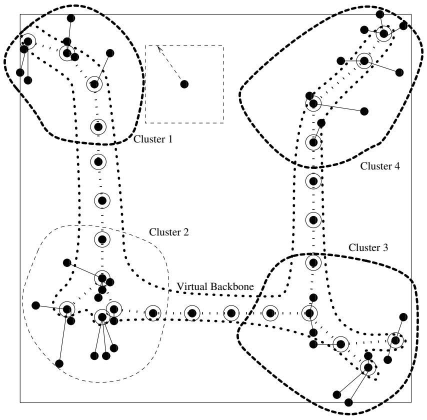


Figure 3. Graph 1 is a clustered graph with four distinct clusters. Nodes are adjacent if within 1 unit square of each other, where the unit square is drawn to scale in the upper middle of the area.

  - isolation: $O(diam(C)\Delta(C))$ time, $O(C)$ messages

- leaf spine node: $O(\Delta + (n/\Delta) + diam(C)\Delta(C))$ time, $O(n)$ messages

- interior spine node: $O(\Delta+(n/\Delta)+diam(C)\Delta(C))$ time, $O(n)$ messages

It is to be noted that the overhead in practice is not as high because large messages are used when a new node joins the spine. For this new algorithm, the size of this large message has to be only $O(n)$, instead of $O(m)$ for the complete topology of G.

For routing updates, we rely on the spine structure. Suppose (u, v) is deleted. For each route such that u is upstream of v, that is, closer to the source, u alerts $dom(u)$, and $dom(u)$ triggers another route-discovery from u to the destination. Similarly, v alerts $dom(v)$ for each route such that v is upstream of u. On the other hand, suppose (u, v) is inserted. Immediately, this new edge is entered into the local domain topologies of $dom(u)$ and $dom(v)$, and within a finite time, (u, v) is announced to the rest of C as long-lived via the exchange of long-lived edges.

5 Simulations

This section demonstrates how PSR performs when nodes move and the topology changes. The setup for the presented set of simulations is the following. Initially, the spine is assumed to have been computed. Each spine node has initially only the information about the structure of the spine, domain membership table, local domain topology, and long cycles in the spine. Initially, for the UDP flows that are ongoing, we assume that the routes have been computed. Thus, spine nodes along the route of this flow eventually snoop and learn the topology information. The initial topology is as shown in Figure 3. Over the period of the simulation (200 seconds), we moved the same 10 or 15 nodes to the same locations. The user traffic model is the following. At any time, there are four senders, one from each cluster. Each sender has a 'working set' of five receivers at any time - two from its own cluster, one each from the other clusters. Every 20 seconds, any one of the receivers from the working set of each sender is chosen at random, and replaced with another receiver from the same cluster. Every 100 seconds, two of the senders are chosen at random and replaced with two others from the same clusters. Each sender sends 30 packets a second in three bursts of 10 packets each, one to each receiver (i.e. each sender sends 10 packets each to any three of the five receivers in its working set). All transmissions are UDP.

Case	A	B	C	D	E	F	G	H
1.1	84	44	2.1	10.6	22%	6.0	3.0%	0
1.2	84	46	2.1	10.4	20%	6.0	3.6%	0
2.1	96	39	2.8	10.9	19%	6.4	5.1%	1%
3.1	79	51	1.7	11.0	24%	5.2	4.1%	1%
3.2	79	51	1.7	11.0	24%	5.2	4.1%	1%
4.1	114	60	3.0	10.1	23%	6.0	3.8%	2.5%
4.2	121	69	2.8	10.4	18%	5.6	4.2%	2.8%

Table 1. Simulation results for the new spine algorithm when multiple nodes move.

For the above simulation setup, we measured the following parameters: (A) messages to update, (B) time to update (in slots), (C) avg. number of hops by which route exceeds shortest path, (D) avg. number of hops for shortest path, (E) % of routes that were derived from some snooped value, (F) avg. time for response for a route query (in slots), (G) % of packets routed via backbone, and (H) % of packets lost. The four cases we simulated are the following: case 1 - non-spine nodes move in the same domain or to another domain, case 2 - non-spine nodes move to an area with no dominator, case 3 - combination of case 1 and case 2, case 4 - non-spine nodes and MCDS leaf nodes move randomly. For each case, we performed two simulations, the first with 10 nodes moving and the second with 15 nodes moving. Table 1 presents the performance results.

The key results from the simulation of the new spine algorithm are the following: (a) there is an acceptable tradeoff between routing overhead and the optimality of shortest paths, (b) the spine serves as a short-term backup route during mobility, (c) the fraction of lost packets is definitely within tolerable limits, and (d) the algorithm scales well. On the other hand, these simulations are on a small set of test scenarios and may not be representative of the general case. Also, as pointed out before, these tests are for UDP rather than TCP. Besides, we have not yet performed any tests with multicast. In our ongoing work, we are addressing all the above issues.

6 Concluding remarks

We have presented a framework for routing in ad hoc networks with specific focus on PSR. Within this framework, we approximate an MCDS for use as the spine and maintain partial global knowledge supplemented by snooping and periodic exchanges of long-lived edges and long-cycle completions. The resulting spine routing algorithm is a hybrid between shortest-paths routing and routing on demand. Finally, the use of the spine can increase the fault-tolerance of our routing algorithm and also add multicast and broadcast routing capabilities.

References

[1] A. Ephremides, J. E. Wieselthier, and D. J. Baker, "A design concept for reliable mobile radio networks with frequency hopping signaling," *Proceedings of the IEEE*, **75**, 1 (Jan 1987) 56–73.

[2] M. Gerla and J. T.-C. Tsai, "Multicluster, mobile, multimedia radio network," *ACM J. Wireless Networks*, **1**, 3 (1995) 255–265.

[3] A. Parekh, "Finding small dominating sets in stationless mobile packet radio networks," Tech. Rep. LIDS-P-2050, Laboratory for Information and Decision Systems, M.I.T., Cambridge, MA, 1991, 19 pp.

[4] B. Das and V. Bharghavan, "Routing in ad-hoc networks using minimum connected dominating sets," *IEEE International Conference on Communications* (ICC '97), (June 1997).

[5] B. Das, R. Sivakumar, and V. Bharghavan, "Routing in ad-hoc networks using a virtual backbone," *Proceedings of the 6th International Conference on Computer Communications and Networks* (IC3N '97), (Sept 1997) 1–20.

[6] R.Sivakumar, B.Das, and V.Bharghavan, "Spine Routing in Ad Hoc Networks", to appear in *ACM/Baltzer Cluster Computing Journal*, 1998.

[7] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," *Mobile Computing*, (ed. T. Imielinski and H. Korth), Kluwer Academic Publishers, 1996.

[8] M. S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *ACM J. Wireless Networks*, **1**, 1 (1995) 61–81.

[9] C.E.Perkins and P.Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *Computer Communications Review*, **24**, 1 (ACM SIGCOMM 1994) 234–244.

[10] L. Kleinrock and F. Kamoun, "Hierarchical routing for large networks: performance evaluation and optimization," *Computer Networks*, **1**, (1977) 155–174.

[11] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Programming Lang. Systems*, **5**, 1 (Jan 1983) 66–77.

[12] S. Corson, J. Macker, and S. Batsell, "Architectural considerations for mobile mesh networking", Internet DRAFT RFC Version 2, May 1996.

[13] C. K. Toh, "A novel distributed routing protocol to support ad-hoc mobile computing," *Proceedings of 15th IEEE Annual International Phoenix Conference on Computers and Communications*, (1996) 480–486.