Topology Management in Ad Hoc Networks [*]

Lichun Bao
Computer Science Department
University of California
Santa Cruz, CA 95064
baolc@soe.ucsc.edu

J.J. Garcia-Luna-Aceves
Computer Engineering Department
University of California
Santa Cruz, CA 95064
jj@soe.ucsc.edu

ABSTRACT

The efficiency of a communication network depends not only on its control protocols, but also on its topology. We propose a distributed topology management algorithm that constructs and maintains a backbone topology based on a minimal dominating set (MDS) of the network. According to this algorithm, each node determines the membership in the MDS for itself and its one-hop neighbors based on two-hop neighbor information that is disseminated among neighboring nodes. The algorithm then ensures that the members of the MDS are connected into a connected dominating set (CDS), which can be used to form the backbone infrastructure of the communication network for such purposes as routing. The correctness of the algorithm is proven, and the efficiency is compared with other topology management heuristics using simulations. Our algorithm shows better behavior and higher stability in ad hoc networks than prior algorithms.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication—*ad hoc networks*; C.2.1 [**Network Architecture and Design**]: Network topology—*topology control*

General Terms

Algorithms

Keywords

Ad hoc networks, minimum dominating set, connected dominating set

1. INTRODUCTION

The topology of an ad hoc network plays a key role in the performance of the control algorithms used in the network for such purposes as scheduling of transmissions, routing, and broadcasting. In many cases, several network links are not needed for establishing efficient sharing of the channel among neighboring nodes or the routing of data packets. Weeding out redundant and unnecessary topology information is usually called *topology control* or *topology management.* Topology management has been effectively applied in ad hoc networks to supplement routing control protocols, such as CEDAR [24, 25], and to schedule efficient channel access to propagate broadcast data [10].

There are two approaches to topology management in ad hoc networks—power control and hierarchical topology organization. Power control mechanisms adjust the power on a per-node basis, so that one-hop neighbor connectivity is balanced and overall network connectivity is ensured [15, 20, 26]. Li *et al.* [18] proved that network connectivity is minimally maintained as long as the decreased power level keeps at least one neighbor remaining connected at every $2\pi/3$ to $5\pi/6$ angular separation. Ramanathan *et al.* [22] proposed to incrementally adjust nodes' power levels so as to keep network connectivity at each step. However, except for the early work by Takagi and Kleinrock [26], topologies derived from power-control schemes often result in unidirectional links that create harmful interference due to the different transmission ranges among one-hop neighbors [21]. The dependencies on volatile information in mobile networks, such as node locations [15], signal strength or angular positions [18] also contribute to the instability of topology control algorithms based on power control. Furthermore, some distributed implementations of these algorithms can hardly improve the throughput of mobile networks [22].

In this paper, we focus on hierarchical topology control, with which a subset of the network nodes is selected to serve as the network backbone over which essential network control functions are supported (e.g., [17]). This approach to topology control is often called *clustering*, and consist of selecting a set of *clusterheads* in a way that every node is associated with a clusterhead, and clusterheads are connected with one another directly or by means of *gateway*s, so that the union of gateways and clusterheads constitute a connected backbone. Once elected, the clusterheads and the gateways help reduce the complexity of maintaining topology information, and can simplify such essential functions as routing, bandwidth allocation, channel access, power control or virtual-circuit support. For clustering to be effective, the

links and nodes that are part of the backbone (i.e., cluster-heads, gateways, and the links that connect them) must be close to minimum and must also be connected.

Ideally, topology control based on clustering would select a minimum and sufficient number of links to serve as the communication backbone of the network, while reducing network maintenance and control overhead. In graph theory, the minimum dominating set problem and the relevant minimum connected dominating set (MCDS) problem best describe the clustering approach to topology control.

The dominating set problem in graph theory consists of finding a subset of nodes with the following property: each node is either in the dominating set, or is adjacent to a node in the dominating set. The members of the dominating set are often called *clusterheads*, whereas the other nodes without special functionalities are called *hosts*. The problem of computing the minimum dominating set is known to be NP-hard [1, 12] even when the complete network topology is available. In ad hoc networks, the difficulty of acquiring complete network topology makes it impossible to compute the "minimum" dominating sets. Instead, a *minimal* dominating set (MDS) is usually pursued based on various heuristics that can guarantee a local minimum election of the dominators in a polynomial number of steps.

The MCDS problem consists of obtaining a minimum subset of nodes in the original graph, such that the nodes compose a dominating set of the graph, and the induced subgraph of an MCDS has the same number of connected components as the original graph. Although attractive, the MCDS is a well-known NP-complete problem in graph theory. Accordingly, sub-optimum solutions must be used to approximate the optimum solution. In this paper, we refer to a sub-optimum solution of the MCDS problem as a connected dominating set (CDS).

In the heuristics that have been proposed in the past, and which we summarize in the following paragraphs, the selected clusterheads are equivalent to a *minimal* dominating set (MDS), and the selected gateways are chosen in such a way that the union of gateways and clusterheads forms the CDS.

Clusterheads can be elected via non-deterministic negotiations or by applying deterministic criteria. Negotiations require multiple incremental steps, and may incur an election jitter during the process, because of the lack of consensus about the nodes being elected as the clusterheads. Examples of this approach are the "core" extraction algorithm [25] and the spanning-tree algorithm [14]. SPAN [8] allows a node to delay the announcement of becoming a clusterhead for random amounts of time to attempt to attain minimum conflicts between clusterheads in its one-hop neighborhood.

In contrast, deterministic criteria can determine the clusterheads in a single round. Different heuristics have been used to form clusters and to elect clusterheads. Several approaches [1, 3, 11, 13, 19] utilized the node identifiers to elect the clusterhead within one or multiple hops. For simplicity, we refer to this approach as **Lowest ID** in our comparative analysis. Banerjee and Khuller [4] assigned a weight value to each node for clusterhead election, which is essentially the same as **Lowest ID**.

The node degree is another commonly used heuristic in which nodes with higher degrees are more likely to become clusterheads [14, 16, 25]. We refer to this approach as **Max Degree**. Chiang *et al.* [9] have shown that the **Lowest ID** algorithm performs better than the clusterhead election algorithms based on node degrees in terms of clusterhead stability in ad hoc networks.

Basu *et al.* [7] suggested to use the mean received-signal strength variations as the metric in clusterhead elections, which favors relatively stationary nodes to become the clusterheads. We refer to this approach as **MOBIC**.

Cluster formation simplifies topology maintenance in ad hoc networks. However, it has a negative effect on the clusterheads, because a clusterhead drains its energy more quickly than a normal node. Therefore, a clusterhead election algorithm must also consider load balancing of the clusterhead role to avoid node or network failure. Except for **Lowest ID**, the aforementioned election algorithms inherently provide clusterhead load balancing in mobile networks. To improve **Lowest ID**, Amis *et al.* [2] provided clusterhead load balancing, which we call **Load Balance**, by running a virtual identifier (VID) and a budget counter at each node. **Load Balance** uses the VID for elections, and the budget for the clusterhead term, thus posing equal opportunity for each node to become a clusterhead.

However, all the existing heuristics have addressed only some aspects of characteristics in ad hoc networks, such as load-balancing, mobility, or algorithmic convergence, while ignoring the others.

Clustering algorithms that build clusters within d hops from the clusterhead (called d-clustering) have also been proposed [1, 17, 23]. However, d-clustering requires flooding in search of clusterheads [1], thus obviates the purpose of the topology management for efficiency. In this paper, we only consider clustering approaches in which a host is always one hop away from a clusterhead.

We introduce a novel approach to the solution of the connected dominating set election problem, which we call *topology management by priority ordering* or **TMPO**. Our approach uses the neighbor-aware contention resolution (NCR) algorithm [5] to provide fast convergence and load-balancing with regard to the battery life and mobility of mobile nodes. Based on NCR, **TMPO** assigns randomized priorities to mobile stations, and elects a minimal dominating set (MDS) and the connected dominating set (CDS) of an ad hoc network according to these priorities. In doing so, **TMPO** requires only two-hop neighbor information for the MDS elections. The dynamic priorities assigned to nodes are derived from the node identifiers and their "willingness" to participate in the backbone formations. The willingness of a node is a function of the mobility and battery life of the node. The integrated consideration of mobility, battery life and deterministic node priorities makes **TMPO** one of the best performing heuristics for topology management in ad hoc networks.

The rest of the paper is organized as follows. Section 2 describes the network topology assumptions made in this paper. Section 3 specifies the minimal dominating set (MDS) election algorithm based on node priorities. Section 4 describes the algorithm that extends the MDS into a connected dominating set (CDS) of an ad hoc network. Section 5 proves the correctness of these MDS and CDS election algorithms. Section 6 analyzes the average size of the elected MDS using a probabilistic model. Section 7 compares **TMPO** with other CDS computation algorithms using simulations. Section 8 summarizes the paper and its key contributions.

2. NETWORK ASSUMPTIONS AND NOTATION

This work assumes that an ad hoc network comprises a group of mobile nodes communicating through a common broadcast channel using omni-directional antennas with the same transmission range. The topology of an ad hoc network is thus presented by an undirected graph $G = (V, E)$, where V is the set of network nodes, and $E \subseteq V \times V$ is the set of links between nodes. The existence of a link $(u, v) \in E$ also means $(v, u) \in E$, and that nodes u and v are within the packet-reception range of each other, in which case u and v are called *one-hop neighbors* of each other. The set of one-hop neighbors of a node i is denoted by N_i^1. Two nodes that are not connected but share at least one common one-hop neighbor are called *two-hop neighbor* of each other.

Each node has one unique identifier, and all transmissions are omnidirectional with the same transmission range. Time is slotted, and synchronization among nodes exists to the time-slot boundary. The current time t is defined by the corresponding time slot number, starting from a consensus temporal point in the past. In addition, a reliable neighbor protocol is assumed to enable the quick update of two-hop neighbor information at each node. Bao and Garcia-Luna-Aceves [6] have proposed approaches for acquiring and synchronizing two-hop neighbor information.

For convenience, the notation and terminology used in the rest of this paper are summarized in Table 1.

Table 1: Notation

MDS	The minimal dominating set.
CDS	The connected dominating set.
Clusterhead	A member of the MDS in a network.
Gateway	A node that connects clusterheads to form the CDS of the network.
Doorway	A node that extends the reach of a clusterhead to form the CDS.
Host	A regular client of a network.
N_i^1	The set of one-hop neighbors of node i.
T	Node priority recomputation interval.
$i.\text{off}$	Time slot offset of node i for priority recomputation.
$i.\text{prio}$	The priority of node i.
$i.\text{type}$	The type of node i, which is one of clusterhead, host, gateway and doorway.
$i.\text{ch}$	The clusterhead elected by node i.
$i.\text{workfor}$	A set of clusterheads or doorways that node i connects to form the CDS.
E_i	The energy level of node i.
s_i	The speed of node i in terms of meters per second.
W_i	The willingness value of node i.

3. MINIMAL DOMINATING SET

3.1 Clusterhead Election Approach

Our approach to establishing a minimal dominating set (MDS) is based on three key observations. First, using negotiations among nodes to establish which nodes should be in the MDS incurs substantial overhead when nodes move around and the quality of links changes frequently. Hence, nodes should be allowed to make MDS membership decisions based on local information. Second, because in an MDS every node is one hop away from a clusterhead, the local information needed at any node needs to include only nodes that are one and two hops away from the node itself. Third, having too many clusterheads around the same set of nodes does not lead to an MDS. Hence, to attain a selection of nodes to the MDS without negotiation, nodes should rank one another using the two-hop neighborhood information they need.

Based on the above, the approach adopted in **TMPO** consists of each node communicating to its neighbors information about all its one-hop neighbors. Using this information, each node computes a priority for each node in its two-hop neighborhood, such that no two nodes can have the same priority at the same instant of time. A node then decides to become a clusterhead if either one of the following criteria are satisfied:

1. The node has the highest priority in its one-hop neighborhood.

2. The node has the highest priority in the one-hop neighborhood of one of its one-hop neighbors.
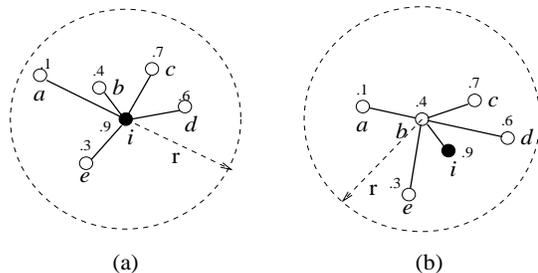

Figure 1: Two cases that enable node i becoming a clusterhead.

Figure 1 illustrates the two criteria that make node i a clusterhead. In Figure 1 (a), node i has the highest priority among its one-hop neighbors. In Figure 1 (b), node i has the highest priority among node b's one-hop neighbors. The number next to each node is the sample priority at a particular moment. For convenience, we represent node priorities using fractional numbers over the range $[0, 1)$ throughout this paper. The algorithms can be easily converted to integer operations in practice. We discuss the computation of node priorities in the following sections.

3.2 Components of Node Priorities

Given that clusterheads provide the backbone for a number of network control functions, their energy consumption is more pronounced than that of ordinary hosts. Low-energy nodes must try to avoid serving as clusterheads to save energy. However, to balance the load of serving as clusterheads, every node should take the responsibility of serving as a clusterhead for some period of time with some likelihood. Furthermore, node mobility has to be considered in clusterhead elections, so that the MDS experiences the least structural changes over time.

To take into account the mobility and energy levels of nodes in their election as members of the MDS, we define

the two-hop neighbor information needed to assign node priorities as consisting of three components: (a) the identifiers of the node's neighbors, (b) the present time, and (c) a "willingness" value assigned to a node as a function of its mobility and energy level.

We denote the willingness value of node i by W_i, the speed of node i by a scalar $s_i \in (0, \infty)$ in terms of meters per second, and the remaining energy on node i as $E_i \in [0, 1)$. The willingness $W_i = f(s_i, E_i)$ is a function that should be defined according to the following criteria:

1. To enhance survivability, each node should have the responsibility of serve as a clusterhead with some non-zero probability determined by its willingness value.

2. To help with the stability of the MDS and the frequency with which clusterhead elections must take place, the willingness value of a node should remain constant as long as the variation of the speed and energy level of the node do not exceed some threshold values.

3. To avoid electing clusterheads that quickly lose connectivity with their neighbors after being elected, the willingness value of a node should decrease drastically after the mobility of the node exceeds a given value.

4. To prolong the battery life of a node, its willingness value should decrease drastically after the remaining energy of the node drops below as given level.

There are many possible functions that can be used to compute the willingness value of a node while adhering to the above criteria. Our approach is given by Eq. (1).

$$W_i = 2^{\log_2(E_i * 0.9) \log_2(s_i + 2)} \qquad (1)$$

where the constants 0.9 and 2 in Eq. (1) eliminate the boundary conditions in the logarithmic operations. The logarithmic operations on the speed and the remaining energy values render higher willingness values in the high energy and low speed field, while giving close to zero values in the low-energy and high-speed region. Figure 2 illustrates the effect of the two factors on the willingness values.
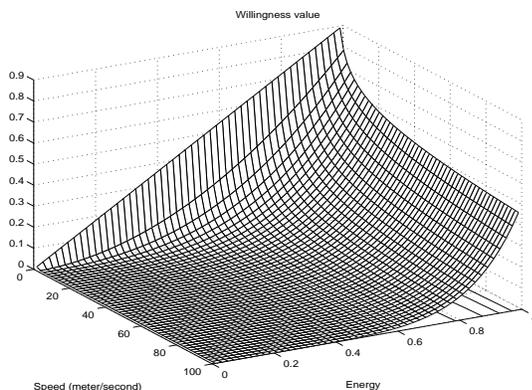
Figure 2: The willingness function based on speed and remaining energy.

3.3 Clusterhead Election Algorithm

As stated before, time is slotted, and the time parameter is represented by the current time-slot number. Node priorities are computed using a pseudo-random number generator based on the node identifiers, their willingness values, and the current time.

The priorities of nodes change periodically to trigger clusterhead re-elections aimed at distributing the role of clusterheads among nodes. The recomputation period is denoted by T and is a multiple of time slots and is the same for all nodes in the network.

The priority of each node is recomputed asynchronously using a time-slot offset so as to avoid synchronous sudden loss of the old network states. Eq. (2) is used at each node i to compute locally its time-slot offset, which is denoted by $i.\texttt{off}$.

$$i.\texttt{off} = \lfloor \texttt{Hash}(i) \cdot T \rfloor \qquad (2)$$

where the function $\texttt{Hash}(x)$ is a pseudo-random number generator that produces a uniformly distributed random number over range $[0, 1)$ based on the input bit-stream x. The floor operation gives an integer offset.

The recomputation of the priority of a node happens whenever the current time slot is a multiple of the recomputation period T plus the time-slot offset of the node, i.e., when the current time is $t = kT + i.\texttt{off}$, and $k = 0, 1, \cdots$. At that time slot, the priority of node i, denoted by $i.\texttt{prio}$, is recomputed according to the following formula:

$$i.\texttt{prio} = \texttt{Hash}(k \oplus i) \cdot W_i \oplus i \qquad (3)$$

where the function $\texttt{Hash}$ is the same as the one defined for Eq. (2), and the sign "$\oplus$" is designated to carry out the bit-concatenation operation on its operands, and has lower order than other operations. The last concatenation with i in the final result is included to distinguish the priorities of different nodes. Once computed, the priority of a node remains the same during the entire recomputation period T.

Because each node knows the node identifiers of its two-hop neighborhood, node i can determine locally which other nodes must recompute their priorities during the same time slot from Eq. (equation:startuptime). Because the willingness values are reported by nodes in its two-hop neighborhood, node i uses Eq. (equation:priority) to compute its own priority and the priorities of all nodes in its two-hop neighborhood that must recompute their priorities. Because Eq. (equation:priority) renders different priority values for different node identifiers, only one node can be selected to have the highest priority during the time slot when node i must recompute its priority. Once nodes have consistent two-hop neighborhood information, if node i decides that it must become a clusterhead, its neighbors do too, because the nodes run the same algorithms using the same information.

The algorithms for determining the clusterhead status of node i are described in Figures 3 – 4 using C-style pseudo code.

Function **Init** in Figure 3 initializes the data structures at node i. For simplicity, the energy level and speed remain constant until the beginning of the next recomputation period. The willingness value of node i is computed according to Eq. (1) and the nodal type is initialized to $\texttt{Host}$ (**Init**

```
/* Initialize */
Init(i, t)
{
1    i.workfor = ∅;
2    W_i = 2^{log_2(E_i*0.9) log_2(s_i+2)};
3    i.type = Host;

     /* Recompute priorities. */
4    for (j ∈ N_i^1 ∪ (⋃_{k∈N_i^1} N_k^1)) {
5       if (t ≡ 0)
6          j.prio = Hash(j) · W_j ⊕ j;
7       else if (t − j.off mod T ≡ 0)
8          j.prio = Hash((t−j.off)/T ⊕ j) · W_j ⊕ j;
9    }
} /* End of Init. */
```

Figure 3: TMPO function for initialization.

lines 2-3). Node i also computes the priority for each two-hop neighbor according to the recomputation period of the neighbor (**Init** lines 4-9).

```
isClusterhead(i)
{
1    for (j ∈ N_i^1 ∪ {i}) {
2       j.ch = j;
        /* Find j's clusterhead. */
3       for (k ∈ N_j^1)
4          if (k.prio > j.ch.prio)
5             j.ch = k;

6       if (j.ch ≡ i)
7          i.type = Clusterhead;
8    }
} /* End of isClusterhead. */
```

Figure 4: TMPO function for electing a clusterhead.

Function **isClusterhead** in Figure 4 elects the clusterhead of a node i, which is indicated by the field $i.\text{ch}$ in the neighbor data structure.

If node i becomes a clusterhead after computing the MDS using function **isClusterhead**, its clusterhead-type attribute needs to be propagated to its two-hop neighbors for further computations.

4. CONNECTED DOMINATING SET

4.1 CDS Election

Because the maximum distance from a clusterhead in the minimal dominating set (MDS) to the closest clusterhead is three, which we prove in Theorem 2, we can derive the CDS of a network by adding some nodes to the MDS, such that clusterheads within two or three hops are connected. Two other types of nodes, called *doorways* and *gateways*, are elected to derive the CDS based only on the priorities of the neighbors with two hops from each node.

The CDS of a network topology is constructed in two steps. In the first step, if two clusterheads in the MDS

are separated by three hops and there are no other clusterheads between them, a node with the highest priority on the shortest paths between the two clusterheads is elected as a *doorway*, and is added to the CDS. Therefore, the addition of a doorway brings the connected components in which the two clusterheads reside one hop closer.

In the second step, if two clusterheads or one clusterhead and one doorway node are only two hops away and there are no other clusterheads between them, one of the nodes between them with the highest priority becomes a *gateway* to connect clusterhead to clusterhead or doorway to clusterhead. After these steps, the CDS is formed. Figures 5 – 6 specify the two steps, and the function **TMPO** in Figure 7 combines all the topology management algorithms.

```
isDoorway(i)
{
1    if (i.type ≡ Clusterhead)
2       return;

3    for (j ∈ N_i^1 and j.type ≡ Clusterhead) {
4       for (k ∈ N_i^1 and k ≠ j
            and k.type ≠ Clusterhead) {
5          for (n ∈ N_k^1, n.type≡Clusterhead
               and n ∉ N_i^1 ∪ N_j^1) {
               /* Case (a) in Figure 8. */
6             if (∃m ∈ N_i^1, {j, n} ⊆ N_m^1)
7                continue n;

8             for (m ∈ N_i^1) {
                  /* Case (b) or (c) in Figure 8. */
9                if (n ∈ N_m^1 and ((m.type ≡ Clusterhead) or
                     (∃p ∈ N_i^1 ∩ N_m^1, p.type ≡ Clusterhead)))
10                  continue n;
11            } /* m */

12            for (m ∈ N_i^1 ∩ N_n^1) {
                  /* Case (d) in Figure 8. */
13                if ((m.prio > i.prio) or
14                   (∃p ∈ N_j^1 ∩ N_m^1, p.prio > i.prio))
15                  continue n;
16            } /* m */

17            i.type = Doorway;
18            i.workfor = i.workfor ∪{j,n};
19         } /* n */
20      } /* k */
21   } /* j */
} /* End of isDoorway. */
```

Figure 5: TMPO function for electing a doorway.

Function **isDoorway** in Figure 5 determines whether node i can become a doorway for other clusterheads. To decide whether node i becomes a doorway for clusterhead n and j, node i needs to assert that

1. Clusterheads n and j are not two hops away (**isDoorway** lines 3-7 illustrated by case (a) in Figure 8).

2. There is no other clusterhead m on the shortest path between clusterhead n and j (**isDoorway** lines 8-11 illustrated by case (b) and (c) in Figure 8).

3. There is no other node m with higher priority than node i on the three-hop path between clusterhead n and j (**isDoorway** lines 12-16 illustrated by case (d) in Figure 8).

```
isGateway(i)
{
1   if (i.type ∈ {Clusterhead, Doorway})
2      return;

3   for (j ∈ N_i^1
4      and j.type ∈ {Clusterhead, Doorway}) {
5      for (k ∈ N_i^1 and k ≠ j and k ∉ N_j^1 and
6         k.type ∈ {Clusterhead, Doorway} and
7         (k.type ≢ Doorway or
8            j.type ≢ Doorway) {
9         if (∃n ∈ N_j^1 ∩ N_k^1, n ≠ i and
                   (/* Case (a) in Figure 9. */
10            n.type ∈ {Clusterhead, Doorway} or
                   /* Case (b) in Figure 9. */
11            n.prio > i.prio))
12         continue k;
13      else {
14         i.type = Gateway;
15         i.workfor = i.workfor ∪{j, k};
16      }
17   } /* k */
18 } /* j */
} /* End of isGateway. */
```

Figure 6: TMPO function for electing a gateway.

```
TMPO(i, t)
{
1   i.oldType = i.type;
2   Init(i, t);
3   isClusterhead(i);
4   isDoorway(i);
5   isGateway(i);

6   /* i's status changes? */
7   if (i.type ≠ i.oldType)
8      Propagate i.type;
} /* End of TMPO. */
```

Figure 7: TMPO description

If the three assertions are satisfied, node i becomes a doorway (**isDoorway** line 17). The attribute i.workfor is the set of clusterheads that make node i become a doorway (**isDoorway** line 18). A doorway needs to notify its one-hop neighbors about its current status for gateway elections.

Function **isGateway** in Figure 6 determines whether node i becomes a gateway to connect two clusterheads or one clusterhead and another doorway, k and j (**isGateway** lines 3-8). According to Figure 9, if there is another clusterhead or doorway between node k and j (**isGateway** line 10), or there is another node with higher priority than node i between node k and node j (**isGateway** line 11), node i cannot become a gateway. Otherwise, node i becomes a gateway, and the attribute i.workfor is the set of nodes that make i become a gateway (**isGateway** lines 13-16).

Function **TMPO** is called after every neighbor information update or when the recomputation period of a neighbor starts. After calling **TMPO**, if node i changes its role between clusterhead, doorway, gateway and host, then node i needs to propagate its new status to its neighbors. Note that the status changes to and from doorway and gateway
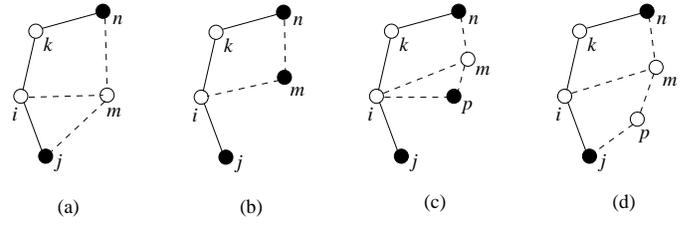

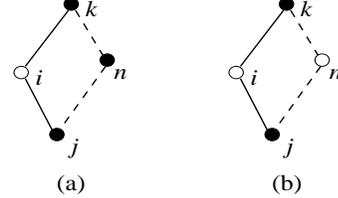**Figure 8: Four cases that may disable node i from becoming a doorway**

Figure 9: Two cases that disable node i from becoming a gateway.

are propagated only to one-hop neighbors, while the status changes from/to clusterheads are required to propagate within two hops, so as to inform other nodes when constructing the CDS. It is the responsibility of a neighbor protocol to propagate the changes in node types to the one- and two-hop neighbors in time.

4.2 CDS Connections

The backbone topology is constructed by using links between the elected clusterheads, doorways and gateways present in the original network topology according to the following rules:

R1: All links between clusterheads are kept in the backbone topology.

R2: The one-hop links from doorways and gateways to the nodes in their respective sets workfor are kept in the backbone topology. Some nodes in the workfor set may be outside of the one-hop neighborhood for doorways, and there are no links from the doorways to these nodes.

The links of the original network topology between gateways or between doorways are not kept in the CDS. Doorways are alway attached to a clusterhead on one end and a gateway on the other end. Gateways are alway attached with clusterheads or doorways.

Figure 10 illustrates the backbone topology construction process using **TMPO** in four steps on a network graph generated by randomly placing 100 nodes over a 1000×1000 square meter area. The radio reception range is 200 meters for all nodes. In Figure 10 (a), every node is a host and network connectivity is very dense in some parts of the network, which increases the overhead of network control functions. In Figure 10 (b), clusterheads are elected, but are disconnected from each other. Hosts are attached to their corresponding clusterheads in their one-hop neighborhood. In Figure 10 (c), gateways are elected. Without adding doorways for extending the coverage of clusterheads, we see that
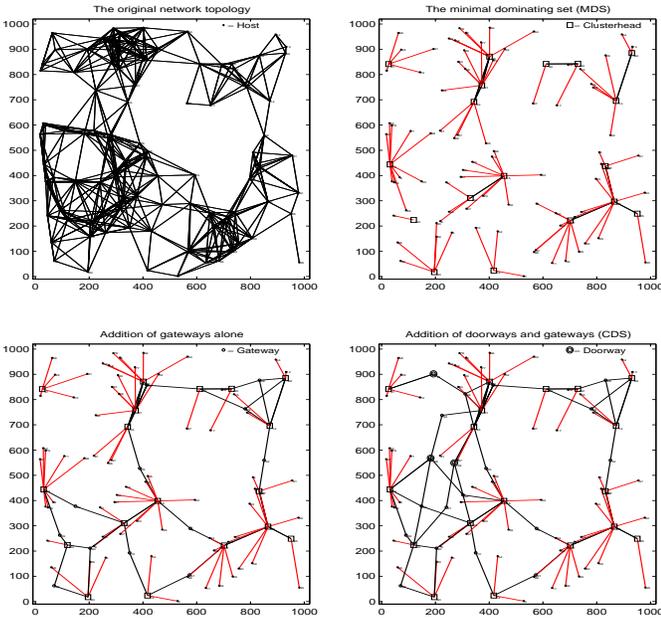
Figure 10: A network topology control example.

adding gateways alone cannot guarantee the connectivity of the backbone topology. Once the doorways are added after the clusterheads election in Figure 10 (d), gateways are again inserted, and the CDS is formed over the original network topology using the CDS construction rules.

The stability of the backbone topology is an important goal in **TMPO**, and is achieved by the following mechanisms. First, the period of willingness adjustment is long to allow enough time to adjust to clusterhead changes. Second, the willingness value is directly related with the speed of nodes. Fast moving nodes get fewer chances to become a clusterhead than slowly moving or static nodes, thus decreasing the possibility of topology changes due to mobility. Third, the willingness value is also related to the remaining energy of the node, so that the clusterhead role is sustained longer and fewer topology changes happen. Forth, except for clusterheads, doorways and gateways are not put in the routing tables built over the CDS, but only provide links between clusterheads. When doorways or gateways fail, there can be other hosts taking over the role, without any routing updates. The transient period between clusterhead connection re-establishment is equal to the delay of the neighbor protocol propagating one-hop neighbor updates. Lastly, nodes change their priorities asynchronously, thus avoiding synchronization problems from clusterhead changes and routing updates.

4.3 Application of Topology Management

The CDS obtained from **TMPO** reduces the topology information at each node with just enough links to maintain network connectivity. Previous research has gone down this path using different topology control algorithms, and applied the derived backbone topology to efficient data forwarding or the provision of QoS services [25, 8].

4.3.1 Efficient Routing

Routing information requires only enough active nodes in

the network for connectivity and data forwarding purposes among hosts and routers. The number of active nodes can be dramatically smaller than the total number of nodes in the network. The CDS of an ad hoc network provides the backbone of the network for this purpose, where clusterheads are responsible for receiving or delivering data packets to hosts under their dominance, while gateways and doorways forward data packet between clusterheads for which they work.

Another application for the CDS problem consists of carrying out reliable broadcasts in ad hoc networks, such that each message from a source node reaches every other network node reliably. Without proper control, broadcast messages may incur very high overhead as each node is exposed to multiple copies of the same message. Utilizing the CDS in the network, only clusterheads need to broadcast, while the intermediate doorways and gateways relay the message between gateways and can use more reliable mechanisms to do so.

4.3.2 Energy Conservation

Energy consumption by nodes in an ad hoc network is due to the data flows generated or forwarded by each node, as well as the signaling overhead of network control protocols. In a network with a flat network topology management, nodes can consume large amounts of energy by maintaining routing information exchanges with every neighbor.

To balance energy consumption as well as to maintain network connectivity, **TMPO** elects a backbone of routers (the CDS) to serve the network routing functionalities based on their energy levels and mobility. Only clusterheads, doorways and gateways need to stay awake continuously. Hosts that are not serving data forwarding can be put in sleeping mode so as to conserve energy and prolong the network lifetime. Routers in the CDS buffer information for sleeping hosts until they wake up and receive the packets. If a backbone router becomes a host and the buffered data are not delivered to the destination host yet, the backbone router can keep holding the data and deliver or delegate them later.

5. CORRECTNESS

THEOREM 1. *The set of clusterheads elected by the algorithm is a dominating set.*

Proof: By the definition of a dominating set, a node is either a dominator itself, or is a one-hop neighbor of a dominator. Because a node either has the highest priority among its one-hop neighbors such that it becomes a dominator itself, or has a neighbor with the highest priority among the one-hop neighbors of the node, which elects the neighbor as a dominator, the network always has a dominating set elected after function **isClusterhead** is called at each node. □

THEOREM 2. *In a dominating set, the maximum distance to another closest clusterhead from any clusterhead is three.*

Proof: We prove the theorem by contradiction. Assume that the maximum distance from a clusterhead a to the closest clusterhead e is four, as illustrated in Figure 11, according to Theorem 1, node c must have been covered by a clusterhead f, which is one hop closer to a than e, thus contradicting the assumption that e is the closest clusterhead to a. □

135
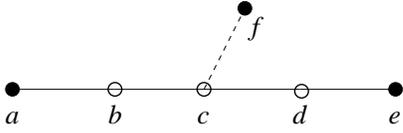
Figure 11: The maximum distance between the closest clusterheads.

THEOREM 3. *Two clusterheads that are within three hops from each other are connected by a path in the CDS.*

Proof: There are three cases to consider for the proof according to the number of hops between the two clusterheads.

1. The two clusterheads are one hop away. In this case, they are directly connected according to rule R1 in the CDS.

2. The two clusterheads are two hops away. In this case, one of the intermediate nodes between the clusterheads either is a clusterhead itself, or becomes a gateway according to the **isGateway** algorithm. Because the link between gateway and clusterhead is kept in the backbone topology by rule R2, the two clusterheads are connected in the CDS.

3. The two clusterheads are three hops away. In this case, if there is any clusterhead on the shortest paths between the two clusterheads, then the connectivity problem is converted to the previous two cases. Otherwise, one of the nodes on the shortest paths has to become a doorway according to function **isDoorway**. Because the doorway is treated like a clusterhead when electing gateways (function **isGateway**), one of the nodes between the newly elected doorway and the other clusterhead must become a gateway. Because the link between the doorway and the clusterhead for which it works is kept in the backbone topology as well as the links from the elected gateway to the doorway and the clusterhead (rule R2), the path between the two clusterheads is preserved in the backbone topology. That is, the two clusterheads are still connected via clusterheads, doorways or gateways in the CDS. □

THEOREM 4. *After **TMPO** terminates, the CDS has the same number of connected components as the original graph.*

Proof: We prove that any pair of clusterheads that are connected in the original graph is still connected via a path in the CDS after **TMPO** terminates.

Suppose that the two clusterheads are v_0 and v_n, and the path between them is $p = v_0 \cdot v_1 \cdot v_2 \cdots v_{n-1} \cdot v_n$ in the original graph. For the endpoints of any link $v_i \cdot v_{i+1}$ on the path p, where $i = 0, 1, \cdots, n-1$, there exist one or two clusterheads that cover node v_i and v_{i+1}. For the case of one clusterhead, the clusterheads of v_i and v_{i+1} are trivially connected. For the case of two clusterheads, the distance between the clusterheads is less than three. From Theorem 3, it follows that the two clusterheads are still connected in the backbone topology. Therefore, there is a path between v_0 and v_n that is composed of clusterheads of the nodes on the path p and other clusterheads, doorways or gateways that connect them. That is, v_0 and v_n are still connected in the CDS. □

6. PERFORMANCE ANALYSIS OF TMPO CLUSTERHEAD ELECTION

Guha and Khuller [14] and Jia *et al.* [16] evaluated the performance of algorithms for constructing dominating sets based on the *performance ratio*, which is the approximate ratio of the cost of a solution derived from an algorithm to the optimal one. In contrast, we evaluate the performance of **TMPO** by the percentage of nodes being elected as clusterheads.

Despite the fact that **TMPO** actually provides a *node-weighted* MDS election algorithm based on the willingness parameter, the performance of the specified algorithm can be evaluated regardless of such weights. Therefore, we consider the case in which all nodes have the same willingness to become a clusterhead, that is, $W_i = 1, \forall i \in V$. Furthermore, we analyze the probability of a node being elected as a clusterhead with the simplifying assumptions that all nodes have the same effective transmission range r to communicate with each other, and that the network is created by uniformly placing an infinite number of nodes on an infinite 2-dimensional plane with average node density ρ.

With the above assumptions, the number of nodes within an area of size S is a random variable following a Poisson distribution as given in Eq. (4).

$$p(k, S) = \frac{(\rho S)^k}{k!} e^{-\rho S} . \tag{4}$$

Because node priorities are evenly distributed over $(0, 1]$ according to Eq. (3), nodes have equal chances to become clusterheads using **TMPO**. That is, the probability of a node winning over k other contenders is $\frac{1}{k+1}$.

For convenience, the variable $T(N)$ and $U(N)$ are introduced to denote two probabilities when the number of contenders k follows a Poisson distribution with mean N. $T(N)$ denotes the probability of a node winning among its contenders. Because the number of contenders follows a Poisson distribution with mean N and all nodes have equal chances of winning, the probability $T(N)$ equals

$$T(N) = \sum_{k=1}^{\infty} \frac{1}{k+1} \frac{N^k}{k!} e^{-N} = \frac{e^N - 1 - N}{N e^N} .$$

Note that k starts from 1 in the expression for $T(N)$, because a node with no contenders does not win at all. $U(N)$ is the probability that a node has at least one contender, which is simply $1 - e^{-N}$.

In addition, N_1 is introduced to denote the average number of one-hop neighbors of a node, which according to the assumptions we have made equals

$$N_1 = \rho \pi r^2$$

As mentioned before, a node i becomes a clusterhead if either of the following two conditions holds:

1. Node i has the highest priority among its one-hop neighbors;

2. Node i does not have the highest priority in its one-hop neighbors, but has the highest priority among the one-hop neighbors of one of i's own one-hop neighbors.

For the first condition, the probability is:

$$p_1 = \sum_{k=0}^{\infty} \frac{N_1^k}{k!} e^{-N_1} \cdot \frac{1}{k+1} = \frac{U(N_1)}{N_1} \ .$$

For the second condition to be satisfied, it must be true that node i has a one-hop neighbor with higher priority, while node i also has the highest priority around one of its one-hop neighbors. Many situations can render node i as the clusterhead in this case. Therefore, we have to consider the lower bound of the probability that node i becomes a clusterhead by considering only a single one-hop neighbor j that makes node i a clusterhead. Under this simplification, the geometric relation between node i and node j is shown in Figure 12, and the distance between them is denoted by tr.
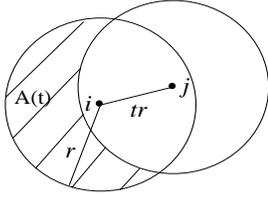

Figure 12: Clusterhead election.

Accordingly, we need to compute the probability of node i having the highest priority within the one-hop neighbors of node j, and the probability of node i having lower priority in the portion of its one-hop neighborhood outside of node j's coverage (the shaded area in Figure 12).

Denote the number of nodes in the shaded area by $A(t)$. We have

$$A(t) = 2\rho r^2 \left[\frac{\pi}{2} - a(t) \right] \ ,$$

where $a(t) = \arccos \frac{t}{2} - \frac{t}{2}\sqrt{1 - \left(\frac{t}{2}\right)^2}$. Therefore, the probability of node i having a lower priority than the nodes in the shaded area is

$$p_2 = \sum_{k=1}^{\infty} \frac{A(t)^k}{k!} e^{-A(t)} \frac{k}{k+1} = U(A(t)) - T(A(t)) \ .$$

In addition, node i should have the highest priority among node j's one-hop neighborhood, of which the probability is:

$$p_3 = \sum_{k=0}^{\infty} \frac{N_1^k}{k!} e^{-N_1} \frac{1}{k+2} = \frac{N_1 - 1}{N_1} \cdot T(N_1) + e^{-N_1} \ .$$

Because the probability density function of parameter t is $p(t) = 2t$, the probability that node i becomes a clusterhead can be obtained by multiplying the above two probabilities and integrating over the range $t \in (0,1]$:

$$p_4 = \int_0^1 p_2 \cdot p_3 \cdot 2t\,dt = \left(\frac{N_1 - 1}{N_1} \cdot T(N_1) + e^{-N_1} \right)$$
$$\cdot \int_0^1 [U(A(t)) - T(A(t))]2t\,dt \ .$$

Because the two conditions are mutually exclusive, the probability of node i becoming a clusterhead is thus:

$$p_{ch} = p_1 + (1 - p_1) \cdot p_4$$
$$= \frac{U(N_1)}{N_1} + \left(1 - \frac{U(N_1)}{N_1}\right)\left(\frac{N_1 - 1}{N_1} \cdot T(N_1) + e^{-N_1}\right)$$
$$\cdot \int_0^1 [U(A(t)) - T(A(t))]2t\,dt \ .$$

Nodes are homogeneous in the randomly generated network with regard to their priority generations and one-hop neighbor information. Therefore, the probability of becoming a clusterhead is also the same for all nodes. Given an area with N nodes in an infinitely large network with uniform node density, the expected size of the MDS in the area is:

$$|MDS_N| = N \cdot p_{ch} \ . \tag{5}$$

To validate the result in Eq. (5) with the performance of function **isClusterhead**, we randomly created a number of networks by placing 100 nodes onto a 1000×1000 square meter plane. The opposite sides of the square are seamed together so as to emulate the infinite plane. All nodes have the same transmission range, which increases from 1 to 400 meters in the individual experiment setting so as to evaluate the performance of **TMPO** at different node densities.

A near-optimum MDS election is also carried out for comparison purposes in the same network topology. The near-optimum election algorithm is based on the aforementioned **Max Degree** algorithm, eliminating redundant clusterheads in the MDS.
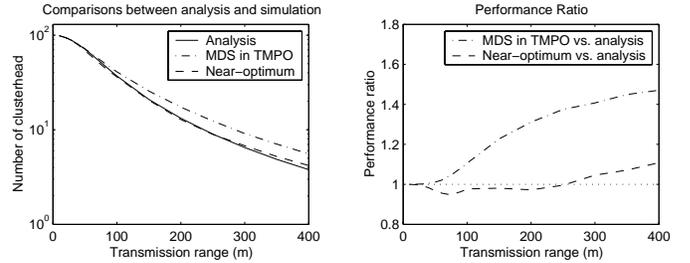

Figure 13: Comparison between theoretical analyses and simulations.

Because we ran the experiments in different network scenarios for many times, and recomputed the MDS at each experiment setting, we achieve the average performances of both algorithms. The analytical results and the results from the experiments of **TMPO** and the near-optimum MDS algorithm are shown in Figure 13. As the results in the left portion of Figure 13 illustrate, the number of elected clusterheads drops quickly as the node transmission range increases, because clusterheads have larger and larger coverage.

The performance ratios of **TMPO** *vs.* the analytical model, and the near-optimum MDS algorithm *vs.* the analytical model are shown in the right portion of Figure 13. It appears that the near-optimum MDS algorithm performs close to the lower-bound of the MDS, while the performance of **TMPO** digresses when the node transmission range increases. The performance ratio between **TMPO** and the analytical results offsets from 1 by as far as 48%, and is due to the simplifications made in our analytical model.

7. SIMULATIONS

We compare the performance of **TMPO** with the optimum topology management algorithm and four other topology management algorithms based on different heuristics using simulations. To establish a fair comparison among these algorithms, some MDS stability optimizations and clusterhead negotiation procedures presented in the original papers are omitted. The algorithms differ from one another only in the clusterhead election process, and use the same procedure to connect clusterheads using doorways and gateways to form the CDS, which does not need negotiation packets and actually improves the performance of the original algorithms based on other heuristics. The following five schemes are compared with **TMPO**:

- **OPTIMUM**: This is a near-optimum approach that uses global topology information, the MDS is constructed by selecting nodes with the highest degree one by one until all nodes outside the MDS are covered by the MDS. Individual nodes with low degree in the MDS are inspected and eliminated from the MDS if the node and its dominated nodes are covered by other clusterheads in the MDS.

- **Lowest ID** [1, 3, 11, 13, 19]: In this approach, the node identifier is used to elect MDS members. A node is elected into the MDS if it has the lowest identifier in the one-hop neighborhood of itself or one of its one-hop neighbors.

- **Max Degree** [14, 16, 25]: In this approach, a node is elected into the MDS if it has the highest degree in the one-hop neighborhood of itself or one of its one-hop neighbors.

- **MOBIC** [7]: In this approach, each node computes a mobility metric based on the received-signal strength variations from its one-hop neighbors. A node becomes a clusterhead if it has the lowest mobility metric in the one-hop neighborhood of itself or one of its one-hop neighbors.

- **Load Balance** [2]: This approach is similar to **Lowest ID** except that it is based on a virtual identifier (VID) assigned to each node. The VID of a node increases every time slot if the node is a host, or remains constant if the node is a clusterhead. Each clusterhead runs a budget that decreases every time slot. A clusterhead resets its VID to 0 and returns to host status when the budget runs out, thus providing load balancing between network nodes.

The simulations are carried out in ad hoc networks generated over a 1000×1000 square meter area with 100 nodes moving in random directions at random speeds. In order to simulate infinite plane, the opposite sides of the area is seamed together so that the simulation plane forms a torus. To mobility scenarios are simulated, of which the speed is selected from 0 to 5 meters/second in low mobility scenarios, or from 0 to 50 meters/second in high mobility scenarios. In each mobility scenario, the radio transmission range is set at different values in each simulation, chosen from 100 to 500 meters, so as to demonstrate the effects of the one-hop neighborhood density in the clusterhead elections.

In **TMPO**, the node priority recomputation period is one minute.

Different types of nodes consume energy at different rates. We ignore the energy consumed due to local computations, but assume that the energy consumption rate is only dependent on the type of the node. A host consumes 0.6% of the total energy per minute in these algorithms, a clusterhead consumes 3%, and a doorway or a gateway consumes 2.4%. Every node starts with an energy level of 1 at the beginning of each simulation.

The algorithms are compared using six different metrics, and the results are shown in Figures 14-16:
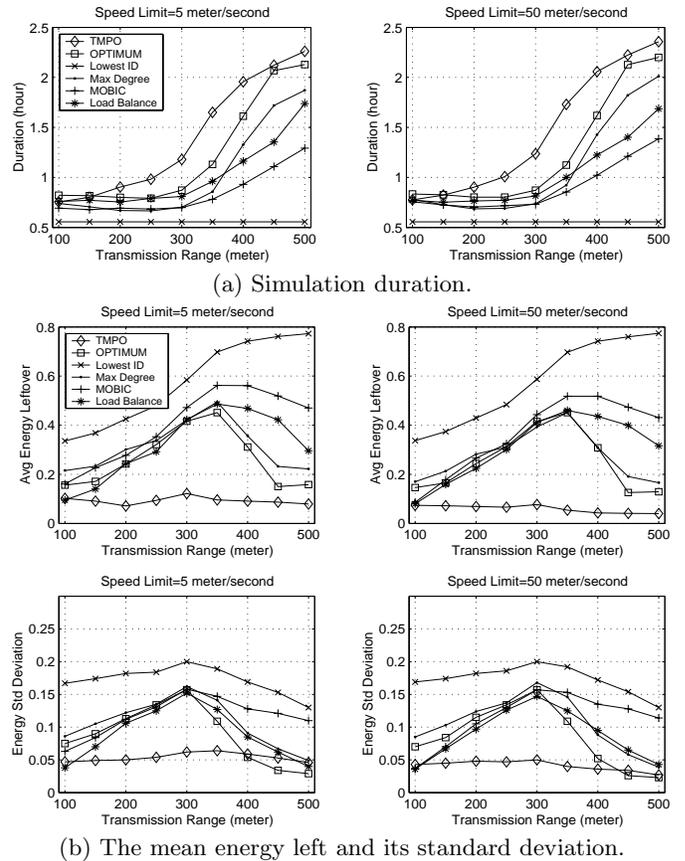

(a) Simulation duration.

(b) The mean energy left and its standard deviation.

**Figure 14: Simulation duration and energy left.**

The simulations stop once any node runs out of energy in the network. The *simulation duration* (Figure 14(a)) measures the load balancing capability of the heuristics to prolong the system lifetime by rotating the clusterhead roles between network nodes. **MOBIC** and **Lowest ID** perform the worst, because the clusterheads are mostly fixed over certain nodes throughout the simulations. In **Lowest ID**, the fact that the node with the lowest identifier is always in the MDS terminates the simulations in fixed time. **TMPO** is one of the best heuristics.

The mean and the standard deviation of the energy left per node when the simulation is over (Figure 14 (b)) indicate the load balancing capability of the heuristics. After each simulation, **TMPO** leaves the network nodes with the least energy and the lowest standard deviation because of its energy-awareness when selecting the MDS, while **Low-**

est ID performs the worst because it always has nodes that are always or never elected to the MDS. The curves peak at transmission range 300- or 350-meter because the two-hop neighborhood of each node begins to overlap in the opposite directions on the plain, which renders less clusterheads and more opportunities to rotate clusterhead roles for some heuristics.
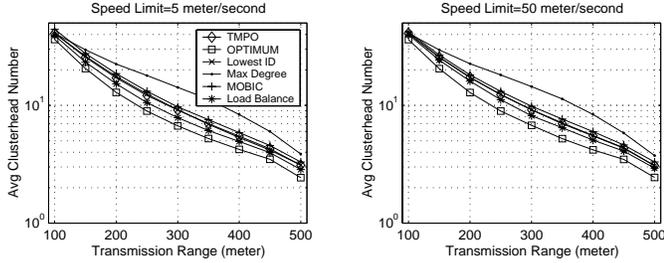
(a) Average number of clusterheads.
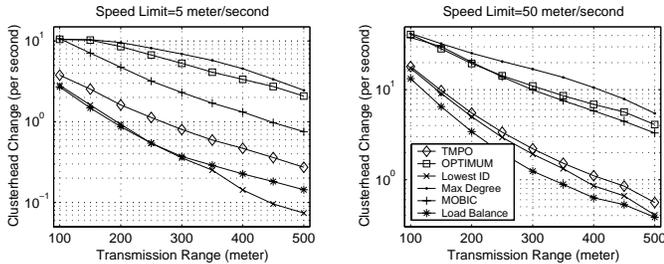
(b) Clusterhead change rate.
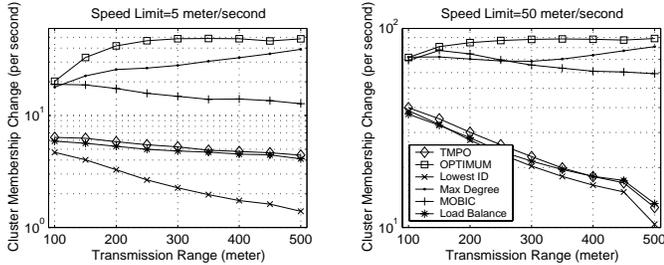
(c) Cluster membership change rate.

Figure 15: Statistics about clusterheads.

The *average number of clusterheads* (Figure 15 (a)) is measured each time slot when clusterhead recomputation happens. As Figure 15 (a) shows, all heuristics perform almost identically, which suggests that the MDS cardinality can hardly prove the advantage of topology management algorithms. It is the topology stability and load-balancing features of the algorithms that make the difference.

The *clusterhead change rate* (Figure 15 (b)) measures the stability of the MDS in a mobile network. **TMPO**, **Lowest ID** and **Load Balance** are the best performing, because they depend on relatively static attributes for clusterhead election, such as node identifiers and priorities which change less frequently than node locations. **OPTIMUM**, **Max Degree** and **MOBIC** perform the worst because the MDS elections depend on the volatile mobility and network topology of ad hoc networks.

The *cluster membership change rate* (Figure 15 (c)) measures the stability of network connections in the presence

of mobility. The cluster membership change rates align to the clusterhead change rates of the examined heuristics. **TMPO**, **Lowest ID** and **Load Balance** still perform the best in both high and low mobility scenarios.
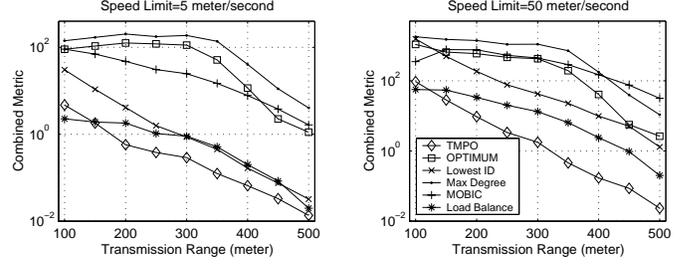

Figure 16: Combined evaluations.

Given the above metrics, it is not easy to see the advantages of different heuristics. The *combined metric* (Figure 16) is the product of the average energy, its standard deviation, the average number of clusterheads, the clusterhead change rate and the cluster membership change rate of the respective simulations. Although it is meaningless to simply multiply several independent metrics, the product fairly compares the overall performance if the individual metrics are equally important. The lower the combined metric of a heuristic, the better the heuristic performs in terms of the clusterhead load-balancing capability and the MDS/CDS stability. As shown in Figure 16, **TMPO** performs near, if not always, the best among all heuristics in both low mobility and high mobility scenarios. **Load balance** is the second best in general. The simulations favored **Load balance**, because it assumes that all nodes start with the same energy level [2], which is the setup of the simulations.

Overall, when the mobility increases from 5 meter/second to 50 meters per second, **TMPO** shows better load balancing capability (Figure 14) and higher topology maintenance stability (Figure 16).

8. CONCLUSION

We have presented **TMPO**, a novel energy-aware topology management approach based on dynamic node priorities in ad hoc networks. **TMPO** consists of two parts that implement the MDS and CDS elections, respectively. **TMPO** builds a stable and energy-aware CDS from the MDS to simplify the topology information for sufficient network connectivity and efficient data communication. Compared to five prior heuristics of MDS and CDS elections in ad hoc networks, **TMPO** offers four key advantages. First, **TMPO** obtains the MDS and CDS of the network without any negotiation stage; only two-hop neighbor information is needed. Second, **TMPO** allows nodes in the network to periodically recompute their priorities, so as to balance the clusterhead role and prolong the battery life of each node. Third, **TMPO** introduces the willingness value of a node, which decides the probability of the node being elected into the MDS according to the battery life and mobility of the node. Fourth, **TMPO** introduces doorway concept for the CDS in addition to the well-known gateway and clusterhead concepts.

A key contribution of this work consists of converting the

static attributes of a node, such as node identifier, into a dynamic control mechanism that incorporates the three key factors for topology management in ad hoc networks — the nodal battery life, mobility, and load balancing. Although existing proposals have addressed all these aspects, **TMPO** constitutes a more comprehensive approach. **TMPO** is unique in that the election of the MDS is locally determined, without the need for any negotiation phase.

9. REFERENCES

[1] A. Amis, R. Prakash, T. Vuong, and D.T. Huynh. MaxMin D-Cluster Formation in Wireless Ad Hoc Networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Mar. 1999.

[2] A.D. Amis and R. Prakash. Load-balancing clusters in wireless ad hoc networks. In *Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pages 25–32, Los Alamitos, CA, Mar. 24-25 2000.

[3] D.J. Baker and A. Ephremides. The architectural organization of a mobile radio network via a distributed algorithm. *IEEE Transactions on Communications*, COM-29(11):1694–701, Nov. 1981.

[4] S. Banerjee and S. Khuller. A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Anchorage, Alaska, Apr. 2001.

[5] L. Bao and J.J. Garcia-Luna-Aceves. A New Approach to Channel Access Scheduling for Ad Hoc Networks. In *Proc. ACM Seventh Annual International Conference on Mobile Computing and networking*, Rome, Italy, Jul. 16-21 2001.

[6] L. Bao and J.J. Garcia-Luna-Aceves. Transmission Scheduling in Ad Hoc Networks with Directional Antennas. In *Proc. ACM Eighth Annual International Conference on Mobile Computing and networking*, Atlanta, Georgia, USA, Sep. 23-28 2002.

[7] P. Basu, N. Khan, and T. D.C. Little. A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks. In *International Workshop on Wireless Networks and Mobile Computing (WNMC2001)*, Scottsdale, Arizona, Apr. 16-19 2001.

[8] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In *Proc. 7th ACM MOBICOM*, Rome, Italy, Jul. 2001.

[9] C.C. Chiang, H.K. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *IEEE Singapore International Conference on Networks SICON'97*, pages 197–211, Singapore, Apr. 14-17 1997.

[10] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, a. Qayyum, and L. Viennot. Optimized Link State Routing Protocol. In *IEEE INMIC*, Pakistan, 2001.

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

[12] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* Freeman, Oxford, UK, 1979.

[13] M. Gerla and J.T.C. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–65, 1995.

[14] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20, (no.4):374–87, Apr. 1998. Springer-Verlag.

[15] L. Hu. Topology control for multihop packet radio networks. *IEEE Transactions on Communications*, 41(10):1474–81, Oct. 1993.

[16] L. Jia, R. Rajaraman, and T. Suel. An Efficient Distributed Algorithm for Constructing Small Dominating Sets. In *Twentieth ACM Symposium on Principles of Distributed Computing PODC'01*, Newport, Rhode Island, Aug. 26-29 2001.

[17] P. Krishna, N. Vaidya, M. Chatterjee, and D. Pradhan. A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM Computer Communication Review*, pages 49–65, Apr. 1997.

[18] L. Li, V. Bahl, Y.M. Wang, and R. Wattenhofer. Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2001.

[19] C.R. Lin and M. Gerla. Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–75, Sep. 1997.

[20] S. Narayanaswamy, V. Kawadia, R. S. Sreenivas, and P. R. Kumar. Power Control in Ad-Hoc Networks: Theory, Architecture, Algorithm and Implementation of the COMPOW Protocol. In *Proceedings of the European Wireless Conference – Next Generation Wireless Networks: Technologies, Protocols, Services and Applications*, pages 156–162, Florence, Italy, Feb. 25-28 2002.

[21] R. Prakash. Unidirectional links prove costly in Wireless Ad-Hoc Networks. In *Proceedings of the Discrete Algorithms and Methods for Mobile Computing and Communications - DialM*, Seattle, WA, Aug. 20 1999.

[22] R. Ramanathan and R. Rosales-Hain. Topology Control of Multihop Wireless Networks using Transmit Power Adjustment. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, page N.A. IEEE, Mar. 26-30 2000.

[23] R. Ramanathan and M. Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications*, 3(1):101–19, 1998.

[24] P. Sinha, R. Sivakumar, and V. Bharghavan. Enhancing ad hoc routing with dynamic virtual infrastructures. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 1763–72, Anchorage, AK, USA, Apr. 22-26 2001.

[25] R. Sivakumar, P. Sinha, and V. Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. *IEEE Journal on Selected Areas in Communications*, 17(8):1454–65, Aug. 1999.

[26] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):246–57, Mar. 1984.