7-1998

Distributed and Mobility-Adaptive Clustering for Ad Hoc Networks

Stefano Basagni

Distributed and Mobility-Adaptive Clustering for Ad Hoc Networks

Stefano Basagni
Department of Electrical Engineering
The University of Texas at Dallas

Technical Report UTD/EE-02-98
July 1998

Distributed and Mobility-Adaptive Clustering for Ad Hoc Networks*
Technical Report UTD/EE-02-98
July 1998

Stefano Basagni

Erik Jonsson School of Engineering and Computer Science MC 33
The University of Texas at Dallas
2601 N. Floyd Rd.
Richardson, TX 75083
Tel. 972 883 2216 Fax 972 883 2710
E-mail: basagni@utdallas.edu

Abstract

A distributed algorithm is presented that partitions the nodes of a fully mobile network (*ad hoc* network) into *clusters*, thus giving the network a hierarchical organization. The algorithm is proven to be adaptive to changes in the network topology due to nodes' mobility and to nodes addition/removal. A new weight-based mechanism is introduced for the efficient cluster formation/maintenance that allows the cluster organization to be configured for specific applications and adaptive to changes in the network status, not available in previous solutions. Simulation results are provided that demonstrate up to an 85% reduction on the communication overhead associated with the cluster maintenance with respect to clustering algorithms previously proposed.

Keywords: Wireless Mobile Networks, Distributed Network Algorithms, Clustering, *Ad Hoc* Networks.

1 Introduction

Obtaining a hierarchical organization of a network is a well-known and studied problem of distributed computing. It has been proven effective in the solution of several problems, such as, minimizing the amount of storage for communication information (e.g., routing and multicast tables), thus reducing information update overhead, optimizing the use of the network bandwidth, distributing resources throughout the network, etc.

In the case of *ad hoc networks*, i.e., wireless networks in which possibly all nodes can be mobile, partitioning the nodes into groups (*clusters*) is similarly important. In addition, *clustering* is crucial for controlling the spatial reuse of the shared channel (e.g., in terms of time division or frequency division schemes), for minimizing the amount of data to be exchanged in order to maintain routing and control information in a mobile environment, as well as for building and maintaining cluster-based *virtual* network architectures.

The notion of cluster organization has been used for ad hoc networks since their appearance. In [1, 2], a "fully distributed linked cluster architecture" is introduced mainly for hierarchical routing and to demonstrate the adaptability of the network to connectivity changes. With the advent

*This work was supported in part by the Army Research Office under contract No. DAAG55-96-1-0382.

of multimedia communications, the use of the cluster architecture for ad hoc network has been revisited by Gerla et. al. [3, 5, 4]. In these latter works the emphasis is toward the allocation of resources, namely, bandwidth and channel, to support multimedia traffic in an ad hoc environment.

In existing solutions for *clustering* of ad hoc networks, the clustering is performed in two phases: clustering *set up* and clustering *maintenance*. The first phase is accomplished by choosing some nodes that act as coordinators of the clustering process (*clusterheads*). Then a *cluster* is formed by associating a clusterhead with some of its *neighbors* (i.e., nodes within the clusterhead's transmission range) that become the *ordinary nodes* of the cluster. Once the cluster is formed, the clusterhead can continue to be the local coordinator for the operations of its cluster (as in [1, 2, 4]), or, in order to avoid bottlenecks, the control can be distributed among the nodes of the cluster [3, 5]. The existing clustering algorithms differ on the criterium for the selection of the clusterheads. For example, in [1, 2, 5] the choice of the clusterheads is based on the unique identifier (ID) associated to each node: the node with the lowest ID is selected as clusterhead, then the cluster is formed by that node and all its neighbors. The same procedure is repeated among the remaining nodes, until each node is assigned to a cluster. When the choice is based on the maximum *degree* (i.e., the maximum number of neighbors) of the nodes, the algorithm described in [4] is obtained. A common assumption for the clustering set up is that the nodes *do not move* while the cluster formation is in progress. This is a major drawback, since in real ad hoc situations, no assumptions can be made on the mobility of the nodes.

Once the nodes are partitioned into clusters, the non mobility assumption is released, and techniques are described on how to maintain the cluster organization in the presence of mobility (clustering maintenance). For instance, in [1] a reorganization of the clusters, due to node mobility, is done periodically, just invoking the clustering process again (as at set up time). Of course, during the re-clustering process the network cannot rely on the cluster organization. Thus, this is a feasible solution only when the network does not need too many reorganizations. In [5] cluster maintenance in the presence of mobility is described where each node v decides locally whether to update its cluster or not. This decision is based on the knowledge of the v's one *and two* hop neighbors and on the knowledge of the local topology of v's neighbor with the *largest degree*. The resulting mobility-adaptive algorithm is thus different from the one used for the cluster formation (based on the nodes' IDs and on the knowledge of their one hop neighbors) and the obtained clustering has different properties with respect to the initial one.

In this paper we present a clustering algorithm suitable for both the clustering set up and maintenance that aims to overcome the above mentioned mobility related limitations. For our *Mobility-Adaptive Clustering Algorithm* (MACA, for short) we obtain the following properties, not available in previous solutions:

- Nodes can move, even during the clustering set up: MACA is *adaptive* to the changes in the topology of the network, due to the mobility of the nodes or to node addition and/or removal.
- MACA is fully *distributed*. A node decides its own role (i.e., clusterhead or ordinary node) solely knowing its current *one* hop neighbors.
- Every ordinary node always has *direct access* to at least one clusterhead. Thus, the nodes in the cluster are at most two hops apart. This guarantees fast intra-cluster communication and fast inter-cluster exchange of information between any pair of nodes.
- MACA uses a *general* mechanism for the *selection of the clusterheads*. The choice is now based on generic *weights* associated with the nodes.

- The number of clusterheads that are allowed to be neighbors is a parameter of the algorithm (*degree of independence*).
- Finally, we define for MACA a new weight-based criterium that allows the nodes to decide whether to change (switch) its role or not depending on the current condition of the network, thus minimizing the overhead due to the switching process.

Notice that the last three properties introduce the possibility for MACA to be configured for the specific ad hoc network in which it operates. For example, the weight associated to a node allows us to express how that node is suitable for the role of clusterhead depending on properties of the node itself, such as its speed, its transmission power, etc. Moreover, previous solutions would not allow two clusterheads to be neighbors, forcing one of the two to resign. This is not always necessary, and two or more clusterheads should be allowed to be neighbors depending on specific network conditions and applications. The same reasoning applies when an ordinary node becomes a neighbor of another clusterhead: the criterium with which it chooses whether to affiliate with the new neighboring clusterhead or not should depend again on the current conditions of the network, and on the specific applications that use the cluster organization. We demonstrate the advantage of our new weight-based setting by presenting simulation results that compare MACA with the "lowest ID first" algorithm of [5]. We show up to an 85% reduction on the communication overhead associated with cluster maintenance.

The paper is organized as follows. In the next section we define a model for ad hoc network and the clustering we want to obtain for these networks. The distributed clustering algorithm which adapts to changes in the network topology (MACA) is then introduced and proved correct in Section 3. The paper concludes with some simulation results. The detailed codes of the procedures that implement MACA can be found in Appendix A.

2 Preliminaries and problem definition

We model an *ad hoc* network by an undirected graph $G = (V, E)$ in which V, $|V| = n$, is the set of (wireless) nodes and there is an edge $\{u, v\} \in E$ if and only if u and v can mutually receive each others' transmission. In this case we say that u and v are neighbors. The set of the neighbors of a node $v \in V$ will be denoted by $\Gamma(v)$. Due to mobility, the graph can change in time.

Every node v in the network is assigned a unique identifier (ID). For simplicity, here we identify each node with its ID and we denote both with v. Finally, we consider weighted networks, i.e., a weight w_v (a real number ≥ 0) is assigned to each node $v \in V$ of the network. For the sake of simplicity, in this paper we stipulate that each node has a different weight.

Clustering an ad hoc network means partitioning its nodes into *clusters*, each one with a *clusterhead* and (possibly) some *ordinary nodes*. The choice of the clusterheads is here based on the *weight* associated to each node: the bigger the weight of a node, the better that node for the role of clusterhead. The process of cluster formation/maintenance is continuously executed at each node, and each node decides its own role so that the following three requirements (that we call "ad hoc clustering properties") are satisfied:

1. Every ordinary node always affiliates with one clusterhead.
2. For every ordinary node v there is no clusterhead $u \in \Gamma(v)$ such that $w_u > w_{Clusterhead} + h$, where *Clusterhead* indicates the current clusterhead of v.
3. A clusterhead cannot have more than k neighboring clusterheads.

Requirement number 1. ensures that each ordinary node has direct access to at least one clusterhead (the one of the cluster to which it belongs), thus allowing fast intra- and inter-cluster communications. The second requirement guarantees that each ordinary node always stays with a clusterhead that can give it a "guaranteed good" service. By varying the threshold parameter h it is possible to reduce the communication overhead associated to the passage of an ordinary node from its current clusterhead to a new neighboring one when it is not necessary. Finally, requirement number 3. allows us to have the number of clusterheads that can be neighbors as a parameter of the algorithm. This, as seen for requirement number 2., and as will be demonstrated by the use of simulations, allows us to consistently reduce the communication overhead due to the change of role of nodes.

3 A Mobility-Adaptive Clustering Algorithm (MACA)

In this section we describe a distributed algorithm for the set up and the maintenance of a cluster organization in the presence of nodes' mobility that satisfies the three requirements listed in the previous section.
We start by making the following two common assumptions:

- A message sent by a node is received correctly within a finite time (a *step*) by all its neighbors.
- Each node knows its own ID, its weight, its role (if it has already decided its role: either a clusterhead or an ordinary node) and the ID, the weight and the role of all its neighbors (if they have already decided their role). When a node has not yet decided what its role is going to be, it is considered as an ordinary node.

The algorithm is executed at each node in such a way that at a certain time a node v decides to change) its role. *This decision is entirely based on the decision of the nodes* $u \in \Gamma(v)$ *such that* $w_u > w_v$.

Except for the initial procedure, the algorithm is message driven: a specific procedure will be executed at a node depending on the reception of the corresponding message. We use three types of messages that are exchanged among the nodes: CH(v), used by a node $v \in V$ to make its neighbors aware that it is going to be a clusterhead, JOIN(v,u), with which a node v communicates to its neighbors that it will be part of the cluster whose clusterhead is node $u \in \Gamma(v)$, $v,u \in V$, and RESIGN(w) that notifies a clusterhead whose weight is $\leq w$ that it has to resign its role. In the following discussion, and in the procedures (see Appendix A), we use the following notation:

- v, the generic node executing the algorithm (from now on we will assume that v encodes not only the node's ID but also its weight w_v);
- *Cluster*(v), the set of nodes in v's cluster. It is initialized to $\emptyset$, and it is updated only if v is a clusterhead;
- *Clusterhead*, the variable in which every node records the (ID of the) clusterhead that it joins. It is initialized to **nil**;
- *Ch*$(-)$, boolean variables. Node v sets *Ch*(u), $u \in \{v\} \cup \Gamma(v)$, to **true** when either it sends a CH(v) message ($v = u$) or it receives a CH(u) message from u ($u \neq v, u \in \Gamma(v)$).

Furthermore:

- Every node is made aware of the failure of a link, or of the presence of a new link by a service of a lower level (this will trigger the execution of the corresponding procedure);
- The procedures of MACA (M-procedures, for short) are "atomic," i.e., they are not interruptible;
- At clustering set up or when a node is added to the network its variables *Clusterhead*, *Ch*$(-)$, and *Cluster*$(-)$ are initialized to **nil**, **false** and $\emptyset$, respectively.

The following two rules define how the nodes assume/change their roles adapting to changes in the network topology.

1. Each time a node v moves into the neighborhood of a clusterhead u with a bigger weight, we require that v switches to u's cluster only if $w_u > w_{Clusterhead} + h$, where *Clusterhead* is the clusterhead of v (it can be v itself) and h is a real number ≥ 0. This should happen independently of the current role of v. With this rule we want to model the fact that we incur the switching overhead only when it is really convenient. When $h = 0$ we simply obtain that each ordinary nodes affiliates with the neighboring clusterhead with the biggest weight.
2. We allow a clusterhead v to have up to k neighboring clusterheads, $0 \leq k < n$. We call this condition the k-neighborhood condition. Choosing $k = 0$ we obtain that no two clusterheads can be neighbors (maximum degree of independence).

The parameters h and k can be different from node to node, and they can vary in time. This allows MACA to self-configure dynamically in order to meet the specific needs of upper layer applications/protocols that requires an underlying clustering organization. At the same time, different values of h and k allow our algorithm to take into account dynamically changing network conditions, such as the network connectivity (related to the average nodal degree, i.e., to the average number of the neighbors of the nodes), variations in the rate of the mobility of the nodes, etc.

The following is the informal description of the six M-procedures. The details can be found in Appendix A.

• *Init.* At the clustering set up, or when a node v is added to the network, it executes the procedure *Init* in order to determine its own role. If among its neighbors there is at least a clusterhead with bigger weight, then v will join it. Otherwise it will be a clusterhead. In this case, the new clusterhead v has to check the number of its neighbors that are already clusterheads. If they exceed k, then a RESIGN message is also transmitted, bearing the weight of the first clusterhead (namely, the one with the $(k+1)$th biggest weight) that violates the k-neighborhood condition (this weight is determined by the operator $\min_k$—see the code of *Init* in Appendix A). On receiving a message RESIGN(w), every clusterhead u such that $w_u \leq w$ will resign. Notice that a neighbor with a bigger weight that has not decided its role yet (this may happen at the clustering set up, or when two or more nodes are added to the network at the same time), will eventually send a message (every node executes the *Init* procedure). If this message is a CH message, then v could possibly resign (after receiving the corresponding RESIGN message) or affiliate with the new clusterhead.

• *Link_failure.* Whenever made aware of the failure of the link with a node u, node v checks if its own role is clusterhead and if u used to belong to its cluster. If this is the case, v removes u from *Cluster*(v). If v is an ordinary node, and u was its own clusterhead, then it is necessary to determine a new role for v. To this aim, v checks if there exists at least a clusterhead $z \in \Gamma(v)$ such that $w_z > w_v$. If this is the case, then v joins the clusterhead with the bigger weight, otherwise it becomes a clusterhead. As in the case of the *Init* procedure, a test on the number of the neighboring clusterheads is now needed, with the possible resigning of some of them.

• *New_link.* When node v is made aware of the presence of a new neighbor u, it checks if u is a clusterhead. If this is the case, and if w_u is bigger than the weight of v's current clusterhead plus the threshold h, than, independently of its own role, v affiliates with u. Otherwise, if v itself is a clusterhead, and the number of its current neighboring clusterheads is $> k$ then the operator $\min_k$ is used to determine the weight of the clusterhead x that violates the k-neighborhood condition. If $w_v > w_x$ then node x has to resign, otherwise, if no clusterhead x exists with a weight smaller than v's weight, v can no longer be a clusterhead, and it will join the neighboring clusterhead with the biggest weight.

• *On receiving* CH(u). When a neighbor u becomes a clusterhead, on receiving the corresponding CH message, node v checks if it has to affiliate with u, i.e., it checks whether w_u is bigger than the weight of v's clusterhead plus the threshold h or not. In this case, independently of its current role, v joins u's cluster. Otherwise, if v is a clusterhead with more than k neighbors which are clusterheads, as in the case of a new link, the weight of the clusterhead x that violates the k-neighborhood condition is determined, and correspondingly the clusterhead with the smallest weight will resign.

• *On receiving* JOIN(u,z). On receiving the message JOIN(u,z), the behavior of node v depends on whether it is a clusterhead or not. In the affirmative, v has to check if either u is joining its cluster ($z = v$: in this case, u is added to *Cluster*(v)) or if u belonged to its cluster and is now joining another cluster ($z \neq v$: in this case, u is removed from *Cluster*(v)). If v is not a clusterhead, it has to check if u was its clusterhead. Only if this is the case, v has to decide its role: It will join the biggest clusterhead x in its neighborhood such that $w_x > w_v$ if such a node exists. Otherwise, it will be a clusterhead. In this latter case, if the k-neighborhood condition is violated, a RESIGN message is transmitted in order for the clusterhead with the smallest weight in v's neighborhood to resign.

• *On receiving* RESIGN(w). On receiving the message RESIGN(w), node v checks if its weight is $\leq w$. In this case, it has to resign and it will join the neighboring clusterhead with the biggest weight. Notice that since the M-procedures are supposed to be not interruptible, and since v could have resigned already, it has also to check if it is still a clusterhead.

We conclude this section by showing that by using the M-procedures we obtain and maintain for any ad hoc network a clustering that always satisfies the "ad hoc clustering properties" listed in Section 2.

Theorem 1 *Using the M-procedures, any ad hoc network is clustered in such a way that the ad hoc clustering properties are always satisfied.*

Proof We start by noticing that it is easy to check from the code of the *Init* procedure (see Appendix A) that as soon as a node has executed this procedure, it is always assigned a role which is consistent with the clustering properties. We proceed by showing that by executing the M-procedures in reaction to changes in the network topology, the nodes assume/change their roles so that the ad hoc clustering properties are always satisfied. (The proof refers directly to the code of the procedures given in Appendix A.)

1. That each ordinary node v does not affiliate with more that one clusterhead is evident by noticing that anytime it sends a JOIN(v, u) message its variable *Clusterhead* is initialized (only) to u. That v affiliates with at least a clusterhead derives from the fact that when it has to decide its role and there are clusterheads with bigger weights among its neighbors, or when a switch to another cluster is required, the ordinary node v always looks for the clusterhead with the biggest weight and affiliates with it. This can be easily checked in the codes of: the *Init* procedure (**then** branch of the outermost **if**); the *Link_Failure* and the

On receiving JOIN(u, z) procedures (when the link with its clusterhead u is broken, or the clusterhead u has resigned, joining another node, v looks for another clusterhead; of course, if no clusterhead is available, it will be a clusterhead), and the *New_Link* and the On receiving CH(u) procedures (if u is the new clusterhead on the block, if node v needs to affiliate with u, it does so by executing the **then** branch of the outermost **if**). Thus, there is no case in which an ordinary node v remains without a clusterhead.

2. At the clustering set up, or when a node is added to the (already clustered) network, or when its current clusterhead either resigns or moves away, or, finally, when it is forced to resign, a node always affiliates with the clusterhead with the biggest weight (if there is no such clusterhead, it will become a clusterhead itself), so that the second ad hoc clustering property is always satisfied (see the code of procedures *Init*, *Link_Failure*, On receiving JOIN and On receiving RESIGN). The other cases to consider are when an ordinary node v switches from a cluster to another, or when v is a clusterhead that resigns to join the cluster of a new neighboring clusterhead. In these cases, the second property is guaranteed by the procedures *New_Link* and On Receiving CH, where node v switches to u's cluster only if $w_u > w_{Clusterhead} + h$.

3. Each time a node becomes a clusterhead, i.e., it transmits a CH message (see the **else** branch of the outermost if in the *Init* procedure, the **else** branch of the innermost if in the *Link_Failure* procedure, and the same branch in the On receiving JOIN procedure), it does so because there is no other neighboring clusterhead with which it can affiliate. In this case, it always checks if it has more than k neighboring clusterheads, and if this is the case, it decides (on a weight basis) the clusterheads that have to resign. When these clusterheads receive the corresponding message RESIGN, they have no choice but to resign, joining the clusterhead with the biggest weight around them. The other cases that remain to be checked are when either a clusterhead v has one of its neighbors that becomes a clusterhead, or a clusterhead moves into its neighborhood, and the weight of the new neighbor does not force v to affiliate with it. In both these cases, the third ad hoc clustering property is guaranteed by the execution of the **else** branch, where it is checked if, upon the arrival of a new clusterhead, the k-neighborhood condition is violated. If this is the case, the clusterhead with the minimum weight is determined, and the corresponding RESIGN message is transmitted.

Thus, the three properties for ad hoc clustering are always satisfied.

4 Simulation Results

Here we demonstrate by the use of simulations that MACA achieves substantial improvements with respect to previously existing clustering algorithms.

We simulate our algorithm by placing $n = 30$ nodes randomly on a grid of size 100×100. The speed s_v and the power p_v of every node v are given in grid units. Two nodes v and u in the network are neighbors if the Euclidean distance d_g between their coordinates in the grid is less than the minimum between their transmission radii (i.e., $d_g(v, u) < \min\{p_v, p_u\}$).

Each node has a "lifetime" which is measured in ticks of the simulation clock. The lifetime of each node is chosen randomly and uniformly in the interval $[0, 400]$. When the lifetime of a node expires, the node is considered "dead" for 20 ticks, and then it is assigned a new lifetime. At every tick of the simulation clock, each node that is "alive" determines its direction randomly,

by choosing it uniformly between 0 and 2π. Each node will then move in that direction according to its current speed. When a node reaches the bounds of the grid, it bounces back with an angle determined by the incoming direction. The final position of the nodes is a function of its initial position and of its current speed.

We define the *stability* of a mobility-adaptive clustering algorithm in terms of the number of *elections* and *reaffiliations* per tick. An election occurs all the times that, due to a change in the topology of the network, either an ordinary node or a node that becomes alive decides to be a clusterhead. We have a reaffiliation when either an ordinary node or a clusterhead, or a node that becomes alive, affiliates as an ordinary node with a newly close clusterhead.

MACA is compared here with the clustering algorithm based on the "lowest ID first" approach presented in [5, 4]. In this algorithm the node with the minimum ID in its current neighborhood is chosen as a clusterhead. Since in [4] it is shown that the "lowest ID first" clustering is always more stable than the clustering obtained following a "largest degree first" approach, we do not consider the latter one.

In our first set of simulations we let the weight associated to a node represent its speed. The idea, here, is that the slower a node is, the better it is for the role of clusterhead. Thus, we define the weight of the node v by $w_v = x + 1 - s_v$, where the speed s_v is chosen uniformly in $[1, x]$, $x \in [1, 100]$. In this way, for each x, the nodes that are moving at lower speed are assigned bigger weights. The percentage gains obtained for elections and reaffiliations by MACA with respect to the "lowest ID first" algorithm when all the nodes $v \in V$ have constant power $p_v = 30$ are shown in figures 1 and 2, respectively, for four different values of k. Figures 3 and 4 show the same measures when $p_v = 40$. The choice of these two values for the power is motivated by the fact that with values between 30 and 40 the network is always guaranteed to be connected.

In the other set of simulations, weights are supposed to represent nodes' transmission power: the bigger the power of a node, the better that node for the role of clusterhead. If $x \in [10, 100]$ indicates the maximum transmission power that a node can have, we initialize each node with a power p_v chosen with uniform distribution in the range $[5, x]$. The weight of the node v is then $w_v = p_v$ (in this way, for each x, the nodes with the bigger transmission powers are assigned bigger weights). The results (percentage gains) for elections and reaffiliations are shown in figures 5 and 6, where the speed s_v of each node $v \in V$ varies in the range $[0, 6]$.

In all the simulations $h = \frac{y_{max} - y_{min}}{2}$, where $y_{min} = 1$ for the simulations with variable speed and $y_{min} = 5$ for the simulations with variable power, and $y_{max} = x$. The confidence level of our results is 95% and their precision is 5%.

5 Conclusions

This paper presented an algorithm for giving and maintaining a cluster organization of an ad hoc network in the presence of mobility of possibly all its nodes. This is a practically important task, especially for all those network algorithms/applications that assume a mobility-adaptive hierarchical organization of the network. We have proven that, by associating a generic weight with each node, the weight-based selection of the clusterheads allows the clustering to be configured for specific applications and adaptive to changes in the network status. The proposed algorithm needs only knowledge of the local topology at each node (one hop neighbors), and allows each ordinary node to have direct access to at least a clusterhead, thus guaranteeing fast inter- and intra-cluster communication between each pair of nodes. Finally, we have demonstrated by the use of simulations that when the weight associated with each node are interpreted as a structural characteristic of that node, namely, its speed or its transmission power, we obtain a consistent reduction of the

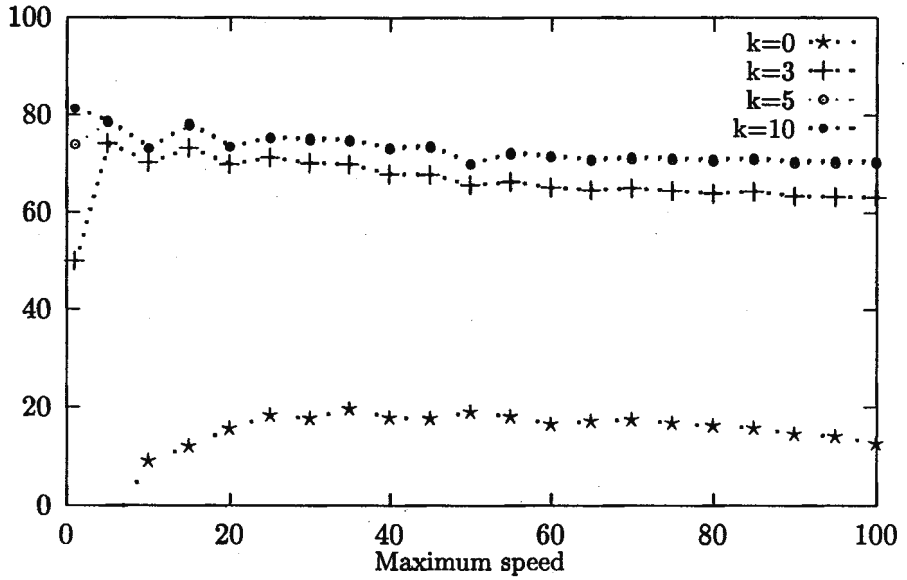

Figure 1 MACA vs. "lowest ID first:" percentage gain for elections when $p_v = 30$ for each $v \in V$

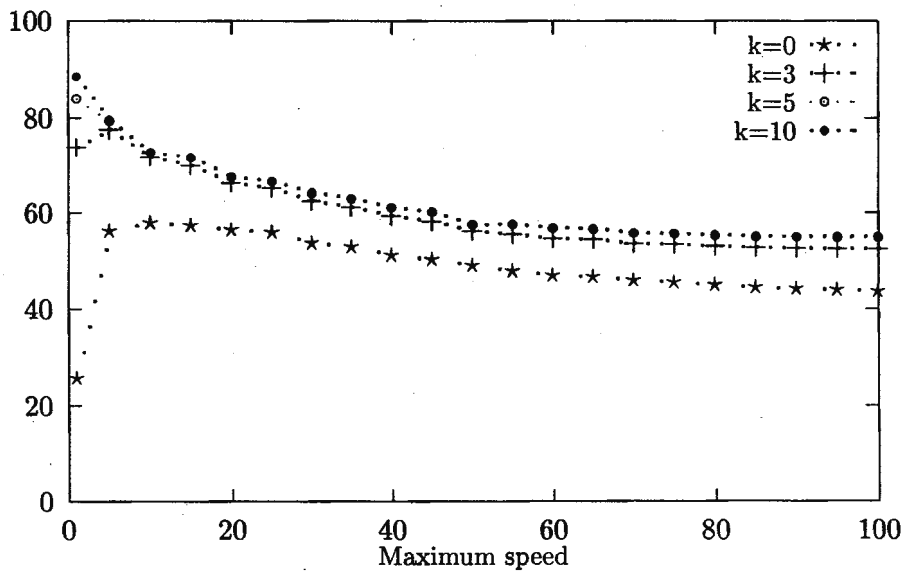

Figure 2 MACA vs. "lowest ID first:" percentage gain for reaffiliations when $p_v = 30$ for each $v \in V$

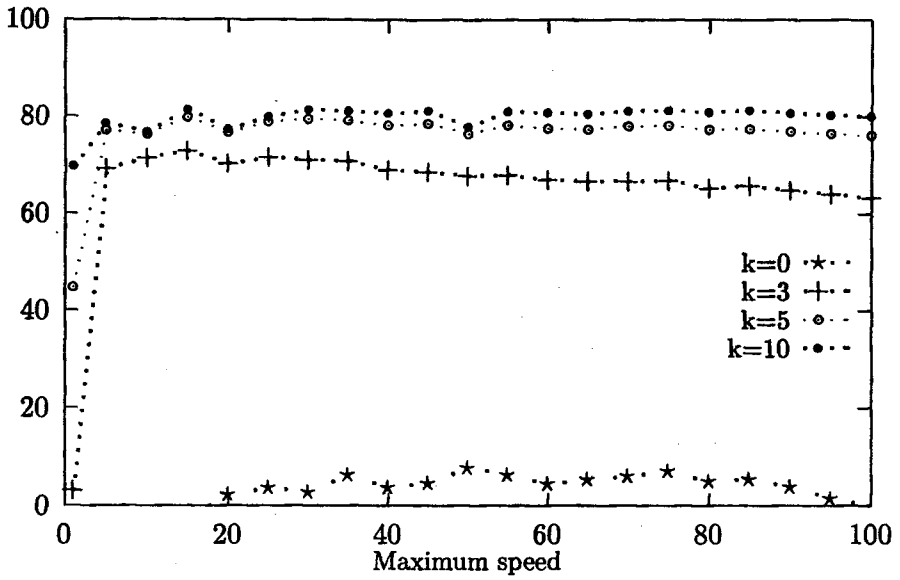

Figure 3 MACA vs. "lowest ID first:" percentage gain for elections when $p_v = 40$ for each $v \in V$

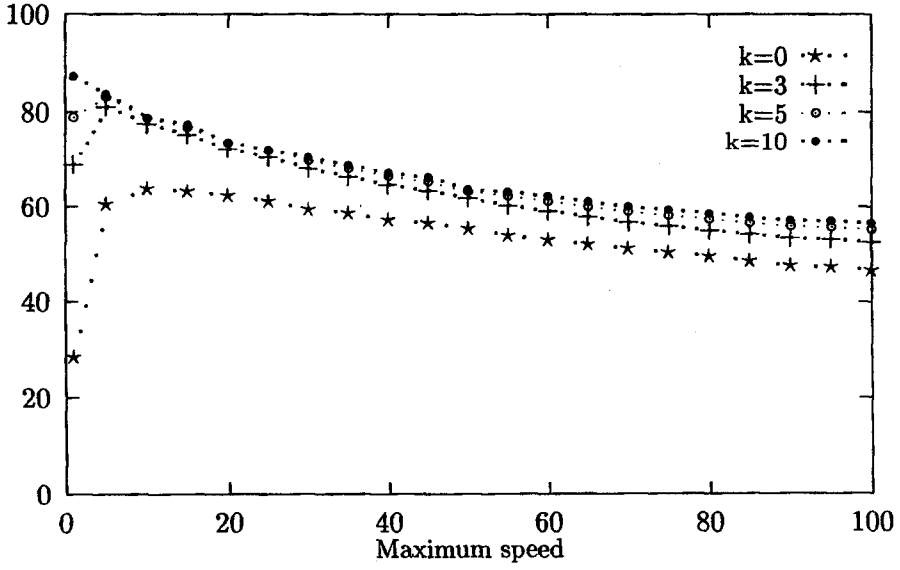

Figure 4 MACA vs. "lowest ID first:" percentage gain for reaffiliations when $p_v = 40$ for each $v \in V$

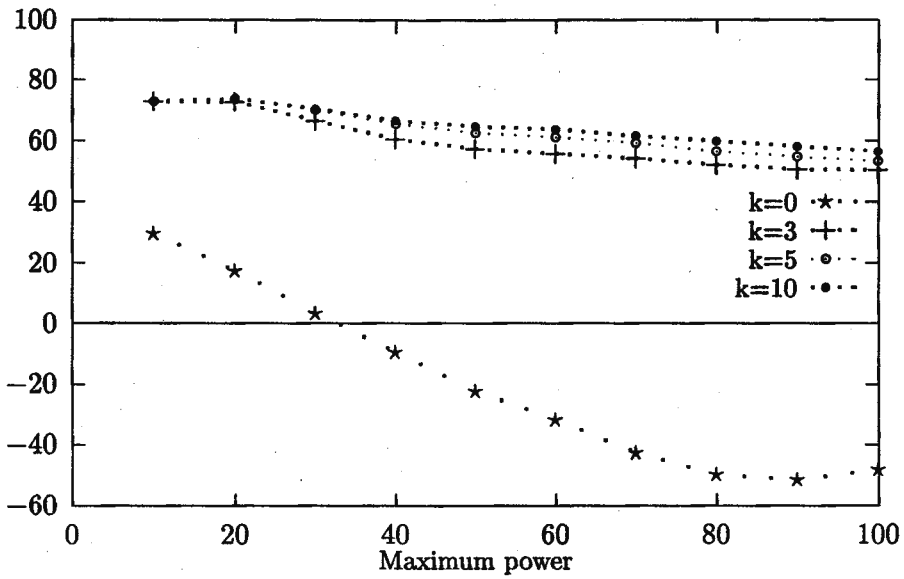

Figure 5 MACA vs. "lowest ID first:" percentage gain for elections when $s_v \in [0, 6]$ for each $v \in V$

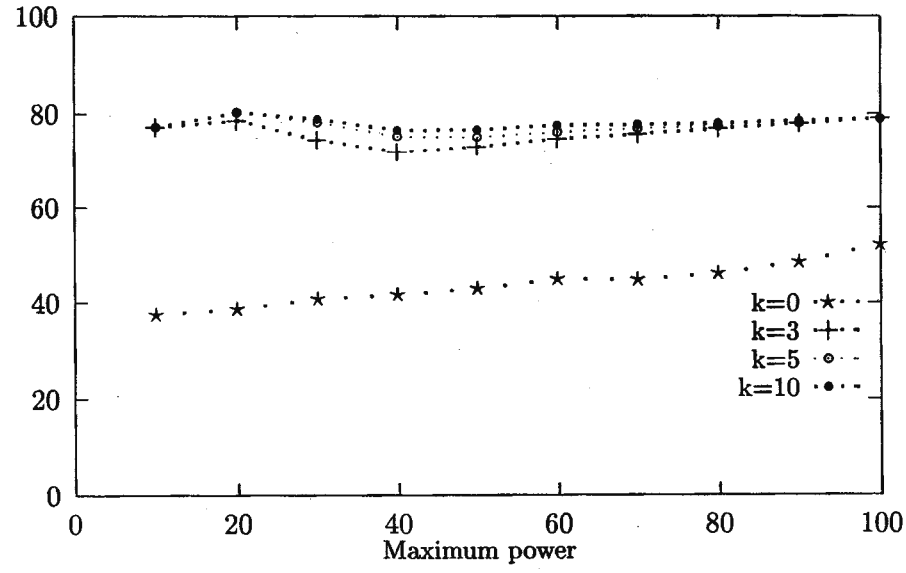

Figure 6 MACA vs. "lowest ID first:" percentage gain for reaffiliations when $s_v \in [0, 6]$ for each $v \in V$

communication overhead associated with the cluster maintenance.

References

[1] BAKER, D. J., EPHREMIDES, A., AND FLYNN, J. A. The design and simulation of a mobile radio network with distributed control. *IEEE Journal on Selected Areas in Communications SAC-2*, 1 (January 1984), 226–237.

[2] EPHREMIDES, A., WIESELTHIER, J. E., AND BAKER, D. J. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE 75*, 1 (January 1987), 56–73.

[3] GERLA, M., AND LIN, C. R. Multimedia transport in multihop dynamic packet radio networks. In *Proceedings of International Conference on Network Protocols* (Tokyo, Japan, 7–10 November 1995), pp. 209–216.

[4] GERLA, M., AND TSAI, J. T.-C. Multicluster, mobile, multimedia radio network. *Wireless Networks 1*, 3 (1995), 255–265.

[5] LIN, C. R., AND GERLA, M. Adaptive clustering for mobile wireless networks. *Journal on Selected Areas in Communications 15*, 7 (September 1997), 1265–1275.

Appendix A. MACA procedures

When a node v is added to the network (this includes the clustering set up) it executes the following procedure:

```
PROCEDURE Init;
begin
   if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
      then begin
              x := max_{w_z > w_v}{z : Ch(z)};
              send JOIN(v,x);
              Clusterhead := x
           end
      else begin
              send CH(v);
              Ch(v) := true;
              Clusterhead := v;
              Cluster(v) := {v};
              if |{z ∈ Γ(v) : Ch(z)}| > k then send RESIGN(min_k{w_z : z ∈ Γ(v) ∧ Ch(z)})
           end
end;
```

When a link with a node u is broken, or a new link with u is created, a node v executes the following procedures:

```
PROCEDURE Link_failure(u);
begin
   if Ch(v) and (u ∈ Cluster(v))
      then Cluster(v) := Cluster(v) \ {u}
```

```
        else if Clusterhead = u then
                if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
                    then begin
                            x := max_{w_z > w_v}{z : Ch(z)};
                            send JOIN(v,x);
                            Clusterhead := x
                         end
                    else begin
                            send CH(v);
                            Ch(v) := true;
                            Clusterhead := v;
                            Cluster(v) := {v};
                            if |{z ∈ Γ(v) : Ch(z)}| > k then send RESIGN(min_k{w_z : z ∈ Γ(v) ∧ Ch(z)})
                         end
end;

PROCEDURE New_link(u);
begin
   if Ch(u) then
       if (w_u > w_Clusterhead + h)
           then begin
                   send JOIN(v,u);
                   Clusterhead := u;
                   if Ch(v) then Ch(v) := false
                end
           else if Ch(v) and |{z ∈ Γ(v) : Ch(z)}| > k then
                      begin
                        w := min_k{w_z : z ∈ Γ(v) ∧ Ch(z)};
                        if w_v > w then send RESIGN(w)
                                   else begin
                                          x := max_{w_z > w_v}{z : Ch(z)};
                                          send JOIN(v,x);
                                          Clusterhead := x;
                                          Ch(v) := false
                                        end
                      end
end;
```

The following procedures are triggered by the reception of the corresponding message:

```
On receiving CH(u);
begin
   if (w_u > w_Clusterhead + h) then begin
                                       send JOIN(v,u);
                                       Clusterhead := u;
                                       if Ch(v) then Ch(v) := false
                                     end
                                else if Ch(v) and |{z ∈ Γ(v) : Ch(z)}| > k then
                                        begin
                                          w := min_k{w_z : z ∈ Γ(v) ∧ Ch(z)};
                                          if w_v > w then send RESIGN(w)
                                                     else begin
```

```
                                                    x := max_{w_x > w_v}{z : Ch(z)};
                                                    send JOIN(v,x);
                                                    Clusterhead := x;
                                                    Ch(v) := false
                                                  end
                                    end
end;

On receiving JOIN(u, z);
begin
   if Ch(v)
      then if z = v then Cluster(v) := Cluster(v) ∪ {u}
                    else if u ∈ Cluster(v) then Cluster(v) := Cluster(v) \ {u}
      else if Clusterhead = u then
              if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
                then begin
                        x := max_{w_x > w_v}{z : Ch(z)};
                        send JOIN(v,x);
                        Clusterhead := x
                     end
                else begin
                        send CH(v);
                        Ch(v) := true;
                        Clusterhead := v;
                        Cluster(v) := {v};
                        if |{z ∈ Γ(v) : Ch(z)}| > k then send RESIGN(min_k{w_z : z ∈ Γ(v) ∧ Ch(z)})
                     end
end;

On receiving RESIGN(w);
begin
   if Ch(v) and w_v ≤ w then begin
                                x := max_{w_x > w_v}{z : Ch(z)};
                                send JOIN(v,x);
                                Clusterhead := x;
                                Ch(v) := false
                             end
end;
```