REPORT R-1025 DECEMBER, 1984 UILU-ENG 84-2219

COORDINATED SCIENCE LABORATORY

CLUSTERING ALGORITHMS FOR HIERARCHICAL ROUTING IN NETWORKS

JAMES ROSSI

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY: N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): R-Report #R-1025 UILU-ENG. 84-2219	5. MONITORING ORGANIZATION REPORT NUMBER(S): N/A
6a. NAME OF PERFORMING ORGANIZATION; 6b. OFFICE SYMBOL (If applicable): N/A	7a. NAME OF MONITORING ORGANIZATION: JSEP
6c. ADDRESS (City, State and ZIP Code): 1101 W. Springfield Avenue, Urbana, Illinois 61801	7b. ADDRESS (City, State and ZIP Code): 800 N. Quincy, Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: JSEP; 8b. OFFICE SYMBOL (If applicable): N/A	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-84-C-0149
8c. ADDRESS (City, State and ZIP Code): 800 N. Quincy, Arlington, VA 22217	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO.; PROJECT NO.; TASK NO.; WORK UNIT NO.
11. TITLE (Include Security Classification): Clustering Algorithms for Hierarchical Routing in Networks (unclassified)	
12. PERSONAL AUTHOR(S): Rossi, James	
13a. TYPE OF REPORT: Technical; 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): December 1984; 15. PAGE COUNT: 34
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES — FIELD; GROUP; SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Packet Radio Networks, Hierarachical Routing

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper we present a distributed asynchronous algorithm to form overlapping clusters on a network with a dynamic topology.

Properties of the algorithm are presented. It is shown that the algorithm is essentially a distributed implementation of a centralized algorithm. Also, bounds on the number or clusters formed on a network with N nodes is derived.

The algorithm (DACA) is then compared with another algorithm (DHCA) for forming overlapping clusters in a dynamic environment. It is shown that in some ways, DACA is better than DHCA. The major disadvantage of DACA is that it passes more bits through the network than DHCA. Which algorithm is better depends on the particular implementation.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code); 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

ABSTRACT

In this paper we present a distributed asynchronous algorithm to form overlapping clusters on a network with a dynamic topology.

Properties of the algorithm are presented. It is shown that the algorithm is essentially a distributed implementation of a centralized algorithm. Also, bounds on the number of clusters formed on a network with N nodes is derived.

The algorithm(DACA) is then compared with another distributed algorithm(DHCA) for forming overlapping clusters in a dynamic environment. It is shown that in some ways, DACA is better than DHCA. The major disadvantage of DACA is that it passes more bits through the network than DHCA. Which algorithm is better depends on the particular implementation.

CLUSTERING ALGORITHMS FOR HIERARCHICAL ROUTING IN NETWORKS

BY

JAMES JOSEPH ROSSI

B.S., Boston University, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1985

Urbana, Illinois

This work was supported by the Joint Services Electronics Program under Contract N00014-84-C-0149.

ACKNOWLEDGEMENT

The author wishes to thank Professor Bruce Hajek for his insight and guidance.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The landmark paper by Kleinrock and Kamoun[1] showed that by clustering nodes–considering a group of n^{th} level nodes as one $n+1^{st}$ level node–the length of the routing table stored at each node would grow as the log of the number of nodes(N) instead of linearly in N. This reduction in table size is translated into a reduction in system overhead, since these tables are passed around the network. That paper assumes that the clustering is performed when the network is built and that each node can be labeled with an ID that will aid in routing. An example of this labeling is given in Figure 1.1.

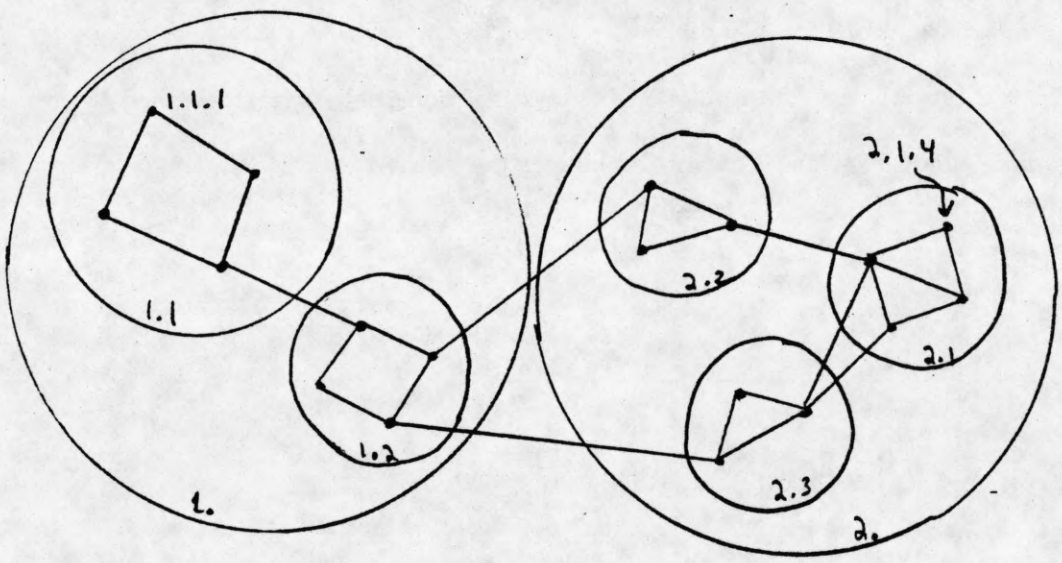


FIGURE 1.1 An example of hierarchical routing for a two-level hierarchy

In this example if node 1.1.1 has a message for node 2.1.4, the message will first be routed along the shortest path to supercluster 2, then along the shortest path to cluster 2.1, and finally along the shortest path to node 2.1.4. This small example shows how hierarchical routing works and the critical role played by the ID of a node.

Networks currently in the research stage have different characteristics than the networks assumed in [1], but we would still like to apply hierarchical routing to them. One such network is the Survivable Radio Network; it consists of mobile nodes and broadcast radio links. In order to implement hierarchical routing, several important issues must be resolved. The two main issues are how to form and maintain clusters in a network with a dynamic topology and how to use these clusters to route messages in the network. Several papers [2],[3],[4],[5] have been aimed at resolving these issues. Paper [2] discusses the issue of routing, and [3] addresses to a lesser extent the issue of clustering. Papers [4] and [5] discuss clustering, but the method of [5] appears to be slow and not well-defined.

It is the purpose of this paper to present a distributed asynchronous algorithm to form overlapping clusters in a dynamic environment. The presented algorithm will then be compared with the algorithm of [4].

Chapter 2 presents the algorithm along with some properties. Chapter 3 briefly presents the algorithm found in [4], and Chapter 4 compares the two algorithms. Lastly, Chapter 5 is the conclusion.

CHAPTER 2

DEMAND-ALLEGIANCE CLUSTERING ALGORITHM

2.1 Description of the Algorithm

2.1.1 Notation

The notation used throughout the remainder of the paper is as follows.

Q^i: Q^i is the priority index assigned to node i. If $Q^i = Q^j$ and $i > j$, then Q^i is considered greater than Q^j in any comparison at any node in the network. Initially, $Q^i > 0$ for each node i in the network. For the decentralized implementation, a node i sets $Q^i = 0$ and tells all nodes within K hops of this news when the node i is no longer a candidate clusterhead. Each node j within K hops of node i would then set its local copy of $Q^i = 0$. A node j more than K hops from node i does not have a local copy of Q^i.

C_i^p: C_i^p is the set of all nodes within p hops of node i.

R: R is the radius in hops of the clusters formed.

S: S is the suppression radius in hops. S+1 is the minimum distance between any two clusterheads. We require that $R \geq S$.

K: This parameter is used in the decentralized algorithm. Each node has information only about those nodes within K hops. The exact information is described later.

2.1.2 Centralized Description

The method employed is a decentralized implementation of the following algorithm. Start with the node i that has the largest Q^i, and declare node i a clusterhead. Its cluster, denoted C_i^R, is the set of all nodes within R hops of node i. If there are any nodes not within S hops of node i, then pick as the next clusterhead the node j having the largest Q^j of all nodes not in C_i^S. Again, if there are any nodes not within S hops of a clusterhead, then repeat as above. When all nodes are within S hops of a clusterhead, the algorithm is complete.

2.1.3 Relationship Among R, S and Q.

From the above description, we can make the following statements regarding the parameters used in the algorithm.

Q^i and S determine which nodes are clusterheads.

S determines how many clusters form for a given network. Proposition 2 in Section 2.2 confirms this point.

R and S determine how much the clusters overlap. The reason we require that $R \geqslant S$ is to eliminate the chance of a node not belonging to a cluster.

The distance between any two nodes in the same cluster is no more than 2R hops.

2.1.4 Decentralized Description and Implementation

2.1.4.1 Decentralized Description

The centralized algorithm uses global information to make decisions. In a decentralized implementation, global information is generally not available, so decisions must be made using only local information. Thus, for the decentralized implementation we assume that there is an integer K with $K \geqslant S$ such that each node i has only the following information about other nodes: the next node along the minimum hop path from node i to each node in C_i^K, the length in hops of each such minimum hop path and the Q^j for each node j in C_i^K. Also, we assume that we have reliable communication in finite time between any two neighboring nodes in the network. This is so that control messages sent among the nodes are actully received by the intended node.

The following distributed algorithm is operating on an asynchronous network. However, we must assume that each node gains the required starting information within a finite amount of time after it begins gathering it. This is necessary for the algorithm to complete in finite time. Also, any node that receives a control message before it has gained the required starting information will take the message and process it until the point where the node checks if it

should be a clusterhead. The node can become a clusterhead only after it has the required starting information.

The decentralized implementation of the algorithm is as follows. A node i starts its portion of the distributed algorithm when it has the necessary starting information(described above). Upon starting, a node i checks if it should be a clusterhead by seeing if $Q^i > Q^j$ for each node $j \neq i$ in C_i^K. If the above is true, then node i sets $Q^i = 0$ and sends a message telling the nodes in C_i^K that it is a clusterhead. Each node p in $C_i^K - C_i^S$, upon receiving the message, sets its copy of $Q^i = 0$, and if the node p is a clusterhead candidate($Q^p \neq 0$), node p checks if it should be a clusterhead in the manner previously described. Each node j in C_i^S, upon receiving the message, takes itself out of consideration by setting $Q^j = 0$, and node j sets its copy of $Q^i = 0$. Node j uses a modified propagation of information protocol(PIF)[6] to send an update message containing $Q^j = 0$ to all nodes in C_j^K. A node p receiving the update message sets its copy of $Q^j = 0$, and if the node p is still participating in the algorithm, it checks if it should be a clusterhead. The above allows only the nodes S+1 hops or more from all clusterheads to be clusterhead candidates at a future time. The algorithm continues as above except that some nodes are no longer in consideration. When all nodes are within S hops of a clusterhead, the algorithm is complete and all nodes are in at least one cluster.

A more formal description of the algorithm is given in the following sections.

2.1.4.2 Messages and Variables Used in the Decentralized Algorithm

Below are the data structures and the messages used in the distributed algorithm. Each node has its own copy of the data structures, and each node is capable of generating any of the messages.

ID: This is the node identification; no two nodes have the same ID.

CID.LIST: This is a list of all the clusterheads that are within R hops of the node. Initially, the list is empty.

CID: This is a clusterhead(or cluster) ID in a CID.LIST at some node.

T(ID,CID): This is the routing table at node ID for cluster CID; ID is in cluster CID. The routing table contains the first node along the current path from node ID to each node in C^R_{CID}. Also, the distance to each node in C^R_{CID} is given in the table. Initially, T(ID,*), for any potential clusterhead * , contains only the previously described information for the nodes in C^K_{ID}. At the completion of the algorithm, each node ID has one T(ID,CID) for each CID in its CID.LIST.

C.O[ID,CID,T(ID,CID)]: This is a message that tells other nodes in T(ID,CID) that node CID is a clusterhead. It is also used to pass the T(ID,CID) to neighboring nodes in T(ID,CID).

Q.U[ID,HC]: This is a message that tells all nodes in C^K_{ID} that $Q^{ID} = 0$ for the remainder of the algorithm. Q.U[ ] is part of the PIF started by node ID. A node j receiving this message would then set its copy of $Q^i = 0$.

HC: This is a counter that allows a Q.U[ ] message to travel only K hops from the message origin. A node receiving the message checks if $HC < K$. If this is so, then HC is incremented in the message, and the message is broadcasted to all neighboring nodes. This is the modified portion of the PIF.

d(i,j): This is the number of hops in the minimum hop path from node i to node j.

M_{ID}: This is a variable used in the PIF protocol to keep track of the received Q.U[ID,*] messages. This prevents multiple copies of the same message from being sent on a link. Initially, $M_{ID} = 0$ for all node ID's. After the first copy of a Q.U[ID,*] message has been broadcasted, $M_{ID} = 1$ for the remainder of the algorithm.

CHANGE.FLAG: This is a variable at each node i that tells the node whether an update has been made to the current T(i,CID) during the table update portion of the algorithm. If any changes have been made, then T(i,CID) is sent to

neighboring nodes.

ON: This variable is used as follows. Initially, ON=0. When a node has the previously described starting information, it starts its part of the distributed algorithm. ON is then set to 1. This variable prevents a node from becoming a clusterhead before it has the required starting information while allowing a node to process incoming messages.

All other variables are as defined in Subsection 2.1.1.

2.1.4.3 Decentralized Implementation

The following algorithm operates at each node i in the network. The simultaneous operation at all nodes yields the desired distributed algorithm.

```
For CHECK                                   /* This checks if node i should become a */
                                            /* clusterhead. */
1  IF Q^i > Q^j                             /* If node i has higher Q */
   for all  j ≠ i  ∈ C_i^K                  /* than any other node */

       THEN                                 /* in C_i^K, then node i becomes */
       BEGIN
       ADD  i to CID.LIST                   /* a clusterhead and tells everyone.*/
          BROADCAST C.O[i,i,T(i,i)]
       Q^i = 0
       END
END FOR


IF Node i receives a C.O[j,CID,T(j,CID)]
  BEGIN
    Call TABLE.UPDATE                       /* Table.update takes T(j,CID)*/
                                            /*and uses it to produce an updated */
                                            /* version of T(i,CID).*/
                                            /* see later description of TABLE.UPDATE */
                                            /* T(i,CID), if it was changed during the*/
                                            /*update, is passed to neighboring nodes*/

                                            /*in C_CID^R via the C.O[ ] message.*/

    Add CID from message to CID.LIST        /* This keeps track of clusterheads */
                                            /*of clusters to which node i belongs.*/
    Q^CID = 0                               /* This takes node CID out of future*/
                                            /*consideration.*/
```

```
2    IF i ∈ C^S_CID                          /* If node i is too close to the clusterhead,*/
       THEN
         BEGIN                               /* then take self out of future consideration.*/
          Q^i = 0
          Broadcast Q.U[i,HC= 1]             /* Tell all nodes within K hops*/
                                             /* that node i is no longer in */
                                             /* consideration.*/
       END
3      ELSE
4      BEGIN
5          IF Q^i ≠ 0 and ON = 1            /* If node i is ON and still in consideration */
6              THEN CHECK                   /*then, check to see if*/
7      END                                  /* node i should become clusterhead.*/
   END IF
END IF


IF NODE i receives Q.U[ID,HC]
                                            /* M_ID = 0 when algorithm is initialized. */
  BEGIN
   Q^ID = 0                                 /* Take node ID out of future consideration.*/
   IF HC < K and M_ID = 0                   /* If message has traveled less than */
       THEN                                 /* K hops and the Q.U[ID,*] message */
         BEGIN                              /* has not been relayed by node i before, then */
          M_ID = 1                          /*send message to neighboring nodes. */
          HC = HC + 1
          BROADCAST Q.U [ID,HC]
       END
   END IF

   IF Q^i ≠ 0 and ON = 1                    /* If node i is still in */
       THEN CHECK                           /* algorithm, then check */
   END IF                                   /* if it should be a clusterhead.*/
END IF
```

When this algorithm is complete, each node will know about all clusterheads within R hops. Also, each node will know the next node along the shortest path to each node in C^R_{CID} for each CID in the CID.LIST at node i.

Procedure TABLE.UPDATE, shown below, is the mechanism that finds the shortest paths between any two nodes in the same cluster. It is similar to the distributed protocol used to find shortest paths in the ARPANET.

PROCEDURE TABLE.UPDATE

```
BEGIN
  CHANGE.FLAG = 0                          /* initialize */
  STORE T(j,CID)
  IF this is first reception  of T(j,CID)
      THEN
      BEGIN
       CHANGE.FLAG = 1
    END
  FOR each node k
      common to T(j,CID) and T(i,CID)
      BEGIN
        IF d(k,i) > d(k,j) + 1             /* If current path is longer */
          THEN                             /*  than new path, then keep*/
              BEGIN                        /*  new path and update     */
                                           /*  distance and next node. */
                UPDATE entry k
                in T(j,CID) with
                 next_node = j
                 dist = d(k,j)+1
                CHANGE.FLAG = 1            /* This keeps track of any */
          END                              /* changes to T(j,CID).*/
      END IF
    END
  END FOR
FOR each entry p in T(i,CID)
   but NOT in T(j,CID) such that p ∈ C^R_CID

  BEGIN                                    /* This is how a node finds out */
  PLACE entry(p) in T(j,CID)               /* about nodes outside K hops of */
  CHANGE.FLAG = 1                          /* itself but within R hops.    */
  END
END FOR
IF CHANGE.FLAG = 1                         /* If any changes were made to table,*/
  THEN
      BEGIN
        T(i,CID) = T(j,CID)                /* then keep modified  table, and */
        SEND C.O[i,CID,T(i,CID)]
          to all neighboring nodes that

          are in C^K_CID

                                           /*send message so that*/
                                           /*other nodes in cluster*/
                                           /* get T(i,CID). */
    END
  END IF
END PROCEDURE TABLE.UPDATE
```

2.2 Properties of the Algorithm

The first question to consider is whether the distributed algorithm using only local information yields the same results as the centralized algorithm using global information. Under a few mild assumptions, the two algorithms produce the same clusterheads. Proposition 1 verifies the above.

Proposition 1: Suppose we are given a network with the following assumptions.

Q^i, for each node i, does not change as we change K.

$S > 0$

All previously stated assumptions about the network operating conditions hold.

The following statements are then true.

a) Any value of $K \geq S$ used in the distributed algorithm will yield the same clusterheads as the centralized algorithm. Furthermore, it does so in finite time.

b) $|C_i^K|$ is an upper bound to the number of times a node checks if it should become a clusterhead. Also $|C_i^K|$ is a nondecreasing function of K.

PROOF:

Define the j^{th} local maximum as the node that becomes a clusterhead in the j^{th} iteration of the *centralized* algorithm. Let m_j be the ID of the j^{th} local maximum. Note that any two local maxima are separated by at least S+1 hops.

Proof of a):

First we show that the two algorithms yield the same clusterheads. The proof of this is by induction. Then we show that this happens in finite time.

P(1) m_1 is a cluster head and no other node in $C_{m_1}^S$ is a clusterhead at the completion of the decentralized algorithm.

pf. Node m_1 is the global maximum, so $Q^{m_1} > Q^j$ for each node $j \neq m_1$ in the network. Any node j in $C^S_{m_1}$ will see that $Q^{m_1} > Q^j$ and, as a result, will not become a clusterhead as long as $Q^{m_1} \neq 0$. Also, a node i cannot force a node j to set $Q^j = 0$ unless node i becomes a clusterhead and node j is in C^S_i. Combining the last two statements, we find that no node can force node m_1 to set $Q^{m_1} = 0$. Eventually node m_1 will start its portion of the distributed algorithm, see that $Q^i > Q^j$ for each node j in C^K_i and become a clusterhead. As a result of node m_1 becoming a clusterhead, each node $j \neq m_1$ in $C^S_{m_1}$ will be forced to set $Q^j = 0$, thereby preventing a node j in $C^S_{m_1}$ from becoming a clusterhead.

Now suppose P(j-1) is true: that $m_1, \ldots, m_{j-1}$ are clusterheads and that no other node in $\bigcup_{i=1}^{j-1} C^S_{m_i}$ is a clusterhead at the completion of the decentralized algorithm. We will show that our statement P(j) is true:

P(j) $m_1, \ldots, m_j$ are clusterheads, and no other node in $\bigcup_{i=1}^{j} C^S_{m_i}$ is a clusterhead at the completion of the distributed algorithm.

pf. Consider node m_j and $C^S_{m_j}$. The nodes in $C^S_{m_j}$ fall into two categories, those nodes in

$$T = \bigcup_{i=1}^{j-1} C^S_{m_i}$$

and those in T^c. Since node m_j is the j^{th} local maximum, $Q^{m_j} > Q^i$ for any node $i \neq m_j$ in $C^S_{m_j} \bigcap T^c$. By our assumption of P(j-1), no node in

$$T - \{m_1, \ldots, m_{j-1}\}$$

is a clusterhead at the completion of the algorithm. Also, requiring that $d(m_i, m_j) \geq S+1$ for all $i \neq j$ gives us that

$$\{m_1, \ldots, m_{j-1}\} \bigcap C^S_{m_j} = \phi.$$

From this we conclude that no node in $T \bigcap C^S_{m_j}$ can ever be a clusterhead. Eventually each node i in $T \bigcap C^S_{m_j}$ will set $Q^i = 0$. Node m_j will wait for this to happen, will then satisfy the condition $Q^{m_j} > Q^i$ for all $i \neq m_j$ in $C^S_{m_j}$ and become a clusterhead. Each node i in $C^S_{m_j}$

will then take itself out of future consideration by setting $Q^i = 0$. This completes the proof of P(j). Therefore, by induction, P(j) is true for all j, which implies that the centralized and the distributed algorithms yield the same set of clusterheads.

To see that the decentralized algorithm finishes in finite time, consider the following. There are a finite number of clusters formed in a network with N nodes; it is trivially upper bounded by N, the number of nodes. If each cluster is formed by the decentralized algorithm in finite time, then the entire algorithm completes in finite time. Each cluster is formed in finite time by the decentralized algorithm since we have reliable communication in finite time and all nodes start participating in the algorithm in finite time.

The reason for having $K \geqslant S$ is as follows. Suppose $K < S$. Then two nodes separated by more than K hops but less than S hops could try to become clusterheads at the same time. Depending on the delays incurred by messages and the relative staggering of the starting times at the two nodes, either one or both of the nodes could become a clusterhead. Clearly, the centralized algorithm would not allow two nodes within S hops to both become clusterheads. So, we require $K \geqslant S$ to prevent the above situation.

□

Proof of b):

A node i stops checking if it should be a clusterhead if $Q^i = 0$. Node i would have to check if it should be a clusterhead a maximum of $\left|C_i^K\right|$ times before it would set $Q^i = 0$, either by becoming a clusterhead or by being within S hops of a clusterhead. This bound is met for a node i when $Q^i < Q^j$ for all j in C_i^K and all nodes in $C_i^K - \{i\}$ do not become clusterheads. $\left|C_i^K\right|$ is a nondecreasing set function of K. This completes the proof of part b and Proposition 1.

□

Proposition 1 not only tells us that the centralized algorithm and the decentralized algorithm with limited information give the same results, it also tells us that looking further into

the network than we can influence(i.e. $K > S$) will not help us. Increasing K cannot decrease the maximum amount of processing at a node. So, for the remainder of the paper we set $K = S$ to reduce the number of parameters.

The above simplification also allows us to remove lines 3-7 of the algorithm. The reasoning for this is as follows. Any node i in $C^R_{CID} - C^K_{CID}$ receiving the C.O[ ] message would have had no knowledge of node CID previous to the message, and as such, would have never competed with node CID.

Another important characterization of the algorithm is how many clusters form in a network with a given set of parameters. This is important because research[3] suggests that the optimum number of clusters for a network with N nodes and a one level hierarchy is $N^{1/2}$. This optimum is with respect to minimizing the maximum table size at any node in the network. The table at node i consists of an entry for each node in a cluster with node i and an entry for each cluster in the network. Proposition 2 provides bounds on the number of clusterheads in a network.

Proposition 2: Given a network with N nodes, diameter D, and a value of S, the number of clusters formed by the algorithm is bounded as follows.

$$\left\lceil \frac{D}{2S+1} \right\rceil \leqslant M \leqslant \left\lfloor \frac{N}{1+\left\lceil \frac{S}{2} \right\rceil} \right\rfloor$$

where $\lfloor a \rfloor$ is the largest integer smaller than a, and $\lceil a \rceil$ is the smallest integer larger than a.

PROOF:

If the network is not connected, then each subnetwork operates independently, and the bounds apply to each subnetwork separately. So without loss of generality, the network is connected.

Given any two clusterheads in a connected network, they are separated by at least S nodes. Associate $\left\lfloor \frac{S}{2} \right\rfloor$ of the nodes with one clusterhead and the other $\left\lfloor \frac{S}{2} \right\rfloor$ with the other clus-

terhead. These two groups do not overlap since $2\left\lfloor\frac{S}{2}\right\rfloor \leqslant S$. If we have M clusterheads, there must be at least $M(1+\left\lfloor\frac{S}{2}\right\rfloor)$ nodes in the network. Thus

$$M \leqslant \left\lfloor \frac{N}{1+\left\lfloor\frac{S}{2}\right\rfloor} \right\rfloor .$$

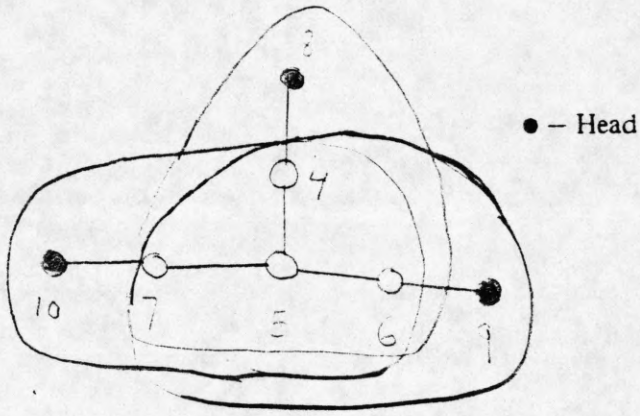
Between any two clusterheads there are at most $2S$ nodes. At worst each clusterhead along the diameter would take up $2S + 1$ nodes. Thus

$$M \geqslant \left\lceil \frac{D}{(2s+1)} \right\rceil .$$

□

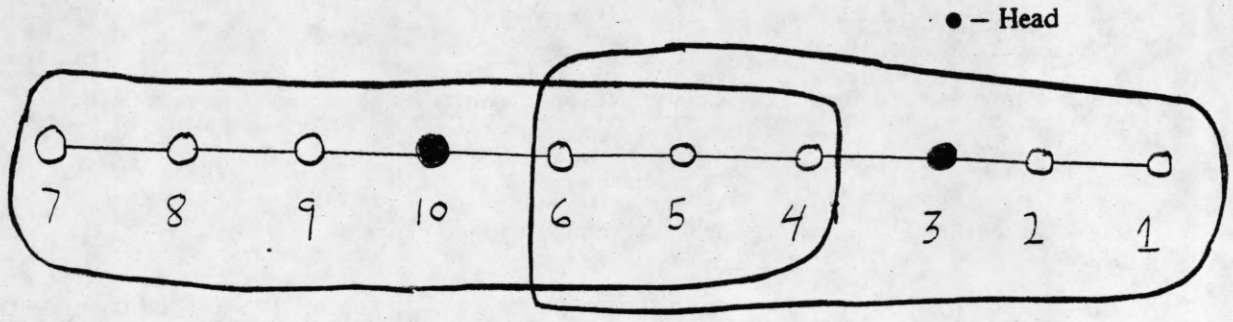
These bounds are loose in general. However given certain networks, these bounds are met. Figure 2.2.1 shows an example where the upper bound is met, and Figure 2.2.2 shows an example where the lower bound is met.

Using these bounds, we can see that the number of clusters is a function of S,N and D only. By monitoring N and D, we can adjust S for each successive running of the algorithm to try for the optimum.



$S = 2, R = 3, N = 7, M \leqslant 3$

FIGURE 2.2.1 This is a network where the upper bound is met.



$$S = 3, R = 3, N = 10, D = 10, M \geq 2$$

FIGURE 2.2.2 This is a network where the lower bound is met.

2.3 Suppression Strategies

The algorithm just presented has the property that a node i, upon becoming a clusterhead, suppresses all nodes within S hops. Thus regardless of how good a clusterhead a node might potentially be, it is not allowed to become one if it is too close to another clusterhead.

Another way to suppress nodes is based on an idea presented in [7]. Suppose that instead of arbitrarily having node i force off a node j, we first see how different C_j^S is from C_i^S for each j in C_i^S . If C_j^S is sufficiently different from C_i^S , then node j is allowed to participate in the next iteration.

One way to see how different two clusters are is to calculate

$$a(i,j) = \frac{|C_j^s \cap C_i^s|}{|C_i^S \cup C_j^S|}.$$

If a(i,j) is smaller than a threshold, then node j is allowed to participate in the next iteration. a(i,j) is easily calculated because when node i becomes a clusterhead it sends out a message containing a table of the nodes in C_i^S.

The presented algorithm is easily modified to incorporate the above idea. If line 2 of the algorithm is replaced with

$$\text{IF } a(i,j) \geq T \text{ ,}$$

then the algorithm will perform the desired operation. Of course a(i,j) must first be calculated.

2.4 Q^i Initialization Strategies

The algorithm presented makes decisions based on the number Q attached to each node. The only constraints placed on the Q's are that $Q^j > 0$ for all j, and that if $Q^i = Q^j$ and $i > j$, then $Q^i > Q^j$ in line 1 of the algorithm.

The easiest and most obvious way to initialize Q^i is simply to let $Q^i = i$. This naive method is biased towards the nodes with higher id numbers and makes no attempt to incorporate the local network conditions into the Q's.

A more intelligent way to assign Q's is such that we try to shorten the time the algorithm takes to complete and also try to minimize the average distance from a clusterhead to a node in the cluster. A function that meets the above requirements is

$$Q^j = \frac{|C_j^S|}{\bar{d}},$$

where

$$\bar{d} = \sum_{i \in C_j^S} \frac{d(i,j)}{|C_j^S|}.$$

This choice of Q^j favors larger clusters, so more nodes set $Q^j = 0$ when a cluster forms. This tends to speed up the algorithm.

Figure 2.4.1 illustrates the Q^i initialization for the three networks used as examples in Chapter 4.

Exactly how well this scheme performs relative to other schemes depends on the cost function used. We can get a feel for its performance if we compare this scheme to that of assigning the Q^i's randomly. Randomly assigning the Q's is similar to using $Q^i = i$ and allowing the nodes to be anywhere in the network.

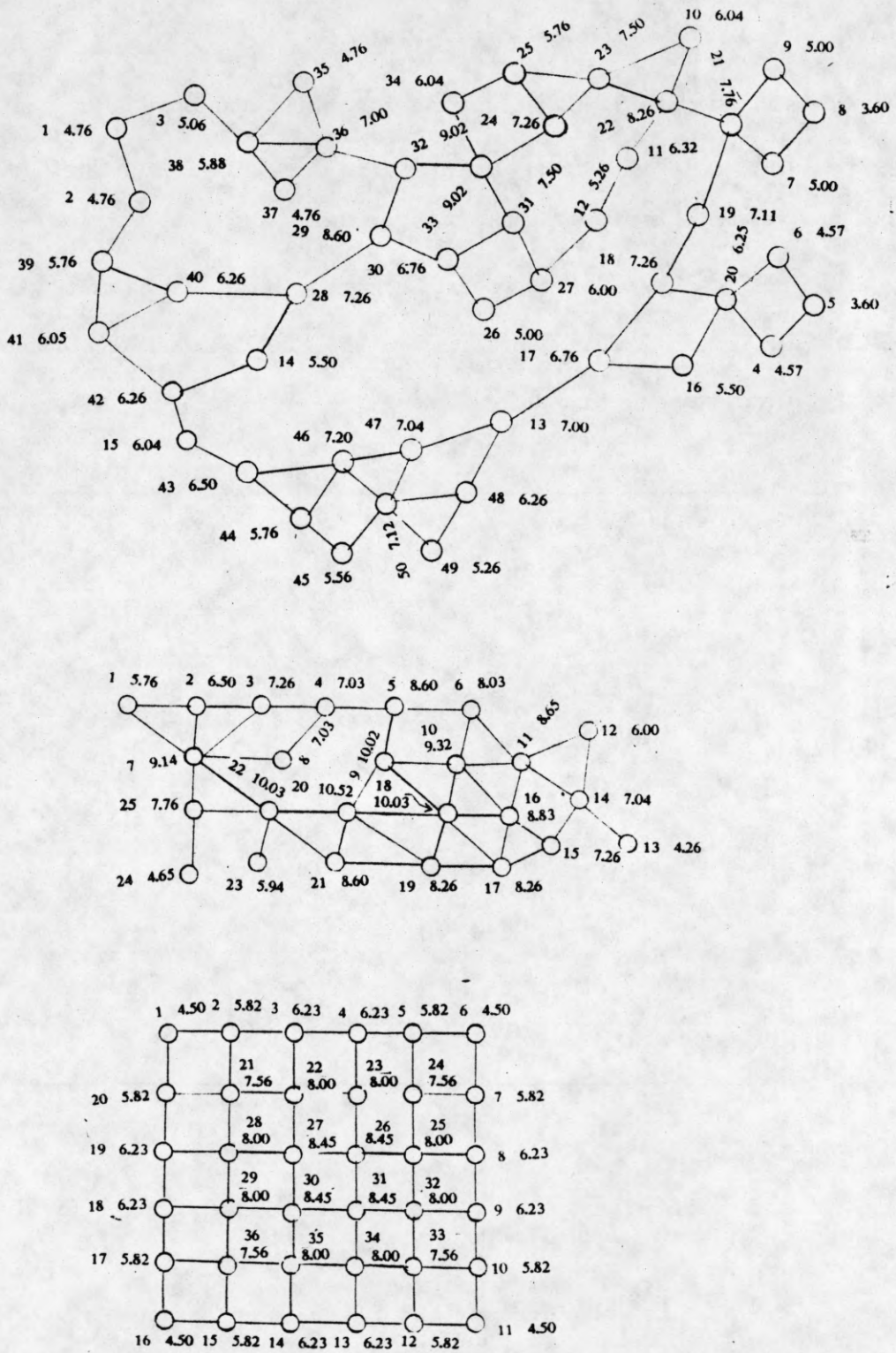


FIGURE 2.4.1 The three sample networks and the assigned Q^i's. The integers are the node ID's, and the floating point numbers are the Q^i's.

For the purpose of illustration, suppose the cost function is

$$J = \frac{1}{N}\sum_{i=1}^{N}\left[\min_{j} d(i,j)\right]^2 + F^*M$$

where j is a clusterhead, F is some finite positive constant and for each i the minimum is taken over all nodes j in node i's CID.LIST. This cost function is the mean square distance from a node to its nearest clusterhead —denoted $\bar{d}$ — plus the cost for M clusterheads.

A graph of $\bar{d}$ verses M shows that for many choices of F the strategy is good. Figure 2.4.2 is such a graph for the fifty node network of Figure 2.4.1. Note that the random selection strategy gives worse results most of the time.

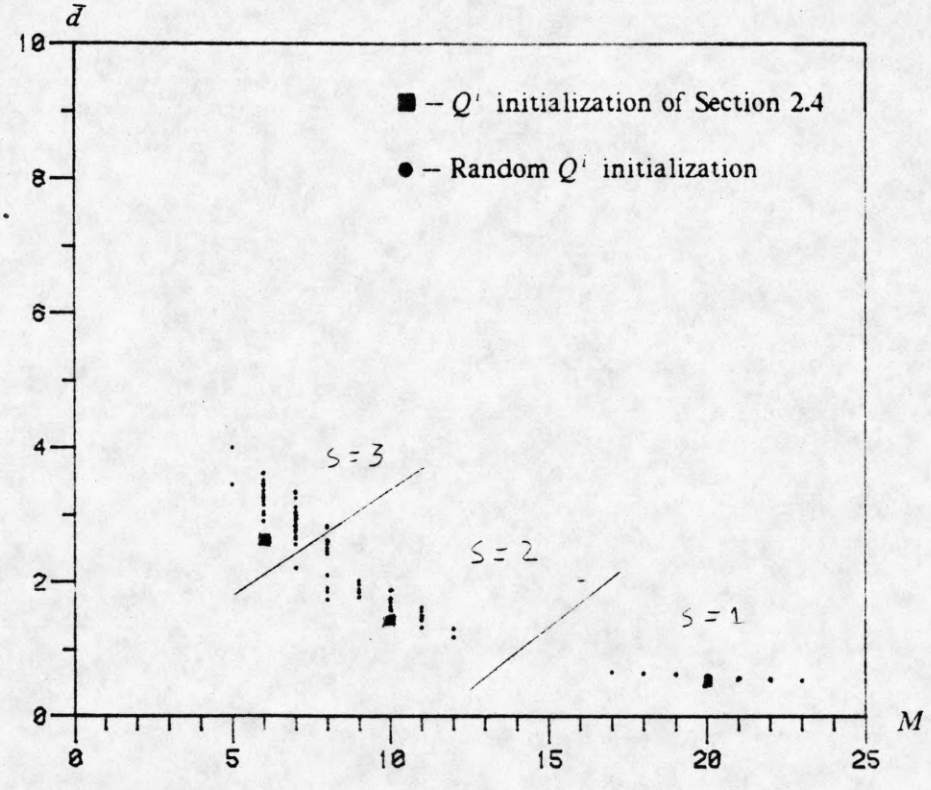


FIGURE 2.4.2 This is a graph of $\bar{d}$ verses M for DACA. There are 150 data points, many of which are the same.

We can visualize our cost for a given F by considering the family of lines with slope $-F$; the cost is the Y intercept. The line with the lowest Y intercept such that the line touches a sample point gives us the minimum cost. Note that for F in either [.45,.78] or [.12,.17], DACA

with the Q^i as defined yields a lower cost than any of the simulation outcomes with random initialization.

The presented initialization strategy is not as good for highly connected networks as it is for sparsley connected networks. To see this, suppose that every node is connected to exactly p other nodes. Then each node will have the same value of Q^i. When this occurs, our algorithm behaves the same as when $Q^i = i$. This makes the extra computation of Q^i unnecessary.

2.5 Maintaining Clusters

There are basically two ways to maintain the cluster structure in a dynamic environment. The first is to periodically run a new cycle of the clustering algorithm. The second is to use a hand off procedure, as in cellular radio, to keep track of nodes as they move from one cluster to another. The preferred method is probably some combination of the two methods. This would allow us to handle both short term fluctuations and long term trends.

CHAPTER 3

DRAFTED-HEAD CLUSTERING ALGORITHM

3.1 Introduction

This section is a brief description of the algorithm presented in [4]. It is included here for later comparison to the Demand-Allegiance Clustering Algorithm(DACA); those seeking detailed information should consult [4].

3.2 Centralized Algorithm Description

The Drafted-Head Clustering Algorithm(DHCA)-- called the Linked Cluster Algorithm in [4]-- is a distributed implementation of the following algorithm. For simplicity, our network has N nodes, and each node is labeled with a distinct number from 1 to N.

Start with node i labeled N, and declare it a clusterhead. Its cluster, C_i^R, is the set of all nodes within R hops of node i. Next, consider the node j labeled N-1. If C_j^R contains a node not in a previous cluster, then C_j^R remains. This procedure is repeated until all nodes belong to at least one cluster.

3.3 Distributed Algorithm Description

The above implemented as a distributed algorithm is as follows. Each node i has the following information for each node in C_i^R: the next node along the minimum hop path from node i to each node in C_i^R, the length in hops of each such path and the Q^j for each node j in C_i^R. Each node i searches its table to find the node j with the largest Q^j. Node i tells that node j to become a clusterhead(i.e., nodes are drafted). Node j then tells all nodes within R hops that it is a clusterhead. Once all the clusters are formed, any clusterhead that finds its cluster completely contained in another cluster deletes itself.

[4] only considers the case where R=1. The algorithm has been generalized here so that it can be compared with the Demand-Allegiance Cluster Algorithm(DACA).

3.4 Q^i Initialization Strategies

In [4] there is no discussion of how to initialize the Q^i's. It appears that the authors have chosen $Q^i = i$.

Suppose we use the Q^i initialization and cost function of Section 2.4. This not only gives us a way to compare DACA and DHCA, it also gives us a way to see if the Q^i initialization is good for this algorithm. Again, we compare this strategy to that of random assignment of the Q's. Figure 3.4.1 shows a graph of $\bar{d}$ verses M for the fifty node network of Figure 4.1.1.

From Figure 3.4.1, we can see that the random labeling does a little better at times and does much worse at other times. Thus, for practical purposes the initialization strategy presented is clearly better.

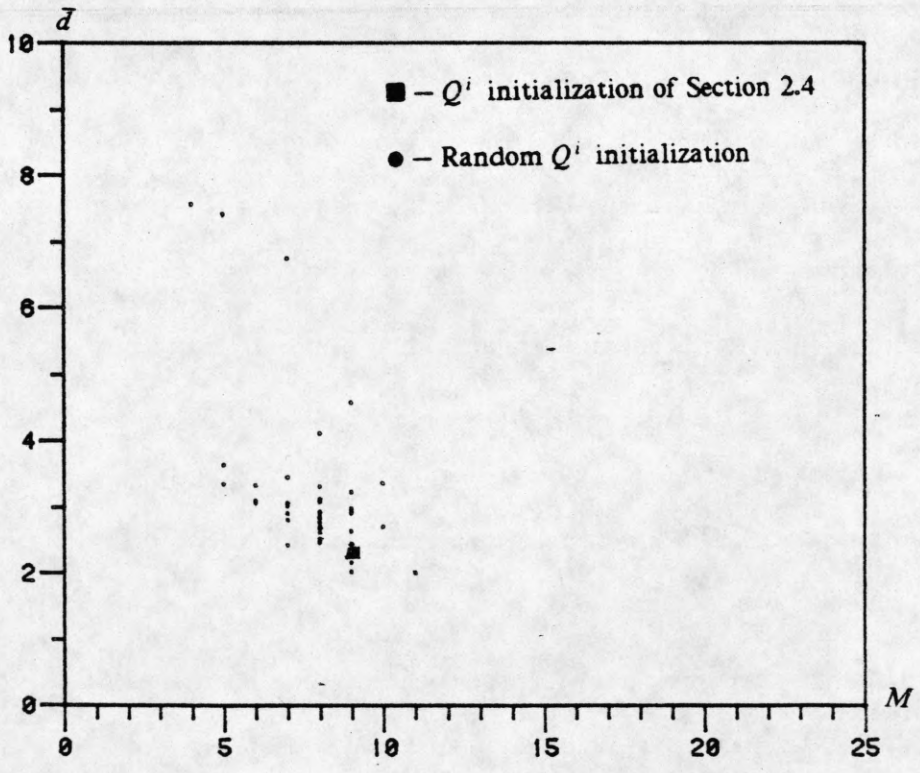


FIGURE 3.4.1 This is a graph of $\bar{d}$ verses M for DHCA.

CHAPTER 4

COMPARISON OF THE DRAFTED-HEAD AND THE DEMAND-ALLEGIANCE CLUSTERING ALGIORITHMS

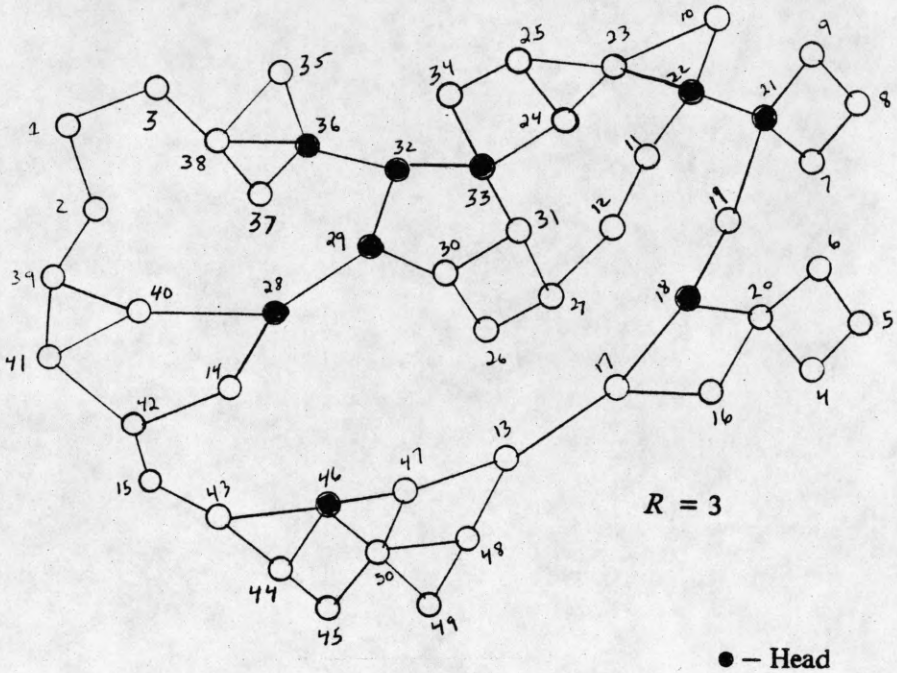
4.1 Main Differences

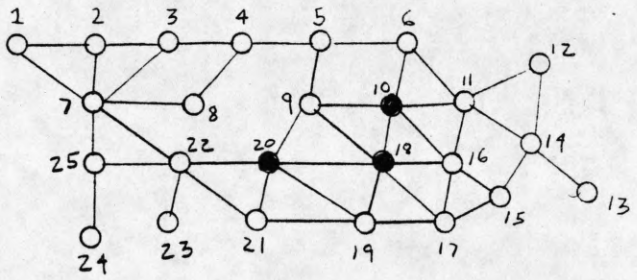
The first difference to note is that DHCA makes no attempt to spread clusterheads evenly throughout the network, while DACA forces clusterheads to be separated by at least S+1 hops. From this we could expect that DHCA would tend to form several aggregations of clusterheads in the network. Figure 4.1.1 shows three outcomes of DHCA, Figure 4.1.2 shows three outcomes of DACA, and Figure 4.1.3 shows three outcomes of DACA using the suppression strategy of Section 2.3. In all cases, the Q^i is as defined in Section 2.4.

The second difference is that DHCA tends to create more clusterheads than DACA for a given instance of a network. This is because in DHCA a node i becomes a clusterhead if it has the largest Q^i in C_j^K for j in C_i^K, while in DACA a node i becomes a clusterhead if it has the largest Q^i in C_i^K at some time. Many of the clusterheads formed by DHCA would be forced off in DACA. The example in Figure 4.1.4 illustrate this point. Also the results in Section 4.2 confirm this point.

Another consideration is how the two algorithms compare with respect to the cluster cost function of Sections 2.4 and 3.4. Figure 4.1.5 is a merging of Figures 2.4.1 and 3.4.1, and corresponds to random Q^i initialization for both algorithms. Note that most of the points corresponding to DACA lie below those corresponding to DHCA. So, for this given cost function, DACA is clearly better than DHCA.

The last important consideration is speed of execution. DACA is slower than DHCA mainly because of the increase in communication overhead. How much more communication is the subject of Section 4.3.





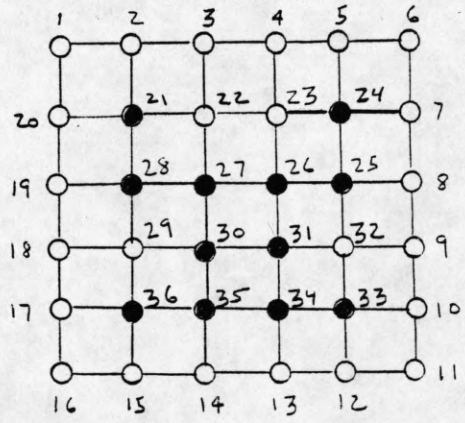
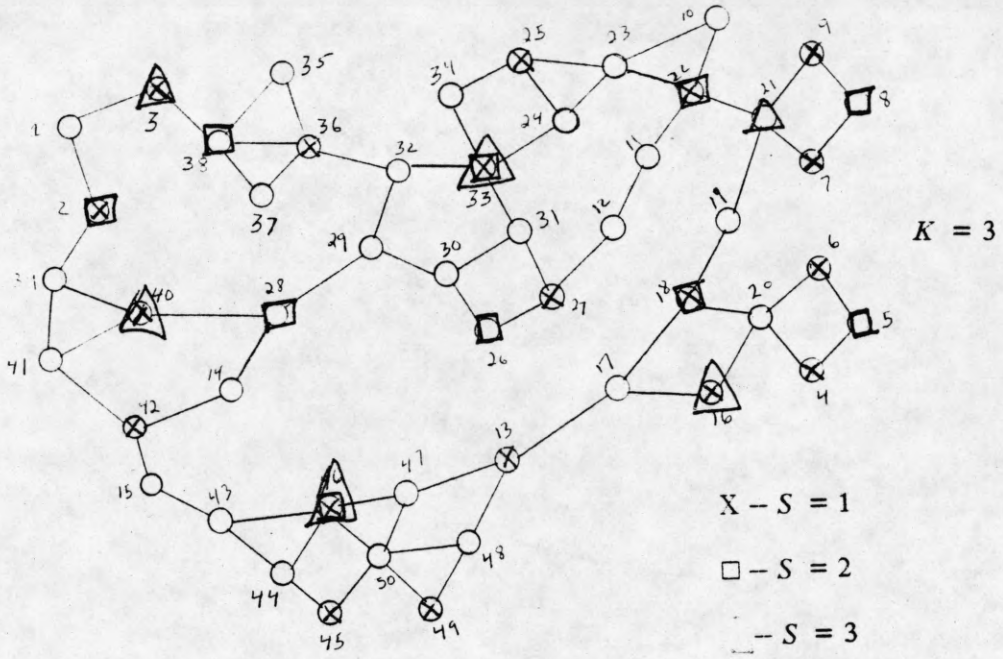
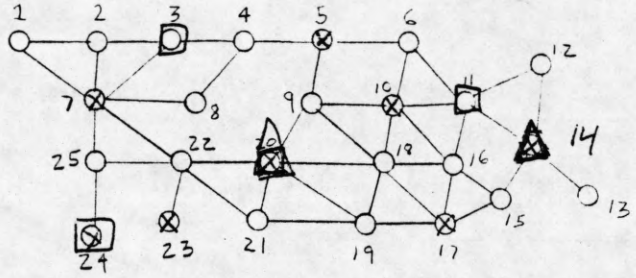


FIGURE 4.1.1 Sample outcomes of DHCA using the priority indices Q^i defined in Section 2.4





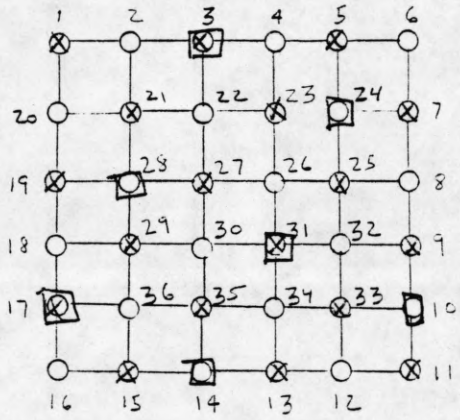
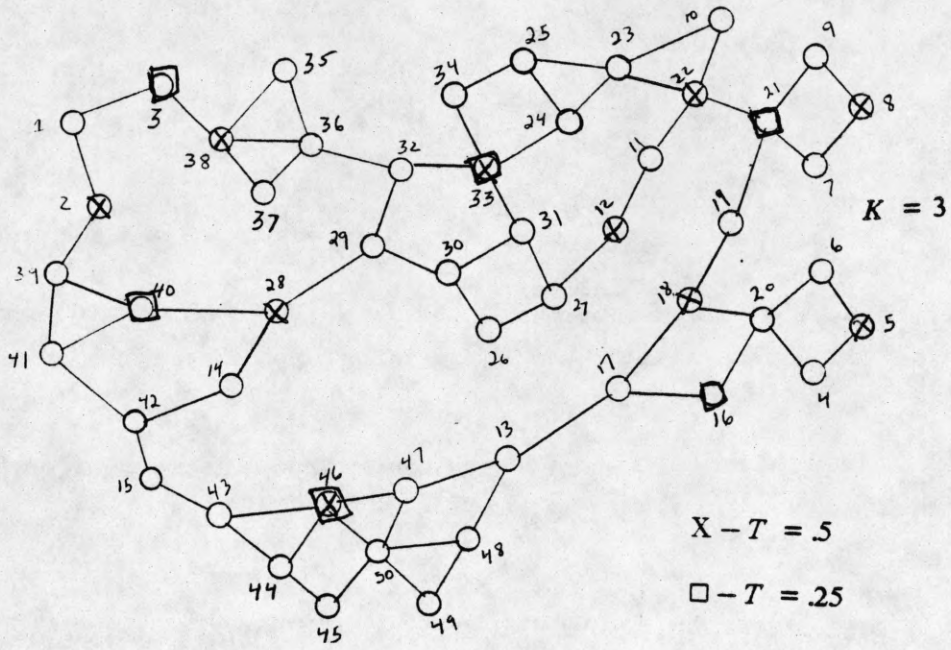
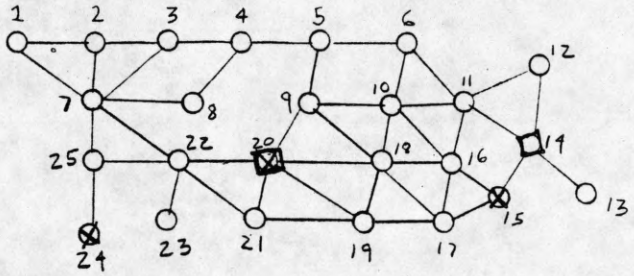


FIGURE 4.1.2 Sample outcomes of DACA using the priority indices Q^i defined in Section 2.4





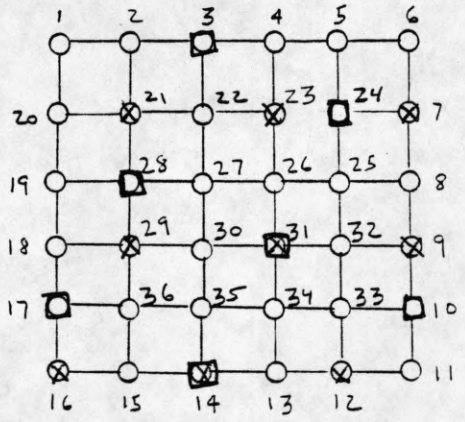
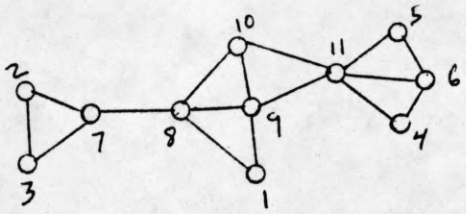


FIGURE 4.1.3 Sample outcomes of DACA with modified suppression using the priority indices Q^i defined in Section 2.4



DACA S=1,R=1 HEADS= 11,8,3 DHCA R=1 HEADS= 11,10,8,7

FIGURE 4.1.4 Example showing that DHCA tends to form more clusters than DACA

4.2 Bounds on the Probability that a Node is a Clusterhead for a Random Network

4.2.1 Model

Section 4.1 shows several examples of networks and the clusterheads generated by each algorithm. These examples are illustrative, but it is still desirable to have some characterization for any network. The probability that a node is a clusterhead is the characterization we

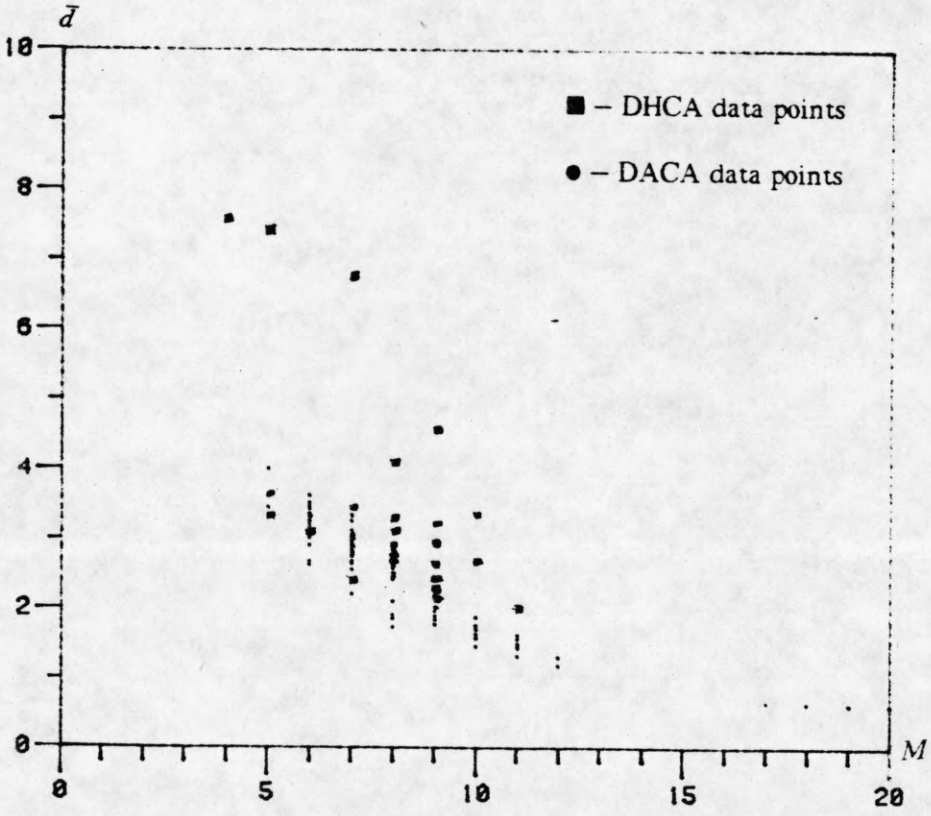


FIGURE 4.1.5 A graph of $\bar{d}$ verses M for DHCA and DACA using random Q^i selection

consider here.

Let the number of points in the plane be Poisson with parameter λ. Let Q^i for each node i in the network be independent of all else and chosen uniformly on [0,1]. If two nodes are less than t units away, where t is the transmission radius, then there is a link between the two nodes.

4.2.2 Analytical Bounds

Fix a node i and let Z be the event that, at the start of the algorithm, a node i has Q^i greater than Q^j for every node $j \neq i$ in C_i^K. Let C_i^S be all the nodes in the circle of area A centered on node i(i.e., $A \approx \pi(St)^2$). P(Z) provides a lower bound for the probability that a node i is a clusterhead for DHCA.

The probability of the event Z given Q^i is as follows.

$$P(Z/Q^i)=\sum_{k=0}^{\infty}(Q^i\lambda A)^k \frac{e^{-\lambda A}}{k!}$$

Manipulations yield

$$P(Z/Q^i)=e^{-\lambda A(1-Q^i)}.$$

Integrating out the conditioning yields

$$P(Z) = \frac{1}{\lambda A}(1-e^{-\lambda A}). \qquad (1)$$

If CH is the event that a node is a clusterhead, then

$$P(CH) \geq P(Z).$$

Fix a node i and let D be the event that, at the start of the algorithm, Q^i is smaller than Q^j for each node j in C_i^K and that node i is not the only node in C_i^K. The event D allows us to upper bound as follows the probability that a node is a clusterhead for DHCA, specifically,

$$1-P(D) \geq P(CH).$$

Proceeding as before,

$$P(D/Q^i)=\sum_{k=1}^{\infty}((1-Q^i)\lambda A)^k \frac{e^{-\lambda A}}{k!}.$$

Manipulation and removal of the conditioning yields the final result

$$P(D) = \frac{1}{\lambda A} - (1+\frac{1}{\lambda A})e^{-\lambda A}.$$

For DHCA, another way to guarantee that a node i is not a clusterhead is to have nodes j with Q^j larger than Q^i fall into each of the three shaded regions B of Figure 4.2.2.1. Let the above event be E.

The probability that all nodes in B have a smaller Q^j than Q^i is the same as equation 1 except that B replaces A in the expression. Thus,

$$P(E) = \left| 1-\frac{1}{\lambda B}(1-e^{-\lambda B}) \right|^3.$$

where $B \approx .057A$.

Putting all the bounds together yields

$$1-max(P(D),P(E)) \geq P(CH) \geq P(Z)$$

for DHCA. Note that the bounds are functions only of the expected number of nodes in a cluster, $\overline{N} = \lambda A$.

Figure 4.2.2.2 shows the bounds as a function of $\overline{N}$.

4.2.3 Simulation

The bounds given are loose in the range of interest, so we performed a simulation to see if the upper bound or lower bound is tighter. In each simulation run, points were scattered

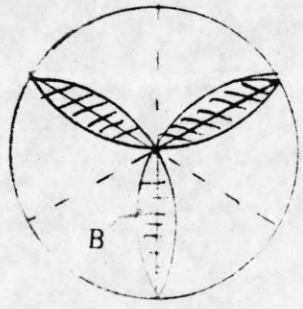


FIGURE 4.2.2.1 A node j with Q^j greater than Q^i must fall into each of the shaded regions.

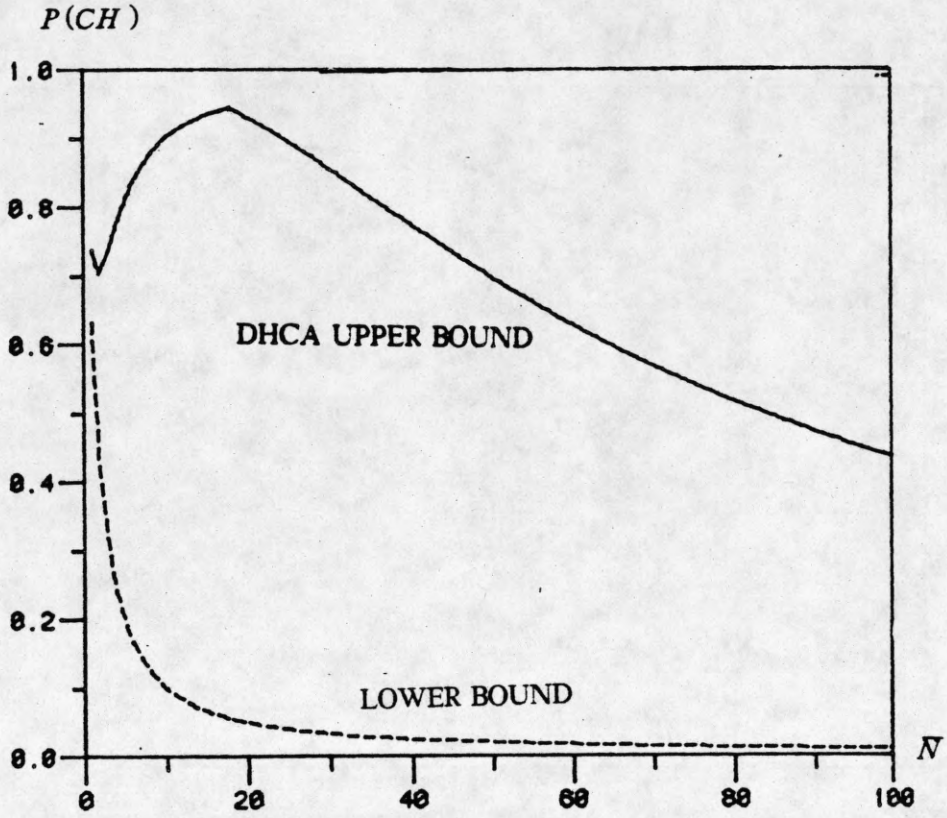


FIGURE 4.2.2.2 Bounds on the probability that a node is a clusterhead

uniformly in a unit square and clusterheads were determined according to the appropriate algorithm. In finding data points for large $\bar{N}$ we were careful to increase λ and not A, since increasing A would cause edge effects to alter the results. Figure 4.2.3.1 shows the results of the simulation. The results confirm the claim that DHCA tends to form more clusterheads for a given S than DACA.

4.2.4 Comments on Bounds

Exactly how these bounds correspond to the algorithms described varies with the parameters of the algorithm. If we are considering DHCA with R=1 or DACA with R=1 and $S=1$, then the bounds are good since, in the bound, $A = \pi t^2$, where t is the transmission radius. However, if $R \neq 1$ for either algorithm, then it is not generally true that $A = \pi(Rt)^2$.

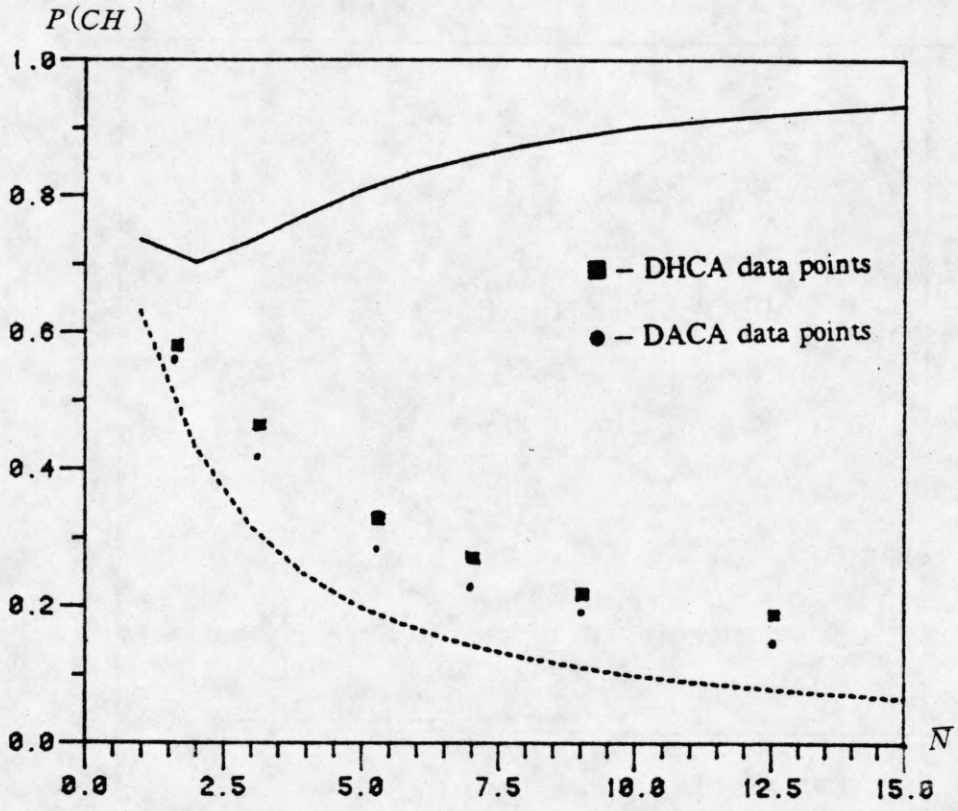


FIGURE 4.2.3.1 Simulation results on the probability that a node is a clusterhead

To see why, consider the situation depicted in Figure 4.2.4.1. Nodes a and b are within 3r of each other but are not connected. According to the bounds, only one of the two would become a clusterhead, but in reality both could become clusterheads. With large $\bar{N}$, the depicted situation occurs with probability 0. This leads us to believe that when $R \neq 1$ *or* $S \neq 1$,

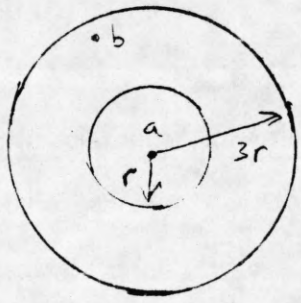


FIGURE 4.2.4.1 A situation where bounds would be inaccurate

the bounds are realistic for large $\bar{N}$. For DACA and large $\bar{N}$, $A \approx \pi(St)^2$, and for DHCA and large $\bar{N}$, $A \approx \pi(Rt)^2$.

4.3 Comparison of Communication Costs

The previous analysis shows that DACA is better than DHCA in some respects. However, it is clear that DACA has a higher communication cost than DHCA. We confirm this statement with the following analysis. Section 4.3.1 establishes an upper bound on the number of bits transmitted in DACA, and Section 4.3.2 establishes an upper bound on the number of bits transmitted in DHCA.

4.3.1 Communication Cost for DACA

In DACA, each node i generates one message. M nodes generate a C.O[i,i,T(i,i)] message, and $N-M$ nodes generate a Q.U[i,HC] message. Each Q.U[i,HC] message contains $log(N) + log(K)$ bits, and if generated by node i, is transmitted no more than $|C_i^K|$ times. So the cost for all Q.U[ ] messages passed through the network is bounded as follows:

$$\text{COST[Q.U[ ]} \leq (N-M)\max_i |C_i^K| (log(N)+log(K)) \text{ bits}$$

The maximum is over all nodes i that are not clusterheads.

Each C.O[ ] message generated contains a T(CID,CID). This table, as it is passed from node to node in C_{CID}^R, contains no more than $|C_{CID}^R|$ entries. Each entry consists of $2log(N) + log(2R)$ bits, N bits for the destination, N bits for the next node on the path and 2R bits for the distance. Also, $2log(N)$ bits are needed to identify from where the message originated(CID) and from where the message was last relayed(j). At worst, each table would be transmitted $|C_{CID}^R|$ times. Combining these facts, we find that the total communication cost for all C.O[i,i,T(i,i)] messages passed through the network is bounded as follows.

$$\text{COST[C.O[ ]} \leq M \left| \max_j |C_j^R|^2 (2log(N)+log(2R))+2logN \right| \text{ bits}$$

The maximum is over all nodes j that are clusterheads.

4.3.2 Communication Cost for DHCA

Each node i, after finding the node j in C_i^R with the largest Q^j, sends out a message to the node j that it has chosen as a clusterhead. This message has $log(N)$ bits and is transmitted at most R times, since the clusterhead is at most R hops away. At worst, $N-1$ nodes in the network would send out such a message. So, an upper bound on the total communication cost for all the C.M[ ] messages passed through the network is bounded as follows.

$$\text{COST[C.M[ ]} \leq (N-1)R \log(N) \text{ bits}$$

Each of the M clusterheads generates a message containing T(CID,CID). The bound for the number of bits needed for all the tables transmitted is the same as presented in Section 4.3.1, namely,

$$\text{COST[C.O[ ]} \leq M \max_j \left|C_j^R\right|^2 (2log(N)+log(2R))+2logN \text{ bits.}$$

Again, the maximum is over all nodes j that are clusterheads.

4.3.3 Comparison of Costs

From the above we see that the main difference in communication cost is in finding the clusterheads and not in passing the tables. In general $R << \max_i \left|C_i^K\right|$, so the cost of finding the heads in DHCA is less than in DACA.

CHAPTER 5

CONCLUSION

In this paper we presented a distributed asynchronous algorithm for forming overlapping clusters on a network with a dynamic topology. The algorithm's properties were presented, and the algorithm was compared to another algorithm designed to perform a similar task. The algorithm of Chapter 2(DACA) performs better than that of Chapter 4(DHCA) in many ways, yet the former has a higher communication cost than the latter.

The communication cost is important in determining how well these algorithms perform in a dynamic environment. If the traffic delay is not too large and node mobility is not to high, then the algorithms should perform well. However, if the traffic delay is large and the node mobility is high, the clusters may be obsolete by the time they form. This aspect of performance is best analyzed by direct simulation of the desired implementation.

REFERENCES

[1] L. Kleinrock and F. Kamoun, Hierarchical Routing for Large Networks- Performance Evaluation and Optimization, Computer Networks, vol 1(1977),pp. 155-174.

[2] N. Shacham and K. Kembla, An Architecture for Large Packet Radio Networks and Some Implementation Considerations, SRNTN 11, October 1983.

[3] N. Shacham , Organization of Dynamic Radio Networks by Overlapping Clusters: Architecture Considerations and Optimization, SRNTN 17, April 1984.

[4] D. Baker and A. Ephremides, The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm, IEEE Transactions on Communications, vol COM-29,no. 1, pp. 1694-1701, November 1981.

[5] J. Burchfiel et al., Clustering Algorithms for Adaptive Hierarchical Organization of Large Networks, SRNTN 5, October 1983.

[6] A. Segall, Distributed Network Protocols, IEEE Transactions on Information Theory, Vol IT-29 no. 1,pp. 23-35, January 1983.

[7] C.C. Gotlieb and S. Kuman, Semantic Clustering of Index Terms, Journal of the ACM, Vol 15, pp. 493-513, 1968.