A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks

Suman Banerjee, Samir Khuller

Abstract—**In this paper we present a clustering scheme to create a hierarchical control structure for multi-hop wireless networks. A cluster is defined as a subset of vertices, whose induced graph is connected. In addition, a cluster is required to obey certain constraints that are useful for management and scalability of the hierarchy. All these constraints cannot be met simultaneously for general graphs, but we show how such a clustering can be obtained for wireless network topologies. Finally, we present an efficient distributed implementation of our clustering algorithm for a set of wireless nodes to create the set of desired clusters.**

Keywords—**Clustering, Ad-hoc networks, Wireless networks, Sensor networks, Hierarchy**

I. Introduction

RAPID advances in hardware design have greatly reduced cost, size and the power requirements of network elements. As a consequence, it is now possible to envision networks comprising of a large number of such small devices. In the Smart Dust project at UC Berkeley [1] and the Wireless Integrated Network Sensors (WINS) project [1] at UCLA researchers are attempting to create a wireless technology, where a large number of mobile devices, with wireless communication capability, can be rapidly deployed and organized into a functional network.

Hierarchical structures have been used to provide scalable solutions in many large networking systems that have been designed [2], [3]. For networks composed of a large number of small, possibly mobile, wireless devices, a static manual configuration would not be a practical solution for creating such hierarchies. In this paper, we focus on the mechanisms required for rapid self-assembly of a potentially large number of such devices. More specifically, we present the design and implementation of an algorithm that can be used to organize these wireless nodes into clusters with a set of desirable properties.

Typically, each cluster in the network, would select a "cluster-representative" that is responsible for cluster management — this responsibility is rotated among the capable nodes of the cluster for load balancing and fault tolerance.

A. Target Environment

While our clustering scheme can be applied to many networking scenarios, our target environment is primarily wireless sensor networks [4], and we exploit certain properties of these networks to make our clustering mechanism efficient in this environment. These networks comprise of a set of sensor nodes scattered arbitrarily over some region. The sensor nodes gather data from the environment and can perform various kinds of activities depending on the applications — which include but is not limited to, collaborative processing of the sensor data to produce an aggregate view of the environment, re-distributing sensor information within the sensor network, or to other remote sites, and performing synchronized actions based on the sensor data gathered. Such wireless networks can be used to create "smart spaces", which can be remotely controlled, monitored as well as adapted for emerging needs.

S. Banerjee and S. Khuller are with the Department of Computer Science, University of Maryland at College Park. Email : {suman,samir}@cs.umd.edu. S. Khuller is supported by NSF Award CCR-9820965.

[1] http://www.janet.ucla.edu/WINS

B. Applicability

The clustering scheme provides an useful service that can be leveraged by different applications to achieve scalability. For example, it can be used to scale a service location and discovery mechanism by distributing the necessary state management to be localized within each cluster. Such a clustering-based technique has been proposed to provide location management of devices for QoS support [5]. Hierarchies based on clustering have also been useful to define scalable routing solutions for multi-hop wireless networks [6], [7], [8] and [9].

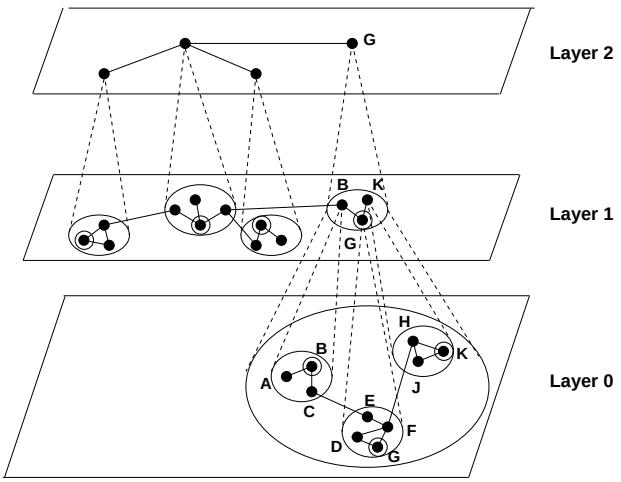

Fig. 1. An example of a three layer hierarchy

The design of our clustering scheme is motivated by the need to generate an applicable hierarchy for multi-hop wireless environment as defined in the Multi-hop Mobile Wireless Network (MMWN) architecture [5]. Such an architecture may be used to implement different services in a distributed and scalable manner. In this architecture, wireless nodes are either *switches* or *endpoints*. Only switches can route packets, but both switches and endpoints can be the source or the destination of data. In wireless sensor networks, all sensor devices deployed will be identical, and hence we treat all nodes as switches, by MMWN terminology. Switches are expected to autonomously group themselves into clusters, each of which functions as a multi-hop packet radio network. A hierarchical control structure is illustrated in Figure 1 with the nodes organized into different *lay-*

ers. An instance of the clustering scheme operates at each of the layers to create a set of clusters at that layer. All nodes in the network are joined to the lowest layer (Layer 0). Three of the clusters of Layer 0 are shown in the figure. Nodes B, G and K are the cluster representatives of these clusters. The representatives of the clusters in a layer join the layer immediately above. The instance of the clustering scheme operating at Layer 1, has placed these nodes (B, G and K) into one cluster. Node G is the representative of this cluster at Layer 1, and hence is also present in Layer 2 (the highest layer in this example). Connectivity between adjacent clusters is provided by *virtual gateways*, that essentially are pairs of peer switches, in the neighboring clusters. Each virtual gateway comprises of multiple such peer pairs. Other details on routing and addressing schemes within and outside the cluster may be found in [5].

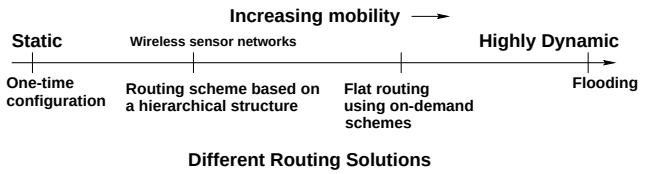

Fig. 2. A routing scheme using a clustering-based hierarchical structure would be most useful in a slowly changing mobility domain

The MMWN architecture is "not currently designed to operate effectively in a network comprised of highly mobile nodes" [5]. Hence, a routing scheme created using the clustering-based hierarchical control infrastructure, like MMWN, is most effective in a slower mobility domain than a flat routing scheme using on-demand routing solutions for highly mobile environments like Dynamic Source Routing (DSR) [10], Ad-hoc On-Demand Distance Vector routing protocol (AODV) [11] and Temporally Ordered Routing Algorithm (TORA) [12], as shown in Figure 2, but would have the benefits of scalability of a hierarchical scheme.

We hypothesize that topology changes in wireless sensor networks will also be slow and infrequent. Once a cloud of sensor devices is deployed, they would mostly stay stationary. New nodes will occasionally join the network, by drifting into the vicinity of the existing network, or some existing nodes will drift away or disappear (e.g., due to loss of power). The clustering scheme needs to maintain clusters across such topology changes and we address such issues in Section IV.

C. Desired goals of the clustering scheme

To create the hierarchical control structure described above, we postulate the following desirable properties that should be present in the clustering mechanism that runs at each layer of the hierarchy. Similar clustering goals have been specified in related clustering work for wireless networks in [13], [14].

- *Each cluster is connected.* This is an obvious requirement to localize and restrict cluster traffic to within the cluster.
- *All clusters should have a minimum and maximum size constraint.* Typically a cluster member maintains complete state information about all other members within its cluster. Hence, a maximum cluster size constraint limits per cluster state to within what can be efficiently maintained by the cluster members. Ideally, we want all clusters to be of the same size, so that no cluster is overburdened, or under-burdened with processing and storage requirements of cluster maintenance. Small clusters are inefficient use of resources at the nodes, while large clusters increase the overhead. For ease of the clustering scheme design, we set the minimum cluster size to be half the maximum size.

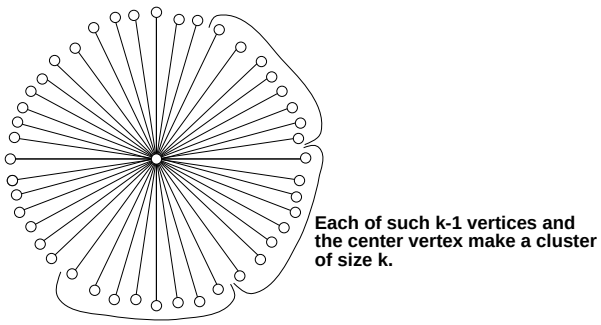

Fig. 3. Clustering in a star graph with one central vertex and $n-1$ radial vertices

- *A node in any layer of the hierarchy belongs to a constant number of clusters in that layer.* Ideally, we would want a node to belong to only a single cluster in a layer. However, it is apparent, that for connected clusters with sizes bounded as described in the previous goal, such a requirement cannot always be met. Some nodes in the network that have very high degree, might need to be included in multiple clusters. For example, in the star graph (shown in Figure 3), any cluster that has size greater than one, needs to include the central node to maintain cluster connectivity. The central node in the star graph will therefore, belong to each cluster in the network. However, we are able to leverage special properties of the communication graph of wireless networks to guarantee a small constant upperbound for the number of clusters to which a node belongs to [2].
- *Two clusters (in any layer) should have low overlap.* All nodes common to two clusters, will have to maintain cluster state and carry intra-cluster traffic for both the clusters. Ideally, clusters should have no overlap. But as discussed above, since in some topologies, some nodes might have to belong to more than a single cluster, zero overlap is not possible. Our clustering scheme guarantees that no two clusters in a layer will have an overlap of more than one node.
- *Clusters should be stable across node mobility.* The clustering scheme should scalably adapt to new nodes drifting into the network, existing nodes disappearing from the network (due to power loss) and other node such node migration scenarios. Such events should cause only a very localized re-clustering (if at all necessary) so that the previous desired goals are maintained. We describe our cluster maintenance scheme for the distributed implementation of clustering that handles node mobility in Section IV-B.

D. Main contributions

In this paper we propose a clustering scheme to create a layered hierarchy, similar to that desired in MMWN [5], for wireless networks. We define our clustering problem in a graph theoretic framework, and present an efficient distributed solution

[2] It should be noted, that in our clustering scheme, most of the nodes belong to only a single cluster in each layer and the few that do not are within the small constant bound as described.

that meets all desirable properties mentioned earlier. For arbitrary graph topologies, sometimes no solution may exist that can satisfy all the requirements of a desirable solution. But in wireless network topologies, properties of the underlying communication graphs may be exploited to achieve desired solutions, as we demonstrate in this paper.

The rest of the paper is structured as follows. We pose our problem in a graph theoretic framework in Section II. We discuss the clustering algorithm in Section III. In Section IV, we demonstrate how our clustering algorithm can be implemented in a distributed environment as the sensor network. We evaluate our clustering scheme through simulations in Section V. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Problem Statement

We first define a generic network clustering problem as follows: Given an undirected graph $G = (V, E)$, and a positive integer k, such that, $1 \leq k \leq |V|$, for each *connected* component, find a collection of subsets (clusters), $V_1, \ldots, V_l$ of V, so that the following conditions are met.

1. $\cup_{i=1}^{\ell} V_i = V$. All vertices are part of some cluster.
2. $G[V_i]$, the subgraph of G induced by V_i is connected.
3. $k \leq |V_i| < 2k$. This is the size bound for the clusters.
4. $|V_i \cap V_j| \sim O(1)$. Two clusters should have up to a small constant number of common vertices. We show, later in the section, why all clusters cannot be guaranteed to be non-overlapping and yet meet the other requirements. [3]
5. $|S(v)| \sim O(1)$, where $S(v) = \{V_i | v \in V_i\}$, i.e.. a vertex belongs to a constant number of clusters.

We note that there may not be a feasible solution to the above problem for any general graph. Requirement (5) would be violated in a star graph (a graph with a single center vertex and $n-1$ radial vertices, and there is an edge between the center vertex and each radial vertex as in Figure 3). For $k \geq 2$, any cluster in the star graph would include the center vertex, for the cluster to be connected. Hence, for the center vertex, c, we would have $|S(c)| \sim O(\frac{n}{k})$, violating requirement (5) of the problem statement. However, the underlying graph structure for a network of wireless nodes has certain useful properties that can be exploited. A wireless node A, can communicate with another node B, if and only if, B lies within the transmission radius, R_A, of node A. The underlying graph, in this case, would have a directed edge $A \rightarrow B$. For our algorithm, we only consider bi-directional edges. So, a valid edge in the graph reflects the fact that both the nodes are within each other's transmission range, i.e., $d(A, B)$, the distance between the nodes A and B is at most $\min(R_A, R_B)$, for them to have an edge in the graph. This is in conformance with the assumptions made for most MAC protocols for wireless environments, including MACA [15], MACAW [16], IEEE standard 802.11 [17], FAMA [18] and RIMA [19].

We first consider the case when all nodes in the network have the same transmission range. In this case, the underlying communication graph, is a *Unit Disk graph* – defined in [20], [21] in terms of "distance" or "proximity" models, which consist of a value $R \geq 0$ and an embedding of the vertices in the plane, such that (u, v) is an edge if and only if $d(u, v) \leq R$. Here, R is the common transmission radius of all wireless nodes. For such graphs, it can be seen that if a node has many neighbors, i.e., a vertex has very high degree, then all these vertices have to be within its transmission radius. These neighboring nodes will, therefore, be relatively close to each other. As a consequence many of these neighboring nodes will be within transmission range of each other and will have edges between themselves in the communication graph. This would prevent the communication graph from having dense "star-like" components embedded in them. This is proved rigorously in Section III-C. We exploit this feature to guarantee that each vertex in the graph is in at most three clusters [4]. This is not possible in general graph topologies, as shown before.

Since the transmission range depends on the power available at the node, in general, for a homogeneous set of sensor nodes, the transmission radii would be close to each other. We also consider the case where different nodes may have different transmission radii. For such scenario, our clustering algorithm would guarantee that no node is a member of more than $O(\log(\frac{R_{\max}}{R_{\min}}))$ clusters, where $R_{\max}$ and $R_{\min}$ are the maximum and minimum transmission radii respectively. We use the term *Bounded Disk graph* to classify these underlying topologies [5]. Hence, the algorithm does not violate Requirement (5) even when orders of magnitude difference exist between the transmission power of the nodes.

If there are nodes with very small transmission radii, then the bound on $|S(v)|$ may be large, but in general, nodes with very small transmission radii would be nearly disconnected from the rest of the network, and can be considered "dead" for all practical purposes.

For the rest of the paper, we will focus only on communication graphs that are either unit disk graphs (for wireless nodes with identical transmission radii R) or bounded disk graphs (where the transmission radii of the nodes are bounded between $R_{\min}$ and $R_{\max}$) [6]. Even for these graphs, to satisfy Requirements (2) and (3), may lead to violation of Requirements (4) and (5), as shown below.

Requirement (4) would be violated even in unit disk graphs as shown in Fig. 4 for any clustering algorithm. The total number of vertices in the tree is n. The zones A, B and C each have $0.3n$ vertices, the segments L1, L2 and L3 have $0.03n$ vertices, while the segment L4 has $0.01n$ vertices. If k, the lower bound of the size of a cluster, is set at $0.45n$, any cluster will have vertices from at least two of the zones A, B and C. Also, from Condition (2), the upper bound of the size of a cluster is $0.9n$, and so there must be at least two clusters to cover all the vertices. Let us choose two such clusters, C_1 and C_2, and let C_1 have vertices

[3] Our clustering algorithm actually guarantees that $|V_i \cap V_j| \leq 1$, i.e. two clusters overlap in not more than one vertex.

[4] In this paper, we only illustrate the results for two-dimensional topologies. The scheme works for any D-dimensional space, with the upper bound of $S(v)$ being a function of *only* the dimensionality, D.

[5] From our algorithm and its proof, it will be apparent that one can construct some specific Bounded Disk Graphs, for which some nodes will necessarily belong to $O(\log(\frac{R_{\max}}{R_{\min}}))$ clusters, to meet the conditions of the problem statement.

[6] Our clustering technique can also be applied to general graph topologies, if we remove the Requirement (5), that each node belong to a constant number of clusters. The upper bound on the number of clusters a node belongs to, *for general graphs*, is the maximum degree of a node, which is usually low for many topologies.

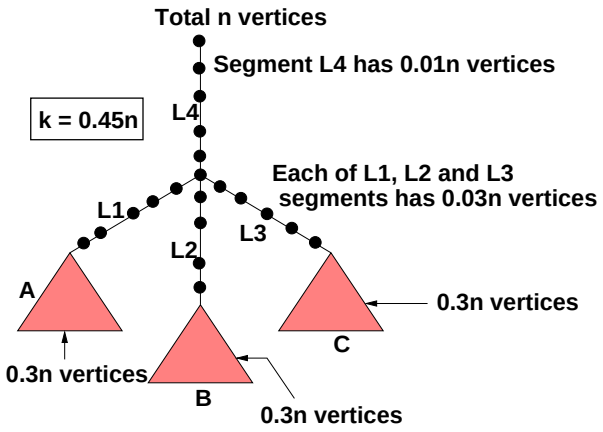

Fig. 4. Violation of constant sharing between clusters (with size parameter k), even for disk graphs

from zones A and B (and maybe some other vertices too) and let C_2 have vertices from zones B and C (and some other vertices as might be necessary). To keep each cluster connected, we must have all vertices in segment L2 belong to both the clusters. This would mean $|C_1| \cap |C_2| \geq |L2| = 0.03n$, i.e., overlap, which is linear in the number of vertices.

Hence, we modify our Requirement (3) as follows :

3(a). $\forall i, |V_i| < 2k$.
3(b). $\forall i$ except one $k \leq |V_i|$, we allow one single cluster in the entire graph to have size smaller than k.

Under such a relaxation, it is possible to cluster the graph in Fig. 4, by making C_1 include zones A and B and the segments L1, L2 and L4, and C_2 include zone C and segment L3.

Hence, our exact problem can be refined as stated below :
Given a disk graph $G = (V, E)$, and a positive integer, k, such that, $1 \leq k \leq |V|$, for each connected component of G, find a collection of subsets $V_1, \ldots, V_l$ of V, so that

1. $\cup_{i=1}^{l} V_i = V$.
2. $G[V_i]$, the subgraph of G induced by the vertices V_i, is connected.
3. The sizes of the subsets are bounded as follows :
 (a) $\forall i, |V_i| < 2k$.
 (b) $\forall i($ except one $) k \leq |V_i|$, i.e., we allow one single cluster in the entire graph to be smaller than k.
4. $|V_i \cap V_j| \sim O(1)$
5. $|S(v)| \sim O(1)$, where $S(v) = \{V_i | v \in V_i\}$, i.e.. a vertex belongs to a constant number of subsets.

Next, we state and prove the algorithm, first for unit disk graphs, when all nodes have the same transmission radius, R. Subsequently, we show how the same algorithm can be applied for bounded disk graphs, where nodes have varying transmission radii, but with Requirement (5) modified as $|S(v)| \sim O(\log(\frac{R_{\max}}{R_{\min}}))$.

III. Solution

We first outline the algorithm as it applies to a connected graph. In this section we first outline the clustering algorithm as it applies to a connected graph (or to each connected component of the graph, if it is not connected).

A. Overview of the Solution

The algorithm proceeds by finding a (rooted) spanning tree of the graph. One could use a Breadth-First-Search tree, or any other tree. The main advantage of a BFS tree is that it has a radius, which is bounded by diameter of the graph.

The algorithm runs in linear time. Let T be this rooted spanning tree, and $T(v)$ denote the subtree of T rooted at vertex v. We use $|T(v)|$ to denote the size of the subtree rooted at v. Let $C(v)$ be the set of children of v in T.

We assume that $|V| \geq 2k$, else we can treat the entire graph as one cluster. First we identify a node u such that $|T(u)| \geq k$ such that for each $v \in C(u)$ we have $|T(v)| < k$. It is clear that such a node always exists. Let $C(u)$ consist of ℓ nodes $v_1, \ldots, v_\ell$.

If we do not worry about Requirement (5), then it is easy to create a set of clusters from the subtree $T(u)$. This can be achieved by partitioning the set of subtrees $\{T(v_1), \ldots, T(v_p)\}$ where $v_1, \ldots, v_p$ are the children of u in the tree, appropriately as described below. The partitions are, by definition, disjoint and each partition consists of a set of subtrees, such that the number of all vertices in the subtrees comprising the partition lies between $k-1$ and $2k-2$. Each partition is created by adding subtrees sequentially to it until the size lies between $k-1$ and $2k-2$. Addition of a single subtree cannot increase the partition size to more than $2k-2$, since each subtree has size at most $k-1$. Only one single partition (the last partition) may have size less than $k-1$. The vertex u is added to each of the partitions to ensure that they are connected. All the partitions are connected and each partition that now meets the desired size bound between k and $2k-1$ (all partitions except possibly the last one) are defined to be clusters. All the vertices put into clusters are deleted from the tree. Vertex u is not deleted if the last partition did not meet the size bounds and hence, not made a cluster. These steps are repeated to create all clusters.

In fact, this algorithm can be implemented via a post-order traversal of T. When we are visiting a vertex, we can check the size of the subtree rooted at that vertex. If the subtree has size $\geq k$ then we can trigger the above scheme. Once we output a set of clusters and consequently, delete the vertices that belong to these clusters, we can update the size of the current subtree and continue with the post-order traversal at the parent of this vertex.

The main problem with the above scheme is that u may belong to many clusters (in the worst case, proportional to the degree of u even though this bound is unlikely to be achieved in practice). We will now make use of the properties of the disk graphs that arise in this application to avoid this problem.

We will prove that for any six vertices that are adjacent to u, there exists a pair of them with an edge between them. Using these edges, we can connect the subtrees rooted at children of u to each other to create the clusters, *without* using vertex u (except in a small number of clusters). In the pseudo-code description of the procedure GraphCluster, this can be observed as the second condition to enter the **while** loop at Line 7. This will guarantee that vertex u belongs to at most a constant number of clusters. It is easy to see that the intersection of clusters will also have at most one vertex.

B. Detailed Description of Algorithm

We use the following notation :

- T : A BFS tree of graph G.
- *root*(T) : Root of the BFS tree.
- $T(x)$: Subtree of T, rooted at vertex x.
- *ClusterSet* : The set of clusters created by the algorithm.
- *UnpChildren* : Variable used to store the set of remaining children (i.e. that has not been deleted) that are yet to be processed at a vertex.
- *PartialClusterSet* : Set of temporary clusters that have size $< k$.
- Empty set is denoted by $\perp$.

Proc. 1 : GRAPHCLUSTER(G, k)

```
1:  T ← BFS tree of G;  ClusterSet ← ⊥
2:  for u ∈ G, in post-order traversal of T do
3:    if (|T(u)| ≥ k) then
        {ASSERT : |T(v)| < k, v ∈ Children(u)}
4:      PartialClusterSet ← ⊥
        UnpChildren ← Children(u)
5:      while ∃v ∈ UnpChildren do
6:        TempCluster ← T(v)
          Remove v from UnpChildren
7:        while (|TempCluster| < k) ∧
            (∃x ∈ UnpChildren, s.t. x has an
            edge to w ∈ Children(u) ∩ TempCluster) do
8:          TempCluster ←
              TempCluster ∪ T(x)
9:          Remove x from UnpChildren
10:       end while
11:       if (|TempCluster| ≥ k) then
12:         ClusterSet ←
              ClusterSet ∪ {TempCluster}
13:         Remove all subtrees in TempCluster
14:       else
            {ASSERT : No such x is found}
15:         PartialClusterSet ←
              PartialClusterSet ∪ TempCluster
16:       end if
17:     end while
18:     MERGEPARTIALCLUSTERS(u, k,
            PartialClusterSet, ClusterSet)
19:     if (Children(u) = ⊥) ∧
          (u has been assigned to some cluster) then
20:       Remove u from the tree
21:     end if
22:   end if
23: end for
24: if PartialClusterSet ≠ ⊥ then
      {ASSERT : |PartialClusterSet| = 1}
      {Let P ∈ PartialClusterSet}
25:   ClusterSet ← ClusterSet ∪ {P ∪ {root(T)}}
26: end if
```

The algorithm creates a BFS tree and traverses the the tree in post-order. Figure 5 shows the processing for a vertex, u, in the tree. The vertex v has already been visited. Since $|T(v)| \geq k$,

Proc. 2 : MERGEPARTIALCLUSTERS(u, k, P, *ClusterSet*)

```
1:  C ← ⊥
2:  while (P ≠ ⊥) do
3:    Pick an arbitrary partial cluster, p from P
4:    C ← C ∪ p;  Remove p from P
5:    if (|C| ≥ k) then
6:      ClusterSet ← ClusterSet ∪ {C ∪ {u}}
7:      Remove all subtrees in C;  C ← ⊥
8:    end if
9:  end while
```

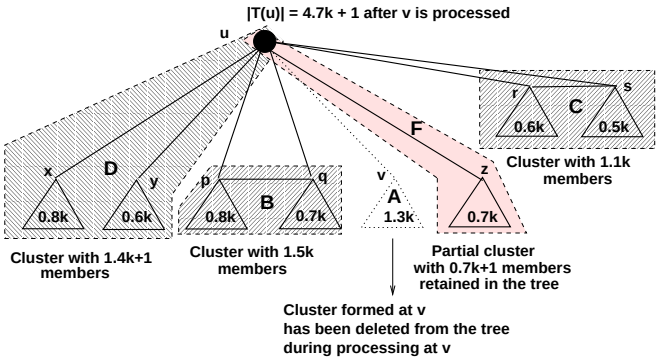

Fig. 5. Example operation at a vertex u for the GRAPHCLUSTER procedure

cluster, A, has been created rooted at v and deleted from the tree (Line 4 of the GRAPHCLUSTER procedure). Hence, at the time u is being processed, there are seven remaining children of u : x, y, p, q, z, r, s and the remaining subtree size of $T(u)$ is $4.7k+1$, that included u. The two clusters, B and C are formed first without including u in any of these clusters (They both satisfy the **if** condition on Line 11 of procedure GRAPHCLUSTER), and are placed into clusters on Line 12. When MERGEPARTIALCLUSTERS is called (Line 18), there are three partial clusters, the subtrees rooted at vertices x, y and z. In MERGEPARTIALCLUSTERS, the cluster D is formed using the partial clusters from the subtrees rooted at x and y, and vertex u is used to connect the two subtrees, which do not share a common edge. Finally, a single partial cluster (the subtree rooted at vertex z) is left. Vertex u is added to this partial cluster to form the partial cluster F, and this is the only subtree that remains in the tree, and each of the other vertices have been placed into some cluster and deleted. Hence, when processing is done at the parent vertex of u, the subtree rooted at u comprises of only the vertices in the partial cluster, F.

If a single partial cluster is left in the tree after the entire post-order traversal, it is made into a cluster of size less than k in Lines 24-26 of procedure GRAPHCLUSTER.

C. Proof of Correctness

We now outline the proof of correctness of the algorithm described above. We use the following standard graph-theoretic terminology :

- The *Neighborhood*, $N(u)$, of a vertex u, is the set of vertices that have an edge to the vertex u in the graph.
- A set of vertices that have no edges between any pair of vertices in the set is termed an *Independent Set*.

• The *Maximum Independent Set,* MIS, of vertices is an independent set with maximum size.

LEMMA III.1. *For unit disk graphs, the maximum independent set,*(MIS)*, in the neighborhood,* $N(u)$*, of a vertex* u *has at most 5 vertices.*

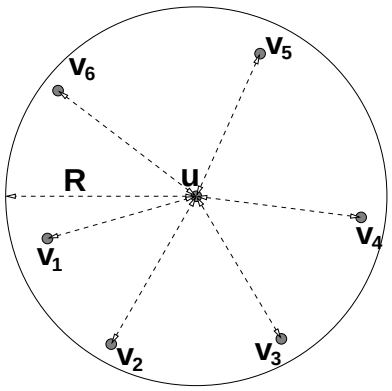

Fig. 6. Every six neighbors of a vertex have at least one edge

Proof. The proof follows from simple geometric arguments. Let the distance parameter of the unit disk graph be R. Consider a vertex, u, s.t. $|N(u)| \geq 6$. Let us assume that there is an Independent Set of size six in the neighborhood, $N(u)$. Let these six vertices be $v_1, \ldots, v_6$ with the vertex indices labeled in a cyclic order as shown in Figure 6. Since, $v_i \in N(u), d(u, v_i) \leq R$. Consider vertices v_i and v_j, such that they are successive vertices from this Independent Set in the cyclic order, i.e. $j = i(mod6)+1$. If $(v_i, v_j) \notin E$, then $d(v_i, v_j) > R$. Also, $R \geq d(u, v_i)$ and $R \geq d(u, v_j)$. So, $d(v_i, v_j) > d(u, v_i)$ and $d(u, v_j)$. Hence, in $\triangle u v_i v_j$, (v_i, v_j) is the largest side, and so $\angle u$ is the largest angle, which must be $> \frac{\pi}{3}$. Hence, $\sum_{i=1}^{6} \angle v_i u v_j > 6 \times \frac{\pi}{3} = 2\pi$, a contradiction. Hence, $\exists i$, such that $d(v_i, v_j) \leq R$, i.e., $(v_i, v_j) \in E$.

OBSERVATION 1. *When the algorithm terminates, each vertex is part of some cluster.*

OBSERVATION 2. *Each cluster formed by the algorithm is connected.*

OBSERVATION 3. *Only one cluster may have size* $< k$ *and all other clusters have sizes between* k *and* $2k$.

OBSERVATION 4. *Any pair of clusters has only one common vertex.*

OBSERVATION 5. *The number of partial clusters, created on exiting the* **while** *loop of Lines 5-17 of procedure* GRAPHCLUSTER*, is five.*

Proof. Each partial cluster in the *PartialClusterSet*, has at least one child of u in the tree, since *TempCluster* (Line 8 of procedure GRAPHCLUSTER), comprises of subtrees of u rooted at some children of u. A partial cluster is added to *PartialClusterSet* in line 15 of procedure GRAPHCLUSTER. Let there be at least six partial clusters $P_1, \ldots, P_6$ numbered in the sequence in which they are created. Let $v_1 \ldots v_6 \in$ *Children*(u) be vertices, in these six different partial clusters respectively. A partial cluster is added to *PartialClusterSet* if the inner **while** loop (Lines 7-10) exits when the second condition on line 7 of procedure GRAPHCLUSTER is **false**. So, at the time P_i was put in *PartialClusterSet* (Line 15 of procedure GRAPHCLUSTER), there would have been no edge from any child of u in P_i to any other unprocessed child of u. These include all children of u in partial clusters P_j, for $j > i$. In particular, for $j > i, (v_i, v_j) \notin E$, i.e. $v_1 \ldots v_6$ form an independent set of vertices. This contradicts Lemma III.1. Hence, there can be up to five partial clusters.

OBSERVATION 6. *During the processing pass (Lines 3-22) of vertex* w *in the post-order traversal of* GRAPHCLUSTER*, it is either deleted in that pass (being put in some cluster(s)) or is left in the tree, and is placed in only one more cluster during the subsequent processing pass of some ancestor* u *of* w.

Proof. Let us assume that a vertex w is not deleted during its processing pass in the post-order traversal. Then, if w is not deleted by the time some ancestor u of w is being processed, w will be part of some subtree rooted at a child of u. Each of the subtrees rooted at children of u is placed in only one single cluster and deleted from the tree, or is not assigned to any cluster during the processing pass of u. Since this is true at each ancestor of w, eventally w will be assigned to a single cluster at some ancestor and get deleted.

OBSERVATION 7. *A vertex* v*, is part of up to three distinct clusters [Requirement 4].*

Proof. A vertex that is part of multiple clusters, would have to satisfy the **if** condition in Line 3 of GRAPHCLUSTER. If this **if** condition fails, the vertex is not placed into any cluster during its processing pass of the post-order traversal, and will be placed in only a single cluster subsequently (due to Observation 6). It can be observed that all other vertices will be put into a single cluster. The vertex that satisfies the **if** condition in Line 3 would not be placed in any cluster created on Line 12 ofGRAPHCLUSTER in the same pass of the **for** loop (Lines 2-23) of the post-order traversal. These clusters comprise only of subtrees rooted at children of u in the tree. In this pass u may only be placed in the clusters created out of the partial clusters (Line 6 of MERGEPARTIALCLUSTERS called from GRAPHCLUSTER). From observation 5, there are only five partial clusters. Each partial cluster has size $< k$. Hence, at least two such partial clusters needs to be merged in Line 4 of MERGEPARTIALCLUSTERS to create a cluster in Line 6. Hence, the maximum number of clusters created out of the five partial clusters in Line 18 (of GRAPHCLUSTER) is two. If an additional partial cluster is still left in tree after processing of u in the post-order traversal, this entire partial cluster along with vertex u will be placed further in only one more cluster (Observation 6). Consequently, the vertex u can be a part of up to three clusters.

Thus, when the algorithm terminates, all the requirements for the problem statement is satisfied.

LEMMA III.2. *If* $R_{\max}$ *and* $R_{\min}$ *are the maximum and minimum radii respectively, then the maximum independent set in* $N(u)$ *has cardinality at most* $O(\log \frac{R_{\max}}{R_{\min}})$.

Proof. Let $I(u)$ be the largest independent set in the subgraph induced by $\{u\} \cup N(u)$. We define a "moat" $b(i)$ for $i \geq 0$, which is the annulus defined by circles of radii $c^i R_{\min}$ and $c^{i+1} R_{\min}$,

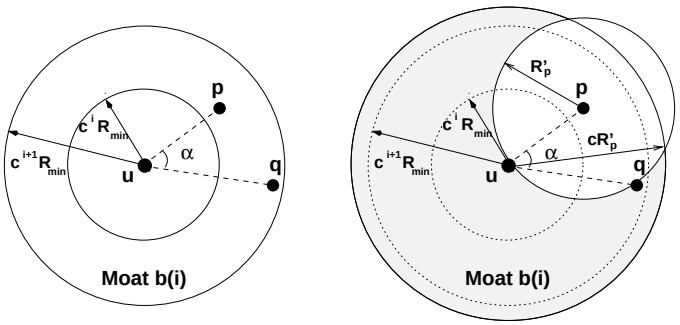

Fig. 7. Two vertices p and q in the neighborhood, $N_i(u)$ of u that are part of the maximum independent set in the subgraph induced by $N_i(u) \cup \{u\}$ have an angular separation, $\alpha > \frac{\pi}{6}$. They belong to the same moat $b(i)$, and there are $\log(\frac{R_{\max}}{R_{\min}})$ such moats.

centered at u (see Figure (7). (Note that the constant, c is chosen to be $\sqrt{3}$ as a scale factor, for ease of proof using geometric properties.) Let $N_i(u)$ be the neighbors of u that are in moat $b(i)$. Note that, any vertex, $v \in N_i(u)$, must have a transmission radius, $R_v \geq c^i R_{\min}$. This condition is needed for (u, v) to be an edge in the graph. We will prove that within each $N_i(u)$ there are at most 11 vertices in $I(u)$. Since the number of moats that contain vertices from $N(u)$ is $O(\log \frac{R_{\max}}{R_{\min}})$, the result follows.

Let p, q be two vertices that are in $N_i(u)$. Without loss of generality, let $R_p \leq R_q$. The distance between p and q is at least $\min(R_p, R_q) = R_p$ since there is no edge between p and q. We can "shrink" the circle centered at p with radius R_p until u is on the boundary of the circle (see Figure 7). This new radius satisfies $R'_p \leq R_p$ and $R'_p \geq c^i R_{\min}$. Notice that the distance from u to q is at most $c^{i+1} R_{\min} \leq c R'_p$. Draw a circle centered at u with radius cR'_p. Notice that q is inside this circle, but outside the circle centered at p with radius R'_p. This implies that q is in the crescent shaped shaded region.

Under these circumstances, the angle between p and q at u is $> \frac{\pi}{6}$, and by the same arguments as in Lemma III.1, there cannot be more than 11 vertices in the moat $b(i)$.

As a consequence of Lemma III.2, the procedure GRAPHCLUSTER would be applicable for Bounded Disk graphs, so that each vertex will be a part of $O(\log(\frac{R_{\max}}{R_{\min}}))$ clusters.

D. Algorithm Complexity

The tree computation of Line 1 GRAPHCLUSTER, takes $O(|E|)$. The computation at each vertex u, in post-order traversal, is $O(deg_T(u))$. i.e. the degree of u in the tree. Hence, the total cost for the entire post-order traversal is $\sum_u deg_T(u) = O(|V|)$. Hence, the complexity of the algorithm is $O(|E|+|V|)$.

IV. DISTRIBUTED IMPLEMENTATION

The algorithm that is described in Section III-B, is a centralized solution to the clustering problem. In the distributed scheme, each wireless node in the network runs an identical protocol. The protocol has two phases : Cluster Creation and Cluster Maintenance. The cluster creation phase of the protocol is invoked very infrequently, when the existing clustering falls below a *quality* threshold. Cluster maintenance is an inexpensive phase of the protocol that handles node mobility and other usual dynamics of the network.

A. Cluster Creation

This is a simple distributed version of the centralized GRAPHCLUSTER algorithm. It can be initiated by any node in the network (the initiator will be the root of the BFS tree). If multiple nodes initiate Cluster Creation at the same time, simple tie breaking heuristics (e.g. initiator with least ID) is imposed to allow only one instance to proceed; the rest are not propagated.

There are two parts to cluster creation : Tree Discovery and Cluster Formation. The messages for Cluster Formation are piggybacked on the messages for the Tree Discovery component.

Tree Discovery : This is a distributed implementation of creating a BFS tree. Each node, u, transmits a *tree discovery beacon*, which indicates its shortest hop-distance to the root, r. The beacon contains the following fields : $\{$*src-Id, parent-Id, root-Id, root-seq-no, root-distance*$\}$. If any neighbor, v of u on receiving this beacon, discovers a shorter path to the root through u, it will update its hop-distance to the root appropriately and will choose u to be its parent in the tree as indicated in Figure 8. As shown, node E originally at distance 3 from the root A, receives a beacon from node D, at distance 1 from the root and consequently chooses D to be its new parent. This decreases the distance of E from the root to 2.

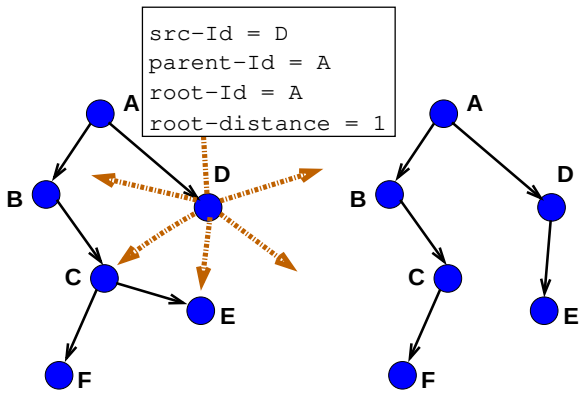

Fig. 8. Periodic tree discovery beacon transmitted by a node during the Tree Discovery part of the Cluster Creation phase to distributedly create a BFS tree. The relevant beacon fields are indicated.

The *parent-Id* field will be initially *NULL*, and change as the appropriate parent in the BFS tree is discovered. The *root-distance* field reflects the distance in hops from the root of the tree. The *root-Id* is used to distinguish between multiple simultaneous initiators of the Cluster Creation phase of which only one instance is allowed to proceed. The *root-seq-no* is used to distinguish between multiple instances of the Cluster Creation phase initiated by the same root node at different time instants. The $\{$*root-Id*, *root-seq-no*$\}$ pair, therefore, uniquely identifies a Cluster Creation phase instance.

Cluster Formation : To create the clusters on the BFS tree, each node needs to discover its subtree size and the adjacency information of each of its children in the BFS tree, as explained below. For this purpose, a *cluster formation message* is piggybacked onto the tree discovery beacon by each node and has

the following fields : $\{$*subtree-size, node-adjacency*$\}$. The subtree size information is aggregated on the tree from the leaves to the root. The subtree size at a node, u is given by $1 + \sum_{v \in Children(u)}$ *subtree-size*(v). When a node, w, detects that its subtree size has crossed the size parameter, k, it initiates cluster formation on its subtree (this is the condition tested in Line 3 of GRAPHCLUSTER). If the entire subtree $T(w)$ is of size $< 2k$ it creates one single cluster for the entire subtree, or else, it will create a set of clusters by appropriately partitioning the children subtrees into these clusters. This information is subsequently propagated down the child subtrees as *cluster assignment messages* to the relevant nodes. The partitioning of child subtrees into clusters is implemented as specified in Lines 4-21 of GRAPHCLUSTER. To do this, node w needs to know the adjacency information of its children in the tree. This is available as the neighborhood information, $N(u)$ carried in the *node-adjacency* field of the cluster formation message from each child, u. In its subsequent cluster formation messages to its parent, node w does not include all the nodes which it has assigned to different clusters. This is equivalent to the deletion of these nodes from the tree in Lines 13 and 20 of GRAPHCLUSTER and Line 7 of MERGEPARTIALCLUSTERS). An example is shown in Figure 9. The subtrees C and D rooted at node v in Figure 9

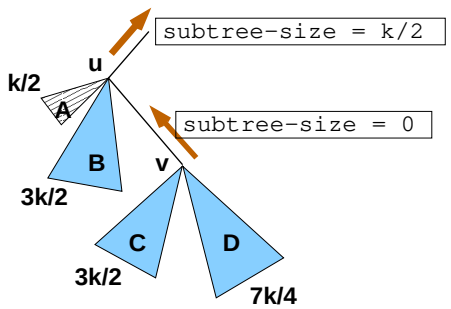

Fig. 9. Cluster formation messages are piggybacked onto the tree discovery beacons. Only the subtree-size field is shown here. The subtree size reported by a node, u, to its parent include only those subtree nodes that have not been put into any cluster at u.

has been put into two separate clusters, i.e. there is no node in the subtree of node v, which has not been put into some cluster. Hence, v reports a subtree size of zero to its parent, u, in the tree. Node u has assigned the entire subtree B into a single cluster but has not assigned subtree A to any cluster. It, therefore, reports the size of this subtree to its parent [7]. This subtree will be assigned a cluster by a node upstream of u. When a node detects that its children and their subtree sizes have not changed for the last *max-consecutive-static-subtree* messages from its children, it terminates the cluster formation phase for all the clusters that it has assigned. To do so, it sends a *terminate cluster message* down the subtrees of these clusters. This procedure happens at each node of the tree. Clusters are, therefore, created from the leaves of the tree towards the root akin to the post-order traversal of the tree. The phase terminates when all the nodes have received cluster terminate messages.

The tree discovery beacons are transmitted by each node once every P units of time, over the duration of the cluster creation phase. The period P should be chosen depending on the average connectivity of a node in the network and the bandwidth available from the wireless interface. With an average degree of four for a network of low bandwidth (100 Kbps) sensor devices, assuming the beacons are 100 bits in length, we can choose P as 40 ms to ensure that the collision probability of beacons is less than 0.1. The average number of beacons sent by a node for the entire cluster creation phase is approximately bounded by the diameter of the network. Hence, for a very large network of $10,000$ sensor nodes distributed on a square grid,with the above properties, an average sensor nodes will send about 100 beacons and the whole network will take less than 8 seconds for the entire cluster creation phase.

As the cluster creation phase ends, only the cluster information needs to be retained by the clusters. The BFS tree does not need to be maintained any further.

During the cluster creation phase, a few nodes may be missed (could be due to transient channel errors over the duration of the cluster creation phase). Such nodes will be able to join some cluster during the Cluster Maintenance phase, discussed next.

B. Cluster Maintenance

Once the cluster creation phase generates a set of clusters, the cluster maintenance phase is invoked to perform small incremental changes to the clusters to handle node mobility, as new nodes arrive and existing nodes depart from the network (battery might run out). For the Cluster Maintenance phase, we relax the upper bound of the cluster size to $3k$. Hence, clusters in the maintenance phase can have sizes between k and $3k$ [8].

New node joins : A sensor node, v, on joining the network,establishes its set of neighbors, $N(v)$. If any node $u \in N(v)$ belongs to some cluster of size $< 3k - 1$ then we add v to the cluster that u belongs to. This also ensures that we maintain connectivity in the cluster and the size requirement.

However, it is possible that all neighbors belong to clusters of size $3k - 1$. In this case, we add v to one such cluster, thus making its size, $3k$. Hence this cluster is split into two separate clusters. The splitting process involves the use of the same distributed implementation described in Cluster Creation phase. However, in this scenario, it is even simpler, since as soon as a cluster is identified to have size $\geq k$, the remaining $2k$ nodes can immediately be put into another cluster. Note, this splitting involves only the nodes in this cluster of size $3k$ being split. Hence, this splitting effort has constant communication and time requirements unlike the clustering of the entire network, for which it is proportional to the diameter of the network. It can also be observed, that once a cluster is split, in this manner, it would require at least k other nodes joining one of the clusters (in the worst case) before another split would be necessary.

Triggering re-clustering : A cluster is split into two when it reaches the $3k$ size upper bound, as a new member joins. However, it might be necessary to share a node, x, by both the new clusters created (to ensure connectivity of each cluster). As a consequence $|S(x)|$ would increase. There are cases when

[7] Node u is also included in this subtree.

[8] At the end of cluster creation, however, all the clusters had size between k and $2k$. Hence, at the end of cluster creation, if a small cluster of size $< k$ exists (there can be only one such cluster), it is merged with another cluster of size $< 2k$.

$|S(x)|$ decreases too. However, by a pathological sequence of joins by other nodes to the set of clusters $S(x)$, the value of $|S(x)|$ can increase without bounds. We consider a clustering to be of "poor" quality, when an estimated average of $|S(x)|$ exceeds a specified threshold. This triggers the cluster-creation phase to re-create a "good" clustering. This estimation can be done periodically at each node by randomly sampling a small subset of its nearby nodes. Network-wide flooding solutions are not used to gather this estimate. In the event multiple nodes trigger re-clustering simultaneously, the cluster creation phase chooses only one of these re-clustering initiators to proceed and inhibits all the others, as already described. For a relatively slowly changing network topology, like the wireless sensor networks, the gap between successive invocations of the cluster-creation phase will very high.

Existing node leaves : When a node leaves, it may cause the cluster(s) it belongs to, become disconnected. However, it may be shown (using the same arguments as in Lemma III.1 and III.2), that the number of remaining connected components of a cluster, due to a node leaving, will be bounded (using the same bounds as before for the Maximum Independent Set in the neighborhood of a single node). Any such connected component, that has its size $\geq k$, is made a cluster. Each component that has a size $< k$, will cease to be a cluster, and each node in the component will attempt to join one of its its neighboring clusters – the same mechanism of a node joining a cluster, applies for each of these nodes.

Link outages and network partitions : A link outage does not alter the number of nodes in the cluster. It may split the cluster into disconnected components, hence is equivalent to the cluster maintenance mechanism, when an existing node leaves. When a network partition happens, there may be two sets of clusters on each side of the partition. When the partitions join again, no special mechanisms are needed.

V. Experimental Results

We simulated the operations of our clustering scheme on a set of wireless sensor nodes. For the simulation, we generate arbitrary wireless topologies. We randomly place a set of nodes in a 1000 unit $\times$ 1000 unit grid, and vary the connectivity for the topology, by appropriately choosing a uniform transmission radii for all the nodes. No intelligent channel access scheme (e.g. using RTS-CTS messages) has been used for these simulations. Hence, messages collisions happened infrequently leading to packet losses. This, however, does not affect the correctness of the clustering protocol as no hard state is maintained at the nodes. The maintained soft state times out when a sequence of consecutive beacons are lost due to collisions or errors. While channel acquisition based access schemes would improve the simulation results, the nature of the results would be unaffected.

Nodes arbitrarily join and leave the topology. Each node ran an instance of the cluster creation phase of the protocol. In the experiments reported here, we chose P, the average time gap between successive beacons transmitted by a node, to be $10ms$.

Time to Stabilize : A wireless node is said to *stabilize* into a final cluster, during the cluster creation phase, when it undergoes no further changes in its cluster membership during this phase, and stops transmitting cluster creation beacons. In Figure 10, we

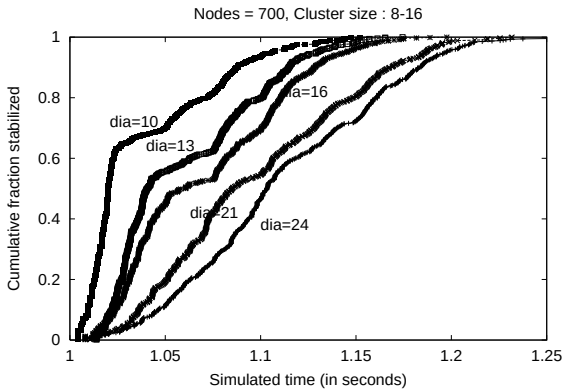

Fig. 10. Cumulative fraction of nodes stabilized into their final clusters during the cluster creation phase with time, in a wireless network for different connectivity (the graph diameter changes) of the topology

Transmit Radius	(Diameter)	Avg. Cluster Dia. for different sizes 10–20	20–40	25–50	40–80
60.0	(24)	4.8	7.8	9.1	11.2
71.0	(21)	4.4	7.6	8.2	10.7
84.9	(16)	4.2	6.8	7.3	10.0
101.1	(13)	3.4	5.1	5.9	7.1

Fig. 11. Cluster diameter for varying connectivity of nodes in the topology and varying cluster sizes

plot results from a set of experiments, on a 700-node topology. The cluster creation phase is initiated by one node at random, at time 1 second into simulated time. The x-axis shows the simulated time elapsed. The y-axis indicates the cumulative fraction of the nodes that have stabilized. As the connectivity of the topology is increased the diameter of the graph decreased, and the cluster creation phase takes lesser time to complete. For a highly connected graph (with diameter 10), the cluster creation phase completes in 150 ms (it is expected to be upper bounded by ($2\times$ *diameter of the graph* $\times$ *avg. inter-beacon period*), which is 200 ms in this case), but for a low connectivity graph (diameter 24) it takes much longer (250 ms). Operations of the cluster maintenance phase involves mostly a single cluster at a time, and hence involve much smaller time scales.

Cluster Diameter : In general, it is desirable to have clusters of low diameter. In Figure 11, we show the average diameter of the different clusters for the 700-node topology. The cluster diameters increase with increasing cluster size and with decreasing connectivity (i.e. increased transmission radius of the individual nodes) of the topology, as would be expected.

We ran experiments on networks with up to 1100 nodes using our simulator. All clusters, as expected, meet the desired cluster size bounds. Only in a few experiments did we see a node having to belong to multiple clusters during the cluster creation phase.

VI. Related Work

Some routing solutions for the Internet have used hierarchies to provide scalability, e.g. OSPF protocol [22] and ATM PNNI [23]. Additionally, in some cases, ATM PNNI [23], [24] splits the network into clusters, called peer-groups, and only summarized information e.g., of cost of traversal, of the peer-groups is exported to the remaining network.

Krishnan et al [13] have explored different graph partitioning schemes for Internet-like graphs. Their target problem is, as a consequence, somewhat different from ours.

In mobile wireless environments, the Zone Routing Protocol (ZRP) [25], has the weak notion of groups, called zones, which are used to limit the propagation of updates. The notion of clustering has also been used previously for hierarchical routing for packet radio networks in [26] and [27]. In [6], [7] clustering algorithms are described for multi-hop mobile radio network, where the clusters are chosen such that the cluster-heads form a dominating set in the underlying graph topology. This makes the number and size of the clusters, largely dependent on the graph topology. They use it primarily for "spatial reuse" of channel spectrum. A similar mechanism is used in Cluster Based Routing Protocol (CBRP) [28] and a more generalized approach is used in [8] for mobile ad-hoc networks. Das et al [9] develop a distributed implementation of approximation algorithms to compute the connected dominating set of a graph [29], to create a routing 'spine' and describe a clustering scheme to create two layered hierarchies. Krishna et al [30] defines a clustering scheme, where each cluster is required to be a clique. These mechanisms are possible by allowing small clusters to exist as may be needed.

VII. Conclusions

In this paper, we demonstrate how certain geometric properties of the wireless networks can be exploited to perform clustering with some desired properties. Generic graph algorithms developed for arbitrary graphs would not exploit the rich geometric information present in specific cases, e.g. the wireless network environment. Even without exact knowledge of node location in different environments, understanding various special properties of the communication graph can lead to better and efficient algorithms.

References

[1] J.M. Kahn, R.H. Katz, and K.S. Pister, "Next century challenges : Mobile networking for Smart Dust," *Mobicom*, Aug. 1999.

[2] L. Kleinrock and K. Faroukh, "Hierarchical routing for large networks," *Computer Networks*, vol. 1, 1997.

[3] Z. Xu, S. Dai, and J.J. Garcia-Luna-Aceves, "Hierarchical routing using link vectors," *IEEE Infocom*, Mar. 1998.

[4] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges : Scalable coordination in sensor networks," *Mobicom*, Aug. 1999.

[5] R. Ramanathan and S. Steenstrup, "Hierarchically organized, multihop mobile wireless networks for quality-of-service support," *Mobile Networks and Applications*, vol. 3, no. 1, June 1998.

[6] M. Gerla and J.T.-C. Tsai, "Multicluster, mobile, multimedia radio network," *ACM-Baltzer Journal of Wireless Networks*, vol. 1, no. 3, 1995.

[7] C.R. Lin and M. Gerla, "Adaptive clustering for mobile, wireless networks," *Journal on Selected Areas of Communication*, vol. 15, no. 7, Sept. 1997.

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

[9] B. Das, R. Sivakumar, and V. Bhargavan, "Routing in ad-hoc networks using a spine," *International Conference on Computers and Communications Networks*, Sept. 1997.

[10] D.B. Johnson and D.A. Maltz, "Dynamic Source Routing in ad hoc wireless networks," *Mobile Computing*, 1996.

[11] C.E. Perkins and E.M. Royer, "Ad-hoc On-demand Distance Vector Routing," *2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999.

[12] V. Park and S. Corson, "Temporally Ordered Routing Algorithm Version 1, functional specification," draft-ietf-manet-tora-spec-03.txt, Internet Draft, IETF, Nov. 2000.

[13] R. Krishnan, R. Ramanathan, and M. Steenstrup, "Optimization algorithms for large self-structuring networks," *IEEE Infocom*, Mar. 1999.

[14] C.V. Ramamoorthy, A. Bhide, and J. Srivastava, "Reliable clustering techniques for large, mobile packet radio networks," *IEEE Infocom*, 1987.

[15] P. Karn, "Maca - a new channel access method for packet radio," *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, 1990.

[16] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "Macaw : A media access protocol for wireless lans," *Sigcomm*, Sept. 1994.

[17] IEEE Computer Society LAN MAN Standards Committee, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-1997," 1997.

[18] C.L. Fullmer and J.J. Garcia-Luna-Aceves, "Solutions to hidden terminal problems in wireless networks," *Sigcomm*, Sept. 1997.

[19] J.J. Garcia-Luna-Aceves and A. Tzamaloukas, "Reversing the collision-avoidance handshake in wireless networks," *Mobicom*, Aug. 1999.

[20] B.N. Clark, C.J. Colbourn, and D.S. Johnson, "Unit disk graphs," *Discrete Mathematics*, 1990.

[21] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, and R.E Stearns, "NC-approximation schemes for NP- and PSAPCE-hard problems for geometric graphs," *Journal of Algorithms*, vol. 26, no. 2, Feb. 1998.

[22] J. Moy, "Open Shortest Path First Version 2," RFC 2178, IETF, July 1997.

[23] ATM Forum Technical Committee, "Private Network-Network Interface, Version 1.0," May 1996.

[24] W.C. Lee, "Topology aggregation for hierarchical networks," *ACM Computer Communications Review*, vol. 25, no. 1, 1995.

[25] Z.J. Haas and M.R. Pearlman, "The Zone Routing Protocol for ad hoc networks," draft-ietf-manet-zrp-03.txt, Internet Draft, IETF, Mar. 2000.

[26] D.J. Baker and A. Ephremedis, "The architectural organization of a mobile radio network via a distributed algorithm," *IEEE Transactions on Communications*, Nov. 1981.

[27] D.J. Baker, J. Wieselthier, and A. Ephremedis, "A distributed algorithm for scheduling the activation links in a self-organizing, mobile, radio network," *IEEE ICC*, 1982.

[28] M. Jiang, J. Li, and Y.C. Tay, "Cluster-Based Routing Protocol (CBRP)," draft-ietf-manet-cbrp-spec-01.txt, Internet Draft, IETF, Aug. 1999.

[29] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets," *Algorithmica*, vol. 20, 1998.

[30] P. Krishna, M. Chatterjee, N.H. Vaidya, and D.K. Pradhan, "A cluster-based approach for routing in ad-hoc networks," *2nd Usenix Symposium*, Apr. 1995.

[31] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," RFC 2608, IETF, June 1999.