Self-organization Protocols for Wireless Sensor Networks

C. Chevallay
NIST
Gaithersburg, MD

R. E. Van Dyck
NIST
Gaithersburg, MD

T. A. Hall
NIST
Gaithersburg, MD

Abstract — **Self-organization is critical for a wireless smart sensor network, due to the large number of nodes, and to the fact that these nodes may be spread over a remote area. Starting with the self-configuration architecture proposed by Subramanian and Katz, we propose a set of distributed algorithms that provides the basis for a complete self-organized sensor network. Specifically, we develop protocols that organize the sensor nodes into clusters and then merge the clusters to form groups. Groups merge to form larger groups, in a hierarchical process that dynamically assigns a unique address to each smart sensor. Additionally, a broadcast tree is constructed in a manner to reduce the maximum number of hops along the tree.**

I. Introduction

Since wireless sensor networks [1] will often have many nodes and will be deployed in remote environments, it is important that they have the ability to self-organize. Early work on wireless self-organization includes the Linked Cluster Algorithm (LCA) of Ephremides *et al.* [2] [3]; LCA uses a fixed TDMA frame structure to form clusters so that all the nodes of the cluster are within one hop of a distinguished node called the cluster head. If the cluster heads of two adjacent clusters are not within transmission range of each other, gateway nodes are designated that connect them. The set of cluster heads and gateways forms the backbone network.

Gerla and Tsai [4] use the linked cluster algorithm to create a multi-hop wireless network suitable for real-time traffic. They use both time division and code division access schemes, and they set up virtual circuits that provide guaranteed bandwidth. Lin and Gerla [5] extend the work in [4] to evaluate the effect of synchronization in the network; their conclusion is that an intermediate degree of synchronization is a useful compromise. Sharony [6] suggests grouping the nodes into both physical and virtual clusters.

Related to clustering is the problem of finding a minimum connected dominating set (MCDS) of the nodes. An MCDS satisfies two properties: (1) each node is either a backbone node or is connected (one hop) to a backbone node, and (2) the backbone nodes are connected. Kozat *et al.* [7] propose a virtual dynamic backbone protocol that constructs a backbone satisfying the MCDS properties. The backbone can then be used for multicasting, broadcasting and QoS routing. Amis *et al.* [8] cluster a network by finding a d-hop dominating set. In this case, the nodes in a cluster may be up to d hops from the cluster head. Since their general problem is NP-complete, they develop a heuristic to organize the network.

Mirkovic *et al.* [9] specifically look at a large scale sensor network with no mobility. They assume that there are too many sensors to allow unique global IDs, and so they develop a multicast tree protocol that allows multiple sinks to obtain data from a (sensor) source. This approach differs from those above in that it does not cluster the nodes. Wieselthier *et al.* [10] also develop protocols for the construction of broadcast and multicast trees in a wireless network with no mobility. To extend the nodes' lives (and hence the network's life), they form a minimum-energy tree from a source to the sinks.

In most of the work mentioned above, the nodes are grouped into clusters that can be used for: (1) backbone formation, (2) transmission management, and (3) routing efficiency. For a wireless sensor network, especially one that is self-organized, it is useful to take this clustering concept even further to form a hierarchical network. Specifically, clusters are aggregated to form groups, and groups are aggregated to form larger groups. Moreover, it is desirable to build address auto-configuration into the procedure. The resulting addressing scheme is typically not IP protocol-based.

The self-configuration architecture proposed by Subramanian and Katz [11] leads to a hierarchical network with address auto-configuration and a number of other useful properties. In this paper, we complete their architecture by proposing a set of distributed algorithms and message formats that allow an actual implementation. We identify a number of important parameters, and we study their effects on the overall system performance. In particular, protocols are developed that organize the nodes into clusters, clusters into groups, and groups into larger groups. An algorithm is given that builds a broadcast tree.

Another important issue is the coordination of the transmissions of the various nodes so that the number of packet collisions is kept at an acceptably low level. In general, this is a fairly difficult problem, heavily dependent on the structure of the network, the amount and types of data and control traffic, and the medium access control (MAC) protocol. In a smart sensor network, collaborative signal processing algorithms may be run in a cluster or a group [12], thereby generating local traffic. The MAC protocol used during this phase of network operation need not be the same as during the self-organization phase. An advantage of our proposed system is that during the organization phase, the MAC layer is a simple Aloha scheme; this is due to the hierarchical structure and the relatively small number of messages that are transmitted during the self-organization process.

II. Self-Organization Algorithm

The core concept of this work is inspired by Subramanian and Katz [11], who proposed a self-configurable system and then presented the main lines of an algorithm by which the sensor nodes organized themselves to form a network. Their algorithm provides a unique address for each node, allows packets to be routed between any pair of nodes, builds both network-wide and local broadcast infrastructures, and reorganizes the network in case of link or node failures. Moreover,

[0]This work was supported by the Advanced Research and Development Activity (ARDA) under contract number 706400.

the algorithm is scalable. They identify four phases in the network:

1 Discovery phase: Each node learns about its neighbors, and it sets its transmission radius.

2 Organizational phase: The nodes start to aggregate together into clusters; clusters aggregate themselves into larger groups, and these aggregate to form even larger groups, *etc.* To some extent, the groups may merge with other groups that are not exactly the same size, but the merging is designed so that the two groups are roughly the same size. This is done so that the routing table maintained at each node is $O(\log N)$, where N is the number of nodes in the network. The merging process is discussed in more detail below.

3 Maintenance phase: Every node periodically emits an "I am alive" message, and it informs its neighbors about it routing table, its own neighbors, and its remaining level of energy. This process is called active monitoring; in passive monitoring, a node has to query another node to know if it is still alive.

4 Self-reorganization phase: A node detects neighbor failures or group partitions, and it accordingly updates its tables. If a group partition occurs, the group will try to join other existing groups, while keeping the hierarchy height balanced.

Random, Ephemeral Transaction Identifiers

During group formation, when smaller groups merge with each other to form larger groups, the nodes from different groups have to exchange messages. Therefore, it is not sufficient to identify a node only with its address, since two nodes in two different groups may have the same one. To remedy this problem each group needs to be assigned a group identifier. Also, when a cluster head has advertised its presence, a node which has not found a cluster, and therefore has no address, may want to join.

In both the above cases, we use Random, Ephemeral TRansaction Identifiers (RETRI) [13]. RETRI are randomly drawn numbers of a given limited size. They can be used when one needs an identifier that should be unique, but only in a certain place or over a limited period of time. Since a RETRI is *probabilistically unique*, there is no absolute guarantee about the uniqueness of such an identifier; however, if one draws these identifiers out of a large enough pool, they have a high probability of being temporally and spatially unique. Determining the proper size of these identifiers is one of the goals of this work.

III. Cluster Formation

To simplify the protocol design and reduce the number of messages that must be exchanged among the nodes, we decided that the members of the cluster would be one hop away from the cluster head. This is consistent with the approach taken by Ephremides *et al.* [2] [3]. Figure 1 shows the complete message sequence between a cluster head and a node interested in joining the cluster.

Cluster Advertising

When any smart sensor comes to life, it first listens for some random time to determine if it can hear a cluster head. If its timer elapses before it hears anything, the node will create its own cluster and become a cluster head itself (each cluster head gets the local address 000). The duration of this time may be dependent on the node's remaining level of energy and its processing capacity, so that a more "powerful" node is more likely to become a cluster head. A new cluster head will locally broadcast an invitation (`join` message in Figure 1) such that all nodes within a radius r will hear it [1]. Initially, $r = r_0$ (see below for a discussion on r_0). The `join` message contains a RETRI cluster ID, and a flag indicating which power class is used to send the message. This flag is used by nodes who may want to join the cluster, so that they know what emission power they should use in order to be heard by the cluster head (roughly assuming channel symmetry). The concept of a power class is simply a way of discretizing the emission power, and each power-class may be able to be mapped to an approximate transmission radius.

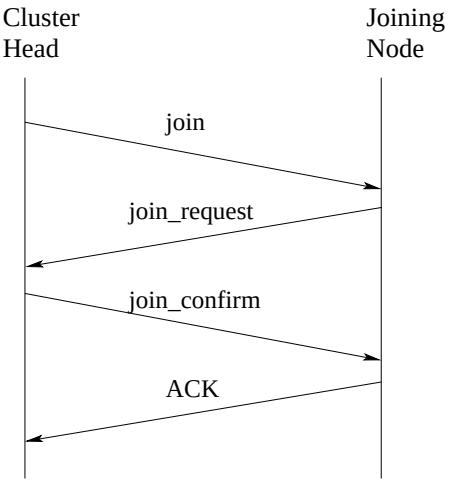

Fig. 1: Message exchange when a node joins an existing cluster.

Answering

If a `join` message is heard during the initial listening time, the node will reply to the cluster head with a `join_request` message, indicating the cluster ID that was heard (this uniquely identifies the cluster head) and its own RETRI temporary node identifier. Since it is quite likely that a collision will occur if all the nodes that heard the cluster head and are willing to join the cluster reply at the same time (this is a *reply storm*), each node introduces a random delay before replying. If a node hears multiple cluster head advertisements, it should join the nearest cluster, *i.e.*, the one whose `join` message was sent with the lowest power class. This action is aimed at preserving energy.

Managing Cluster Size

The goal is to have neither too many nor too few nodes in a cluster, in order to provide good connectivity and redundancy while limiting collisions. Let $n(A)$ be the minimum number of nodes in a cluster whose cluster head is A, and let $N(A)$ be the maximum number (including A in both cases). In [11], it was suggested that one use $N(x) = 8$, so that 3-bit addresses can be locally assigned within a cluster.

[1] Depending on the terrain and local jamming, the actual region that is capable of receiving the message may be very different in shape from a circle. Yet without loss of generality, we can still refer to this region as having a radius, r.

If there are not enough nodes in the cluster, the cluster head keeps broadcasting `join` messages, increasing the transmission radius according to $r = k \cdot r$, $k > 1$. k should be chosen to limit the number of consecutive retransmissions to gain new members. Ideally, only one retransmission should be necessary (assuming no collisions among the joining nodes); if after the first message, only half the minimum required number of nodes join the cluster, then the broadcast area should be roughly doubled. (This scheme assumes a uniform distribution of nodes over the area, which may be a reasonable approximation on a local scale.) k is linked to the defined power classes. Note that one can not increase the transmitter power indefinitely (*e.g.* the cluster head is in a remote area with few surrounding nodes). Therefore, there is a maximum radius R_{max}; if $r > R_{max}$, the cluster formation algorithm is terminated, and each node tries to join previously existing clusters. For this purpose, cluster heads should continue to advertise once their cluster is formed.

Upon receiving a `join_request` message, the cluster head will check if it still has some room in its cluster to accommodate the new node. If the answer is yes, the cluster head replies with a `join_confirm` message, using the temporary address as the destination; the cluster head also assigns the new cluster member one of the free local addresses. The node then sends an acknowledge (`ACK`) message. If the cluster head receives a `join_request` but the cluster is full, it simply ignores this message. The node that wants to join will reset after some time, and start listening again to cluster head advertisements.

To illustrate the cluster formation procedure, consider the simple eleven node wireless network shown in Figure 2(a). Here, the thin lines indicate the radio links between nodes. Figure 2(b) shows the resulting clusters, indicated by the dashed lines. The three shaded nodes declared themselves cluster heads, and the other nodes each joined a cluster. Each cluster has its own local broadcast tree, as indicated by the heavy lines. The merging of these clusters is discussed in Section IV.

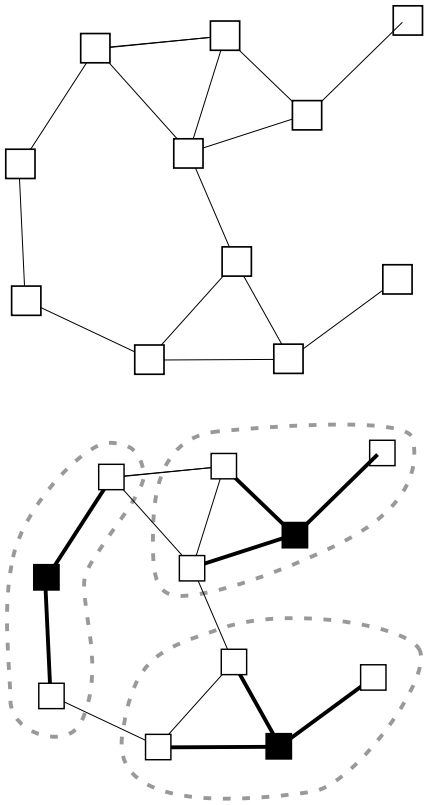

Fig. 2: $\frac{(a)}{(b)}$ (a) The initial network. (b) Three clusters formed.

IV. Cluster Merging

Overview

The network self-organization algorithm is based on the fact that the groups will regularly try to merge with each other (if possible), until the entire network becomes one single group. At this time, each node has a unique address. When two groups merge, they are the two halves of a new, larger group. Therefore, the nodes in each group will increase their address sizes, as well as modify their addresses (by adding one or more new bits to the front of it, to insure address uniqueness).

Since the addresses are dynamically assigned, a sink node will have to broadcast its address once it receives one. Thus, smart sensor nodes can not be programmed with an instruction such as "once you have made a decision, send it to the sink node whose address is xxx". Instead, the command would be "once you have made a decision, send it to the sink node; it will have advertised itself as such when it joined your local network."

To describe the merge operation, it is useful to define the difference of address size between the two merging groups' nodes. Let m be the address size in the larger group, and let n be the address size in the smaller one. Define $\Delta := m - n$. Figure 3 shows the flow of messages exchanged during the merge operation, while Figure 4 shows how two clusters merge to form a larger group. In this case, node C initiates the merge with node B's cluster. After the merge process, the two clusters have combined their broadcast trees, as shown by the heavy line between nodes B and C. A discussion of the merging process continues below.

Finding a Suitable Group

The first step in the merge process is to choose a neighboring group to merge with. All nodes have to advertise the size of their addresses as well as their group ID in the "I am alive" beacon messages that they occasionally transmit. The reason for transmitting the address size is so that a group will try to merge with another group that is more or less the same size. The beacon is very useful during network self-organization, since it triggers merge operations. However as the size of each connected component of the network grows, the frequency of the advertisements may be reduced for non-border nodes (these are the nodes whose neighbors all belong to the node's group), providing energy saving and a reduced packet collision rate.

When a group detects more than one neighbor that could be a suitable candidate for merging, it needs some criteria to rank the choices:

- For addressing efficiency, a group should try to merge with another group whose address size is as big as its own (this is to maximize address space utilization, and to not leave "holes" in the address space). So, a group should first try to join a group of the same size ($\Delta = 0$), then if this is not possible, one where $\Delta = 1$, *etc.*

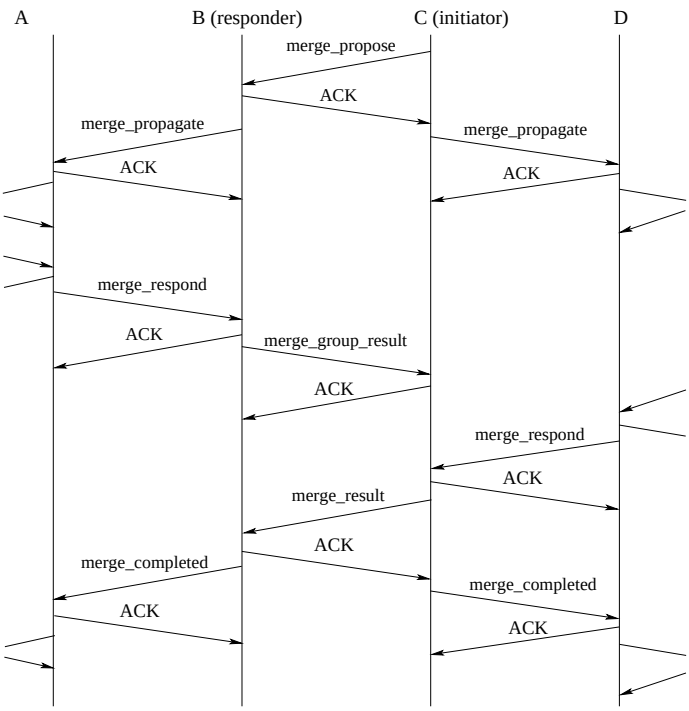

Fig. 3: Message flow during merging operation.

- If two (or more) neighboring groups who are candidates for the merge have the same address size, other criteria must be used. Possible choices include the number of links between each neighbor and the initiator's group, or the expected traffic between the two groups. Alternatively, we propose a metric called *attractiveness*.

Attractiveness

The broadcast tree is built as groups merge with each other. At each step, a link is created between the node that initiated the merge process and the one that was invited.

When the node pair is randomly chosen, one does not have any control on the resulting tree structure. For efficiency reasons, the tree structure should be as close as possible to a binary tree, and a configuration that should be avoided occurs when all clusters are linked along a single backbone chain. This is clearly not optimal for broadcasting information through the entire network, since the information can not be sent in parallel. Unfortunately, if the merges occur randomly, the tree very often takes on this chain-like appearance.

So to build the tree, one would like that a new link is formed between two groups at nodes that are close to the *centers* of their respective groups. The center of a group is where the two subgroups were linked when the group was built, and it is supposed to be relatively close to the group's topological center. To achieve this objective, we introduce the concept of *attractiveness*. When two groups merge, the link between them (to merge their broadcast trees) should be created between nodes that have a high attractiveness, when compared to their neighboring nodes. The closer a node is to its group's center, the higher the attractiveness. Attractiveness is denoted by A, and it is regularly advertised in the beacon message.

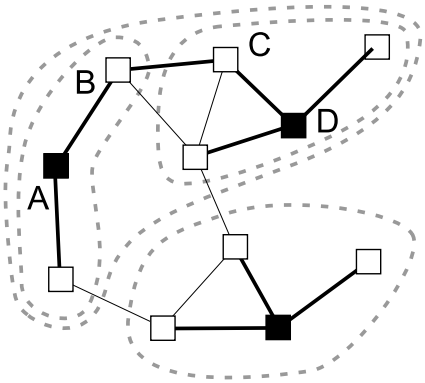

Fig. 4: Two clusters merging into a larger group.

Attractiveness is computed as follows. Initially, when a node joins a cluster, its attractiveness is set to 0 (attractiveness is not defined for nodes who have not yet joined a cluster). Then, as groups merge with each other, the broadcast tree expands with new links being created. All members of the cluster at each end of the new link increase their attractiveness an amount corresponding to the new address size. For example, if the new address size is two bits larger then the basic one (*i.e.*, at cluster formation), then the new attractiveness is two. Within a cluster, some nodes should have a higher attractiveness than others (since they are closer to the group's center). Therefore, 0.25 is added to the cluster head, and 0.5 to the node on the end of the new link.

As a result, a node with a relatively high attractiveness should be more likely to initiate a merge process, and it should very likely choose, among its neighbors, a node with a high attractiveness as the other end of the link. The time between tries to merge with one of its neighbors is found by taking the basic time and dividing by A^2, thereby allowing more attractive nodes to check more frequently. The probability to choose a particular neighbor i is computed by

$$p_i = \frac{A_i^\alpha}{\sum_{neighbors} A^\alpha}, \tag{1}$$

where α is a scaling factor to be chosen later. Thus, two new fields are added to the `merge_completed` packet. The first one is a single bit indicating whether or not there is a new value for attractiveness, and if this is the case, the second field gives the new value. The attractiveness must be updated only within the cluster to which the link extremity belongs.

Merging Procedure

The merging procedure assumes two groups, whose address sizes are m and n bits, respectively; without loss of generality, assume $m \geq n$. Also, remember that $\Delta = m - n$. Two cases need to be considered.

- Same-level fusion: $m = n$.

In this case, the two groups to be merged have the same address size. This is the best situation, since it will maximize utilization of the address space. If the merge actually takes, place, the nodes in one of the groups will add the bit 0 in front of their existing addresses, while the nodes in the other group will add the bit 1.

- $m > n$

Check Δ, which decides between a fusion (although unbalanced), and an "adoption."

- Small Δ: unbalanced fusion.

  The two groups will merge as if they had the same address size. The smallest group will have to add more bits in front of its address than the usual 0 or 1 bit, and there will be a "hole" in the resulting address space. Subramanian and Katz [11] propose $\Delta < 3$, which we use for the present.

- Large Δ: adoption.

  Having an unbalanced merged here would create a very big hole in the address space, which is not desirable. Therefore, the smallest group should be adopted by the larger one; that is, it should try to fit into one of the holes in the larger group's address space. This illustrates that it is good to have an address tree structure that has some small holes for flexibility.

Inviting the other Node

When the merge initiator, node C in Figures 3 and 4, has finally decided on node B (the responder), it will notify it with a `merge_propose` message. Then, each of these two nodes will have to ask the other nodes in its own group whether or not they approve the merge. For scalability reasons, it is not a good idea to query every single node in the current network; instead, the group head has the authority to make the decision. Hence, nodes B and C each need to propagate the query to their respective group heads.

Packets may be routed directly to the group head. Here, `merge_propagate` packets are instead routed along the broadcast tree. Each node knows the direction of the group head. When the group head gets the message, and no such process is currently under way, it will approve the merge. Otherwise, it disapproves. The answer `merge_respond` is then sent back to the originating node using the reverse path along the broadcast tree. In the current implementation, if an intermediate node, which has already propagated a query towards its group head, receives another query, it does not propagate the second one. Instead, it sends a negative `merge_respond` to the second querying node, thus saving messages to/from the group head.

Merge Conclusion

When both the initiator and the responder nodes each have an answer to their queries, it is time to decide about the merge. The responder communicates its group's answer to the initiator through a `merge_group_result` message. Only if both broadcast trees have returned a positive result will the merge take place. The initiator will then tell the responder of the final decision, using a `merge_result` message. Finally, each of these two nodes will broadcast the same result down its broadcast tree using a `merge_completed` message. If the result is positive, it is sent to all nodes in both groups. If negative, it is only sent to the nodes that were aware of the merge operation, so they can properly set their internal state.

Upon reception of a positive `merge_completed` message, a node has to update its neighbor and routing tables. The number of new bits in the address, the address, and the group identifier of the new group are all changed. This way, each node has a group identifier for each address size; the first one is the cluster identifier. The attractiveness is updated within the respective cluster of the initiator and the responder.

After a successful merge, the two broadcast trees have merged into one. The link occurs between the initiator node and the responder, and so the broadcast tree spans the two groups. The initiator node becomes the group head for the new group. As two groups of any size merge, the procedure is the same: the two previous broadcast trees are merged. This ensures that the final broadcast tree reaches every node, and that it does not contain any loops.

V. Simulation Results

The self-organization protocols have been simulated using the implementation of the Scalable Simulation Framework (SSF) created by Dartmouth College [14]. A key assumption in the present version of our simulation is that there are no packet collisions. While one can conceive a MAC layer that uses frequency hopping and/or multi-user detection to minimize the number of collisions, we acknowledge that this assumption is not intended to be completely realistic. Instead, it allows us to determine the initial scalability of the system. Still, the messages are on the order of 100 bits/packet, so even at the relatively low bit rate of 10,000 bits/sec, a packet takes about 10 msec for transmission. Consequently, there will not be all that many collisions, assuming the transmission range is kept reasonably small.

Figure 5(a) shows the average number of packets required for the nodes to form clusters. Each data point is an average of 100 trials, where a new network is randomly generated for each trial. One can clearly see that the number of packets grows linearly with the number of nodes. This is primarily due to the parallelism in the clustering process. Moreover, the average number of neighbors has very little effect on the results. Networks where each node only has three neighbors are often disconnected, while those with seven or more have a high probability of being connected. Figure 5(b) gives the average number of clusters as a function of the network size. Here, the average number of neighbors does change the results. A network where each node averages three neighbors has almost twice the number of clusters as one where each node has 15 neighbors. Note that presently each cluster head is limited to having seven ordinary nodes in the cluster, which increases the number of clusters.

To get a sense for the locations of the cluster heads and the types of clusters, it is useful to consider the network shown in Figure 6. Thirty-six nodes, depicted by the small circles, are placed along the sides of a road that forks. A typical sensing application would like to detect, classify, and track vehicles driving down the road. The self-organization algorithm creates nine clusters, ranging in size from a single cluster head up to the maximum of eight nodes. The cluster heads are designated by boxes around the nodes, and the transmission radii are shown by the larger circles. One can compare these results to those obtained using the linked cluster algorithm, *cf.* Figure 4 in [12].

VI. Discussion and Conclusions

This paper develops and defines a distributed algorithm for self-organization of wireless sensor networks. The simulation results show that the parallel nature of the cluster formation phase allows it to scale to large networks. Further work remains in investigating the impact of a number of parameters, including R_{max} and k, which relate to the transmission radius, and therefore transmission power, of the individual nodes. Increasing the maximum radius of the cluster implies higher

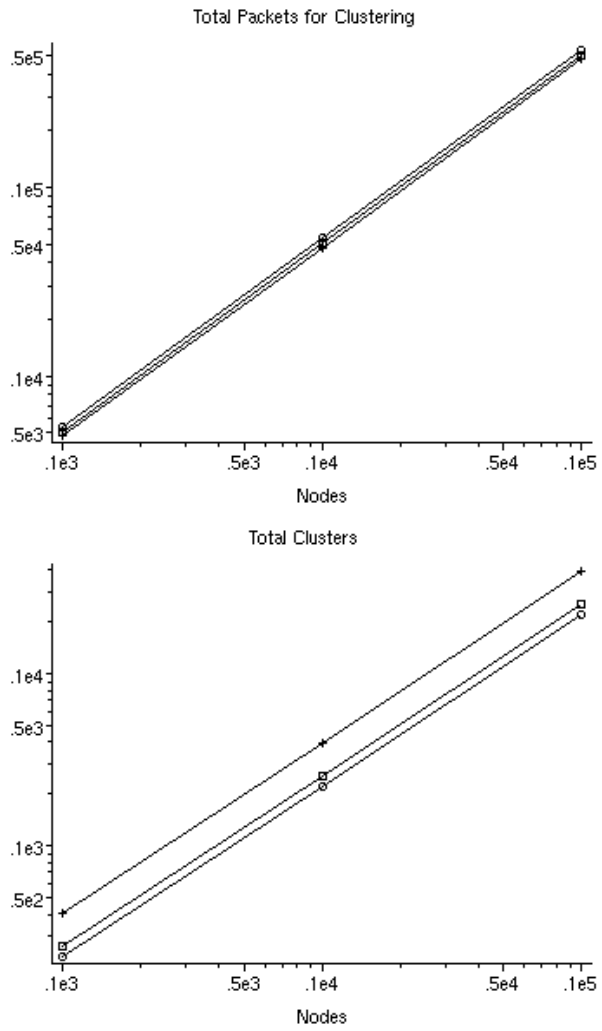

Fig. 5: $\boxed{\frac{(a)}{(b)}}$ (a) Average number of packets, and (b) average number of clusters, both as a function of the number of nodes. Crosses represent networks where each node averages 3 neighbors, boxes represent 7 neighbors, and circles 15 neighbors.

energy expenditure and increased interference. The impact of k is also important. If k is too small, then the number of retransmissions to gain new members is high, resulting in a longer time needed to self-organize the network. It may also adversely affect the energy required. If k is too large, then the transmission power may be higher than needed. One would also like to constrain the maximum energy required by a single node. Investigating these issues will require a more detailed look at the MAC layer design. The optimal choice of time-outs is also of interest, since it determines the time required to self-organize and the resulting structure. Ultimately, we wish to measure the performance of the algorithm by the ability of the network to perform its sensing task.

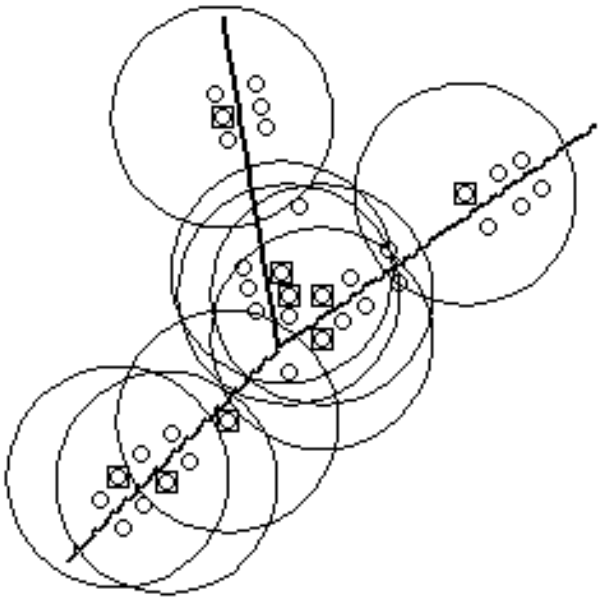

Fig. 6: Example of clustering a 36 node network. The nine cluster heads are depicted by boxes, and the ordinary nodes are depicted by circles.

References

[1] G. J. Pottie, "Wireless sensor networks," *IEEE Information Theory Workshop*, pp. 139-140, June. 1998.

[2] D. J. Baker and A. Ephremides, "The architectural organization of a mobile radio network via a distributed algorithm," *IEEE Trans. on Comm.*, vol. COM-29, pp. 1694-1701, Nov. 1981.

[3] A. Ephremides, J. E. Wieselthier, and D. J. Baker, "A design concept for reliable mobile radio networks with frequency hopping signaling," *Proc. IEEE*. Vol. 75, pp. 56-73, Jan. 1987.

[4] M. Gerla and J. T.-C. Tsai, "Multicluster, mobile, multimedia radio network," *Wireless Networks*, pp. 255-265, 1995.

[5] C. R. Lin and M. Gerla, "Adaptive clustering for mobile wireless networks," *IEEE J. Sel. Areas in Comm.*, Vol. 15, No. 7, pp. 1265-1275, Sept. 1997.

[6] J. Sharony, "An architecture for mobile radio networks with dynamically changing topology using virtual subnets," *Mobile Networks and Applications*, pp. 75-86, 1996.

[7] U. C. Kozat, G. Kondylis, B. Ryu, and M. K. Marina, "Virtual dynamic backbone for mobile ad hoc networks," *Proc. IEEE Int. Conf. on Comm.*, Helsinki, Finland, 2001.

[8] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, "Max-min D-cluster formation in wireless ad hoc networks," *Proc. IEEE Infocom*, pp. 32-41, 2000.

[9] J. Mirkovic, G. P. Venkataramani, S. Lu, and L. Zhang, "A self-organizing approach to data forwarding in large-scale sensor networks," *IEEE Int. Conf. on Comm.*, Helsinki, Finland, 2001.

[10] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "On the construction of energy-efficient broadcast and multicast tree in wireless networks," *Proc. IEEE Infocom*, pp. 585-594, 2000.

[11] L. Subramanian and R. H. Katz, "An architecture for building self-configurable systems," *Proc. IEEE MobiHoc*, 2000.

[12] R. E. Van Dyck and L. E. Miller, "Distributed sensor processing over an ad-hoc wireless network: simulation framework and performance criteria," *IEEE Milcom*, Mclean, VA, Oct. 2001.

[13] J. Elson, and D. Estrin, "Random, ephemeral transaction identifiers in dynamic sensor networks," *Proc. 21st Int. Conf. on Distributed Computing Systems*, pp. 459-468, 2001.

[14] J. Liu and D. M. Nicol, *Dartmouth Scalable Simulation Framework, Version 3.1 User's Manual*, http://www.cs.dartmouth.edu/research/DaSSF/, Aug. 2001.