FAIL-SAFE HIERARCHICAL ORGANIZATION FOR WIRELESS SENSOR NETWORKS

Stefano Basagni
ECE Department
Northeastern University
Boston, MA, U.S.A.
E-mail: basagni@ece.neu.edu

Chiara Petrioli and Roberto Petroccia
Dipartimento di Informatica
Università di Roma "la Sapienza"
Roma, Italy
E-mail: {petrioli,petroccia}@di.uniroma1.it

ABSTRACT

This paper presents the definition and evaluation of a new protocol for providing a wireless sensor network (WSN) with a hierarchical organization. Differently from previously proposed solutions, our protocol, termed CC ("double c," for clique clustering*), includes in its operation a fail-safe mechanism for dealing with node failure or removal, which are typical of WSNs. More specifically, the network is partitioned into clusters that are* cliques*, i.e., nodes in each clusters are directly connected to each others. An efficient mechanism for building a connected backbone among the clique clusters is provided. Clustering, backbone formation and backbone maintenance are completely localized, in the precise sense that only nodes physically close to a failing node are involved in the reconfiguration process. We compare the performance of CC with that of DMAC, a protocol that has been previously proposed for building and maintaining clusters and backbones in presence of node removal. Our comparison concerns metrics that are central to WSN research, such as time for clustering and backbone reorganization, corresponding overhead (in bytes and transmission energy), backbone size, extent of the reorganization (i.e., the number of nodes involved in it), and backbone route length. Our ns2-based simulation results show that the design criteria chosen for CC are effective in producing backbones that can be reconfigured quickly (*63% *faster than DMAC's) and with remarkably lower overhead.*

I. INTRODUCTION

The extensive research and experimentation performed in the past couple of decades witness the remarkable interest of the academic and industrial community in wireless ad hoc and sensor networks (WSNs), and in protocols that make possible to deploy these networks effectively and at reasonable cost. Among the topics that have received particular attention, *clustering and backbone formation* have triggered quite a community-wide discussion. The reasons are to be found in the traditional use of a superimposed hierarchical structure over the "flat" network topology for favoring routing scalability in terms of routing table size, reduced routing overhead, etc. There are also reasons that pertain specifically to wireless multi-hop networks such as ad hoc and WSNs. In the latter case, especially, clustering provides a natural choice for selecting aggregation points where to merge redundant sensed data. It has been demonstrated that the beneficial effects of data fusion largely counteracts the sub-optimality of routes over the backbone [1]. Backbone formation also provides a straightforward way for topology control. Once clusters and a backbone have been formed, one node per each cluster (the clusterhead) remains awake to perform the network operations, while the radio interface of all the other nodes is turned off (sleep mode) for energy saving and prolonged network lifetime [2].

The majority of clustering and backbone formation protocols proposed for WSNs have been designed for static, or quasi static networks. This implies that when nodes move away, or fail, the clustering process must be re-executed. Recently some protocols have been proposed for ad hoc networks [3] and for WSNs [2] that explicitly cope with these different types of network dynamics without repeating the whole clustering protocol. Clustering maintenance, as well as backbone maintenance, however impose non-negligible overhead, and decrease the overall network performance. (For quite an extensive list of works on clustering, see [4]).

In this paper we introduce and evaluate a new protocol where coping with nodal failures, and especially with the removal of a node from a WSN, is embedded in the design of the clustering and backbone formation protocols. The idea is that of building, and connecting, clusters which are resilient to the failures of one or more nodes. For this purpose, we introduce a new protocol where the clusters are *cliques*, i.e., subsets of the network nodes that are all connected to each other. The protocol, termed *clique covering*, or CC (*double c*) for short, is executed at each node (i.e., it is distributed) based on simple assumptions and information about the node's immediate neighbors (i.e., it is localized). Once the CC clusterheads have been selected a backbone is efficiently formed that is connected. We demonstrate CC through simulations. In particular, we show its effectiveness in quickly and efficiently responding to nodal failure/removal by comparing it with DMAC, a clustering protocol for which backbone construction and reorganization have been defined and tested. The Distributed Mobility-Adaptive Clustering (DMAC) was originally proposed to extend clustering algorithms for static multi-hop scenarios with mechanisms for coping with the mobility of nodes. (For details the reader is referred to [3].) DMAC gives an example of how the removal or failure of a node can trigger a chain reaction involving nodes in the network that can be quite far from the one removed/failed. CC and DMAC are compared with respect to metrics that show how building clusters that are cliques is effective in reducing time, overhead and the number of nodes involved in a reorganization while producing reasonably small backbones. Metrics of interest to our comparison include the following (all taken after nodal removal or failure): Time it takes to re-build a connected backbone, overhead for cluster construction and maintenance, number of nodes involved in the backbone reorganization, backbone size and route length. We observed that while producing backbones bigger than DMAC (up to 39%), CC reacts to network changes efficiently and quickly.

1-4244-1513-06/07/$25.00 (c)2007 IEEE

More precisely, DMAC incurs almost seven times higher overhead than CC, and needs up to 63% longer time to reconfigure after nodal failure. Therefore, CC provides a suitable solution for organizing dynamic WSNs hierarchically in realistic scenarios, where overhead-imposed energy consumption and reorganization time must be minimized.

The paper is organized as follows. Previous works on partitioning a multi-hop network into cliques are reviewed in Section II. Section III describes our CC protocol as well as backbone construction and reorganization after nodal failure/removal. The performance comparison between CC and DMAC is investigated in Section IV. Finally, Section V concludes the paper.

II. RELATED WORK

The problem of grouping the nodes of a multi-hop network in clusters, interconnected via gateways into connected backbones for increased scalability and overall protocol efficiency, has been widely investigated both in the context of ad hoc networking research, and, more recently, in the context of WSNs. A quite exhaustive survey on these topics can be found in [4], where a performance-based comparison of leading solutions for WSN clustering and backbone formation is also presented. The majority of the clustering schemes proposed for WSNs (including the ones evaluated in [4]) work for static networks. In other words, they cannot deal with network dynamics such as nodes mobility, add or removal of nodes, failures and "death" of nodes due to energy depletion. Despite this is a reasonable initial approach for networks such as WSNs, which are usually static, the growing attention for mobile sensor networks [5] and the demand for fail-safe solutions requires to design protocols that can effectively operate in presence of network dynamics. The problem of clustering set up and maintenance in the face of varying network dynamics has received little attention so far. Works in this area have been focusing mostly on how to maintain a cluster organization after an initial clustering has been set up [6].

The topic we deal with in this paper concerns showing that a clique-based clustering can provide an effective solution for dealing with nodal failures. Furthermore, we demonstrate that this approach outperforms other well-known and established connected dominating sets-based approaches to clustering, which also deals with clustering and backbone reorganization after nodal removal/failure, such as DMAC [3]. Here we review those solutions that have been presented so far for clique clustering. One of the first works on the subject is [7], where clustering is used for making routing in dynamic (ad hoc) networks more scalable. More precisely, the authors investigate the partitioning of the network into clusters of diameter k, with the particular case of a cluster being a clique when $k = 1$. The focus of this paper is demonstrating the effectiveness of the presented ad hoc routing protocols in delivering packets among pairs of selected nodes, and the clustering is seen as a mean of favoring scalability. In other words, clustering is here only functional to routing and the description of the clustering organization, the analysis of its properties and its demonstration is not presented. Furthermore, in dense scenarios, this solutions is not practical, since it calculates all cliques a node can belong to and this computation generates quite a remarkable amount of overhead, with a detrimental effect on network performance.

An approach similar to that of [7] is explored in [8], where starting from the construction of a tree spanning the network nodes, clusters are built in a distributed way where two nodes in a cluster are at most k hops apart. When $k = 1$ this protocol clearly generates cliques. The motivations of this work concern enabling scalable routing. Differently from [7], clustering properties are investigated. It is shown that the proposed protocol approximates the lowest number k-clustering (i.e., a clustering with the lowest number of clusters of diameter at most k) with a competitive ratio of $O(k)$. The focus of this paper is mostly theoretical, and the proposed algorithm concerns setting up clusters on a static network with no nodal failures.

A distributed algorithm for partitioning a *multi agent system* (MAS) where agents and their interconnections are represented by a graph is described in [9] (ad hoc networks, and hence WSNs, are particular MASes). Every node v computes all the cliques it belongs to. Then v selects the biggest clique and checks whether all the nodes in that clique agree to affiliate to it. If this is the case, a cluster is formed, and its nodes communicate this information to their neighbors. Upon receiving this information, every other node recomputes its cliques based on the updated information. If the biggest clique of a node cannot be formed, the node updates the information it has and selects the next biggest clique, repeating the process until it finds the clique it can participate in. The efficiency of this protocol depends on how large a clique can be. For this reasons, the authors limit the size of the cliques to contain the corresponding protocol duration and overhead. Computing all cliques (up to a given size) and repeatedly advertising the current biggest clique generate non-negligible overhead. Furthermore, the presented protocol is not suitable for dynamic networks. Node coalitions (i.e., cliques) in MASes are simply disrupted when a node moves away or fails. A clustering method similar to the one presented in [9] is also used in [10] and [11]. Determining the cliques a node belongs to is aided by the knowledge of geographic information (each node knows its coordinates) in [10]. In this case each node finds its cliques by computing the cliques around every edge, and then it decides to participate to the biggest one, if possible. Although distributed, and hence suitable for WSNs, this algorithm makes some strong assumptions. For instance, it is assumed that all the nodes around a link are easily found. The protocol has no provisions for coping with any kind of network dynamics. In [11] an extra step is added to the basic steps of the algorithm of [9] for securing the clustering construction. Upon determining its membership to a clique each node performs a conformity check for identifying potential malicious nodes. If no such node is found, nodes enforce a clique agreement and exit the protocol. Otherwise, the malicious node is removed from the network and the nodes start the protocol from scratch. As for similar solutions, this protocol has no methods for dealing with mobility or node failure.

The problem of efficiently computing all maximal cliques in a unit disk graph (UDG, a common model for static wireless ad hoc and sensor networks) is investigated in [12]. In this work the focus is that of finding (a centralized) polynomial time approximation algorithm for producing all maximal cliques by

using key geometric structures of UDGs, which in turn requires each node to be aware of their location. Specifically, for each edge in the graph, the set of nodes that are included in the area delimited by some given shapes based on that edge are considered. It is then shown that all possible cliques having this as the longest edge are contained in that area. An algorithm is presented to select some of these cliques as the clusters, and the authors also indicate that the proposed algorithm can be distributed. No indication on how the algorithm reacts to network dynamics is provided.

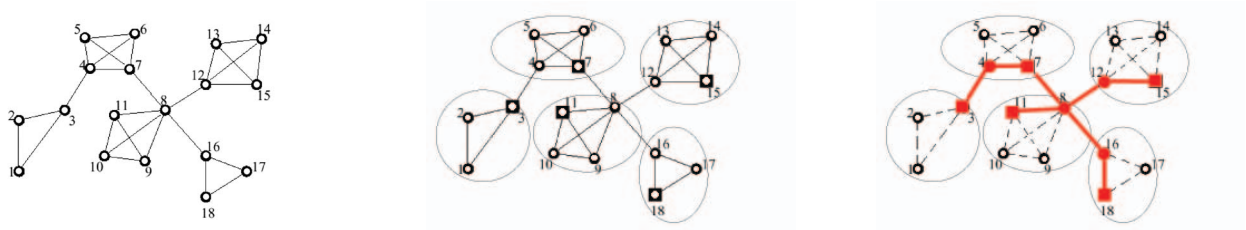
All the above algorithms about clustering by cliques are pretty much concerned with building the clusters per se, or with using the resulting network partitioning to favor routing, security, etc. None of these solutions has been extended to include explicit backbone formation, or to incorporate features for automatically counteracting the effect of a possible failures (mobility, battery depletion, etc.) This prompted us to come up with a protocol which builds clique-clusters and connect them into a backbone which is fail-safe to network dynamics such as node removal/failure while incurring reasonable overhead.

III. CLIQUE COVERING(CC)

The CC protocol produces a clustering that satisfies the following properties: 1) Every non-clusterhead node has at least a clusterhead as neighbor (i.e., the set of clusterheads is a dominating set); 2) Every node in a cluster can communicate directly with every other node in the cluster (clique property), and 3) every non-clusterhead node affiliates to the cluster of the first clusterhead inviting it.

In describing CC we assume that every node knows its own unique identifier (ID), its own *weight* and the ID and weight of each of its neighbors. The weight of a node is a real number ≥ 0 which depends on the node current status and application requirements, and that indicates the suitability of the node for being selected as a clusterhead. The higher the weight the better is a node for assuming that role. The protocol is executed at each node v in such a way that v decides its role (clusterhead or ordinary node) as soon as its "heavier" neighbors (neighbors with bigger weight) have decided their own role. The protocol is started by the *init nodes*, i.e., by those nodes that have the bigger weight among all their neighbors (ties are broken via the node unique ID). An init node sends a (broadcast) message telling it will be a clusterhead. Upon receiving this message, a node exchanges with the sender information about its own neighbors. Based on the received information, a clusterhead selects all those neighbors that can be associated to its own cluster while maintaining the clique property, and invites them to join it. A node whose heavier neighbors have joined other clusters or have finished inviting nodes, and that has not been invited to be part of any cluster decides to be a clusterhead itself. The protocol terminates when every node belongs to a cluster being either a clusterhead or an ordinary node and knows the role and clusterhead of all its neighbors. Except for the initial procedure, which is executed by each node when it starts the protocol operations, CC is message-triggered. We have six types of messages. 1) CH(v). Sent by a node v to declare that it will be a clusterhead. This is a broadcast message, i.e., a message that has to reach all v's neighbors. 2) NEIGHBORS(v,u). Sent by a node v to its neighbor u to communicate v's neighbor list. This is a unicast message from node v to node u. 3) ASSOC_REQUEST(v,u). Node v invites node u to be part of its cluster by sending to node u this unicast message. Nodes are invited according to their degree, i.e., to the number of their neighbors: Nodes with higher degree are invited first. 4) ASSOC_RESPONSE(u,z). Sent by a node v to its neighbor u to communicate v's response to u's association request. If the response is **yes** node v communicates to its clusterhead u to be part of u's cluster, $z = u$, otherwise node v communicates to a clusterhead u to be part of z's cluster, $z \neq u$. 5) ASSOC_FINISH(v). Sent by a node v to declare that it has concluded its cluster formation. This is a broadcast message. This message has the meaning of communicating that a clusterhead has completed its cluster. It will not try to enroll any other nodes. 6) JOIN(v,u). Sent by a node v to declare that now it is associated to clusterhead u. This is a broadcast message.

Lack of space prevents us from giving the actual pseudo code of the CC's procedure. Therefore we just describe the functioning of the algorithm through an example. (Details can be found in [13], where the correctness of CC is also proved.) Let us consider the ad hoc network depicted in Fig. 1(a), where the numbers associated to the nodes indicate both nodes ID and weight. All nodes execute procedure *InitCC* to determine whether they are the bigger nodes in their neighborhood. Being the heaviest nodes in their neighborhood, only nodes 11, 15 and 18 are init nodes, and hence they send a CH message. In particular, nodes 8, 9 and 10 receive the message CH(11) from their neighbor 11, nodes 16 and 17 receive CH(18) from 18 and nodes 12, 13 and 14 receive CH(15). All nodes 8, 9, 10, 12, 13, 14, 16 and 17 reply to the CH message by sending the list of their neighbors (with a message NEIGHBORS). Upon receiving NEIGHBORS$(8, 11)$, NEIGHBORS$(9, 11)$ and NEIGHBORS$(10, 11)$, node 11 starts to select nodes to affiliate them to its cluster. Node 8 is the neighbor with the higher degree; therefore, node 11 invites node 8 by unicasting the message ASSOC_REQUEST$(11, 8)$. Since node 8 has no clusterhead yet, it decides to affiliate to node 11, sends ASSOC_RESPONSE$(8, 11)$ to 11 and broadcasts the message JOIN$(8, 11)$. Node 11, after reception of ASSOC_RESPONSE$(8, 11)$ sees that 8 accepted its request and moves to invite other nodes. Upon receiving the message JOIN, neighbors of node 8 update information about node 8 and 11. Node 11, then invites first node 10 and then node 9, since inviting both of them would not violate the clique property. Node 10, like 8, accepts, by unicasting ASSOC_RESPONSE$(10, 11)$, and sends the message JOIN$(10, 11)$ to all its neighbors. Later, node 9 accepts 11's invitation and informs all neighbors about enrolling in 11's cluster. At this point, 11 cannot invite other nodes and sends message ASSOC_FINISH(11) to all its neighbors. Since 11 now knows the role and the cluster of all its neighbors it can **EXIT** the execution of CC. Nodes 9 and 10 can also **EXIT** the protocol. Before exiting, node 8 has to wait for the decision of its neighbors. Similarly, nodes 15 and 18 form their clusters, containing nodes 12, 13 and 14 and 16 and 17, respectively. In the meanwhile, node 7 has been made aware of the role and the clusterhead of its heavier neighbor 8 (via a JOIN$(8, 11)$ message) and knows that node 8 will not be a clusterhead. Node 7 has no choice but becoming a clusterhead itself and hence it broadcasts



(a) A sensor network (b) Clique covering (c) A connected backbone

Figure 1. A WSN, the CC-induced clustering and a backbone connecting the clusters

the message CH(7) to which those neighbors that have not decided what to do (nodes 4, 5 and 6) reply by sending their neighbor lists. Based on these lists, node 7 will invite, in the order, nodes 4, 6 and 5 and then, after having updated its cluster, quits. By accepting node 7's invitation and by sending message JOIN(4, 7), node 4 "frees" node 3 from waiting any longer. The last cluster is formed with node 3 as clusterhead and nodes 1 and 2 as its affiliated nodes. The result of the execution of CC is shown in Fig. 1(b).

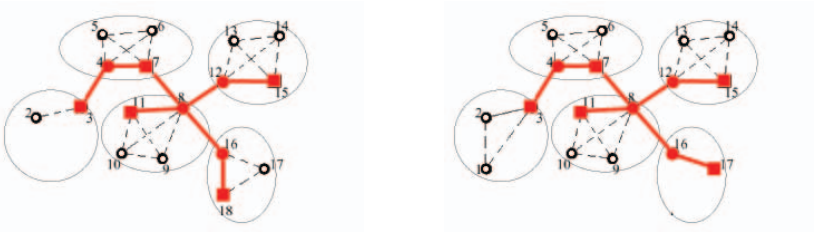
Backbone formation. The formation of a connected backbone starts as soon as the cliques are formed. At this time, every node knows the ID and weight of each neighbor as well as the ID and the weight of the clusterhead to which each neighbor is affiliated. This information is enough for each node to compute the paths to clusterheads different from its own that are one and two hops away. For each clusterhead one path is selected. In particular, if a node is one hop away from a clusterhead different from its own, this path is chosen. Otherwise, a path with an intermediate node to another clusterhead is chosen, preferring an intermediate node with greater weight (ties are broken based on the ID). Finally, if the node is not a clusterhead, it informs its own clusterhead of the information just gathered.

Once a clusterhead has determined its own connection information and has received this information from its own nodes, it can determine the best paths to all the clusterheads that are at most three hops away. This is the *necessary* and *sufficient* condition to form a connected backbone (the proof closely follows the lines of that in [14]). More specifically, clusterheads that are one hop away are joined by their direct link. Clusterheads that are two hops away are connected via the common neighbor with the greatest weight (ties are broken based on the unique nodal ID). Finally, clusterheads that are three hops away are joined by selecting two *intermediate* nodes between them. The selection of these intermediate nodes between clusterheads c_1 and c_2, with $w_{c_1} > w_{c_2}$,[1] is performed as follows. Among all nodes in c_2 cluster that are two hops away from c_1, the heaviest is selected. Among all its neighbors in c_1's cluster, the heaviest is selected. When a clusterhead v has calculated the routes to all the other clusterheads up to three hops away, it sends this information to the nodes in its cluster. Therefore, each node knows if it is used as gateway to interconnect to adjacent clusters, and how to reach neighboring clusterheads.

[1] Without loss of generality we assume from now on that all nodal weights are different. If this would not be the case, as mentioned earlier, ties can be broken by using the unique nodal ID.

The following example shows how a backbone is built from the cluster partition depicted in Fig. 1(b). Nodes 1 and 2, like 5 and 6, 13, 14 and 17 and 9 and 10 have no neighbors that are affiliated to a different clusterhead than their own. Each of them unicasts this information to its clusterhead via an INFO message. Node 4, affiliated to clusterhead 7, knows that it can reach clusterhead 3 directly. Node 4, therefore unicasts this information to its clusterhead. Node 8, affiliated to clusterhead 11, knows that it can reach clusterhead 7 directly, and that it can also reach clusterhead 15 through node 12 and clusterhead 18 through node 16. Node 8 unicasts this information to its clusterhead. Similarly, node 12, affiliated to clusterhead 15, and node 16, affiliated to clusterhead 18, know that they can reach clusterhead 11 through node 8. Each of them unicasts this information to its clusterhead. When clusterhead 3 receives INFO from all nodes in its cluster, it knows that the only reachable clusterhead is 7 through node 4. Therefore it unicasts this information to all nodes in its cluster. When clusterhead 7 receives INFO from all the nodes in its cluster, it knows, from INFO of node 4, that it can reach clusterheads 3. It also knows that it can reach clusterhead 11 through node 8. Node 7 unicasts this information to all nodes in its cluster. When clusterhead 11 receives INFO from all the nodes in its cluster, it knows, from INFO of node 8, that it can reach clusterhead 7 using only node 8 as intermediate gateway and that it can reach clusterheads 15 and 18 using two intermediate gateway, nodes 8 and 12 for clusterhead 15, and nodes 8 and 16 for clusterhead 18. Node 11 unicasts this information to nodes in its cluster. Similarly, when clusterheads 15 and 18 receive all INFO from nodes in their cluster, they know, from INFO of node 12 and 16, respectively, that they can reach clusterhead 11 using node 12 for clusterhead 15, and node 16, for clusterhead 18, as intermediate gateways with node 8. Both clusterheads unicast this information to nodes in their cluster. At the end of the backbone formation procedure, clusterhead 3 and 7 know that they can reach each other using node 4 as intermediate gateway. Clusterheads 7 and 11 know that they can reach each other using node 8 as intermediate gateway. Clusterheads 11 and 15 know that they can reach each other using nodes 8 and 12 as intermediate gateways. Clusterheads 11 and 18 know that they can reach each other using nodes 8 and 16 as intermediate gateways. The result of the backbone formation on the clustering of Fig. 1(b) is shown in Fig. 1(c), where solid nodes and links are those in the backbone.

Backbone reorganization. Perhaps, what makes the CC solution most interesting is the ability of the nodes in the backbone to reorganize themselves quickly and efficiently when one of the



(a) Node 1 fails: Nothing changes

(b) Clusterhead 18 fails: 17 is the new clusterhead

Figure 2. Backbone reorganization

backbone nodes fails. Here we give a brief overview of the actions undertaken by a node after the failure of a neighboring backbone node through the following example. (The actual "clique advantage" will be then made clear in the next section, where we quantify in terms or time and overhead the difference between CC and previous solutions.) Let us consider the cluster partition depicted in Fig. 1(b). Assume now that node 1 fails. Node 2 and clusterhead 3 are made aware of the failure, and remove 1 from the list of nodes in their cluster. Node 1 is an ordinary node so no other reorganization or information exchange is needed. The result of the reorganization procedure is depicted in Fig. 2(a). Node 3's clusterhead is still a clique and the backbone is still connected. Assume now that clusterhead 18 fails. In this case nodes 17 and 16 are made aware of the failure, and remove node 18 from the list of nodes in their cluster. Nodes 17 and 16 then need to check if they have to forward the failure information. Node 17, not being a backbone node does not have to forward any information. However, node 16 needs to send a DEAD message to node 8 because it was used with node 8 to connect clusterheads 11 and 18. Then each node checks if it is the better node in the cluster to act as clusterhead. Node 16 knows that node 17 is better and therefore does not promote itself to clusterhead. Node 17 instead knows that it will become the new clusterhead and broadcasts this information via a CH message asking also for connection informations with a INFO_REQUEST message. When node 8 receives the DEAD message from node 16 it updates its information about node 18, forwards this information to its clusterhead 11 and locally records that node 16 must select a new clusterhead. When node 16 receives the CH message sent by 17 it replies broadcasting a JOIN message to make aware all neighbors about its new clusterhead. When it receives the INFO_REQUEST message it replies with an INFO message saying that it can reach clusterhead 11 through node 8. When node 11 receives the DEAD message sent by 8 it updates its information about node 18 and sends an INFO_REQUEST to nodes in its cluster to try to find a route to the new clusterhead. Node 8, after the reception of the JOIN message sent by node 16, knows the new clusterhead of node 16 and can reply to its clusterhead saying that it can reach the new clusterhead 17 through node 16. Now, clusterheads 11 and 17 know how to reach each other. At the end of the reorganization procedures the new clique is created and the backbone is still connected (Fig. 2(b)). Only local exchange of information are needed to react to a clusterhead failure.

The case of gateway failure is treated similarly to the failure of a clusterhead. In this case, it is not needed to find a new clusterhead, and only the INFO_REQUEST and INFO messages are exchanged to find routes to connect the clusterheads.

IV. EXPERIMENTAL RESULTS

We investigated the performance of the proposed protocol CC via simulations. More specifically, we implemented CC using the VINT project network simulator (ns2) [15]. We also implemented the protocol DMAC [3] for comparing CC with a solution which is able to cope with nodal failure.

Simulation scenarios. Our implementation is based on the CMU wireless extension to ns2, i.e., on the IEEE 802.11 MAC with the DCF. The standard parameters have been modified to take sensor nodes characteristics into account (e.g., shorter transmission radius and different values for the energy model). The sensor deployment area is a square of side L where n static wireless nodes are randomly and uniformly scattered. The transmission radius of each node is set to 35 meters. Two nodes are neighbors if and only if their Euclidean distance is less than or equal to the transmission radius. In our simulations, the number of nodes n varies among the values 50, 100, 150, 200, 300, and 400, while L has been set to 200m. This allows us to test our protocol on increasingly dense networks, from (moderately) sparse networks with average degree equal to 4 to dense networks where the average degree is around $33,5$.

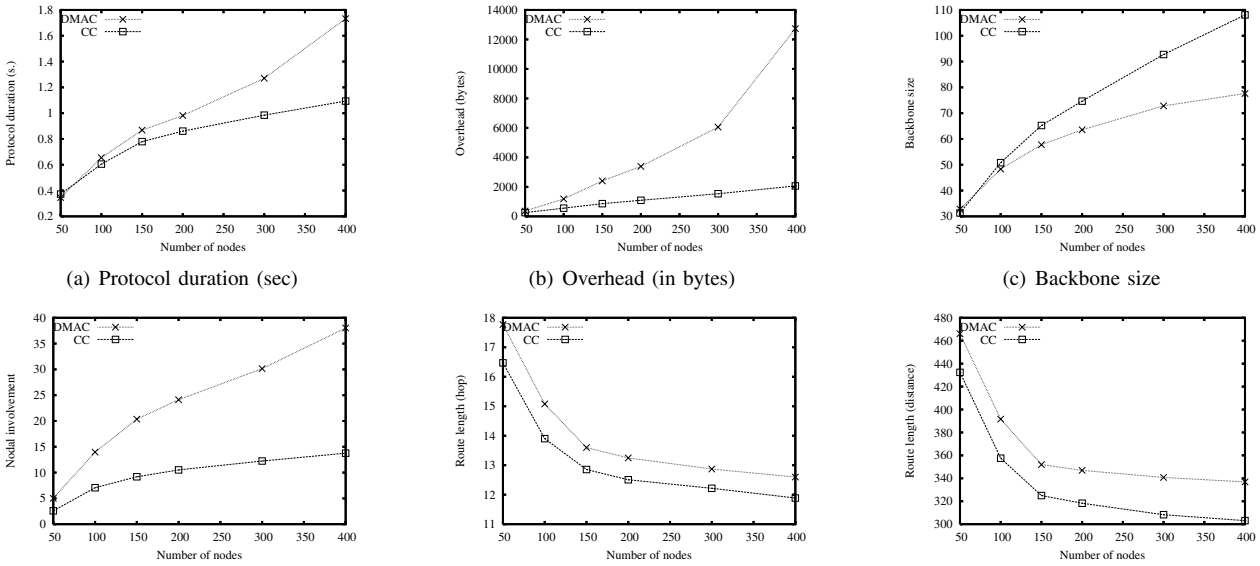
The effectiveness of CC and DMAC in reacting to nodal failures has been measured in scenarios where an initial clustering and backbone formation have been performed, and then nodes are removed from the network randomly and uniformly until the network itself is disconnected. (This happens, on average, after 35 to 50% of the nodes have been removed, in networks from 50 to 400 nodes, respectively.) In this setting, we have considered the following metrics:

(a) Protocol duration (in seconds), i.e., the time needed by each protocol to complete cluster reorganization and backbone connection after a node is removed.

(b) The overhead (in bytes) associated to the protocol operations for reorganization. These are physical layer measurements, which account for collisions and for the corresponding automatic packet retransmissions at the MAC layer.

(c) Nodal involvement. This is the number of nodes involved in cluster and backbone reorganization as a consequence of the removal of a node. This metric gives a measure of how resilient a protocol is to the chain reactions triggered by a node removal.

(d) Backbone size, i.e., the fraction of the network nodes that are in the backbone.



(a) Protocol duration (sec) (b) Overhead (in bytes) (c) Backbone size

(d) Nodal involvement (e) Route length: In hop (f) Route length: Distance

Figure 3. CC vs. DMAC: A performance comparison

(e) Route length over the subgraph G_b of the network topology graph induced by the backbone construction.

The corresponding graphs are shown in Fig. 3. Every point in the figures has been obtained by averaging over 250 experiments for each network size. The resulting statistical confidence is 95%, with a 5% precision.

Fig. 3(a) shows the average time needed to CC and DMAC to reorganize the backbone after a node failure. The picture clearly shows that CC achieves a significant improvement. For instance, DMAC needs 63% more time than CC to reorganize after a failure in dense networks. This is due to the CC design choice of involving in the reorganization only nodes in the proximity of the failing one. This is not the case for DMAC, where the failure/removal of a node can trigger a chain of role changes that could involve nodes far away in the network. We have quantified this in Fig. 3(d), which depicts the number of nodes that are involved in a network reorganization (i.e., that exchange messages because of the failure). The number of nodes exchanging traffic after a node failure is up to 176% higher in DMAC than in CC. Overall, the nodes involved in the reorganization are about 10% of all the network nodes in DMAC, while only from 3 to 5% of them are required to react to a nodal failure in CC. The difference in the time needed by DMAC and CC to reorganize the network is even more evident if we consider the maximum reorganization duration (as opposed to the average). CC reorganization always terminates within 4s, while DMAC can last up to 30s!

Time is not the only drawback of using DMAC. This protocol also imposes higher overhead for backbone maintenance (Fig. 3(b)). After a failure, the number of bytes that need to be transmitted by DMAC nodes is almost seven times higher that that needed by CC in networks with 400 nodes. The reason is twofold. On one side, DMAC involves more nodes in the reorganization. On the other side, in DMAC it may happen that messages are exchanged to reflect a role change that is only temporary (e.g., a node which proposes itself as clusterhead might later realize that there is a better candidate to serve in that role). The need to exchange only local information and the fact that each node has a correct local view of the network and makes stable decisions on role changes, explains why CC shows lower overhead after a failure. The rest of our experiments concerns the structure of the backbone generated after nodal removal by the two protocols. We have investigated metrics such as 1) the size of the backbone (which has an impact on the percentage of nodes that can operate at a lower duty cycle without affecting the communication service); 2) the number of clusters, the distribution of their size, and the number and types of gateways selected for cluster interconnection, and 3) the length of the backbone routes (which has an impact on the data latency). Due to lack of space here we focus on backbone size and route length, only briefly commenting on the other metrics. Fig. 3(c) compares the size of the backbones generated by CC and DMAC on network topologies with a number of nodes varying between 50 and 400. The figure clearly shows the drawbacks of a clique-based approach. Having to meet the clique property clusters in CC tends to be smaller. 50% of the clusters have a size less than or equal to 6 (8) in CC when $n = 300$ ($n = 400$). The cluster size is instead higher in DMAC. In networks with 300 nodes half the clusters have a size ≤ 10. The size grows to 13 when $n = 400$. Having partitioned the nodes in smaller clusters, CC also needs a higher number of gateways to interconnect each cluster with all the adjacent ones. This is confirmed by the fact that the number of adjacent clusters is 48% higher in CC than in DMAC when $n = 400$. Being the number of clusters and the number of gateways higher, CC produces an overall bigger backbone (up to 39% bigger) than DMAC. On the other hand the possibility to communicate directly between nodes belonging to the same clique and the high density of the resulting backbone favor short routes in a CC structure. Fig. 3(e) shows the average route lengths on the backbone (in number of hops), while Fig. 3(f) depicts the average distance traversed by a route. DMAC produces backbones with

7% longer routes than those on a CC backbone. The difference is even more evident if we consider the distance traversed by a route (12% shorter in CC). In CC packets traverse a lower number of shorter hops, imposing lower energy consumption for delivering packets to the sink.

For its characteristics, CC turns out to be a promising solutions for organizing a WSN hierarchically. Its only drawback might be the increased backbone size, with respect to the solution provided by DMAC. When this metric is particularly important, the CC backbone can be made smaller by breaking some of its cycles, thus obtaining a slimmer structure which is still connected. We have previously proposed an efficient general method for backbone "sparsification" in [4]. In our experiments we have also applied this technique to CC. Preliminary results show that it is possible to get a backbone which is up to 23% smaller than DMAC's maintaining the low overhead and fast reorganization features of the original CC protocol. As expected, this comes at the price of longer route lengths (comparable to those of DMAC) and of a less robust backbone.

V. CONCLUSION

We have described a new distributed and localized protocol for clustering and backbone formation in wireless sensor networks. *Clique clustering* (CC) partitions the network into cliques, thus providing each cluster with a fail-safe mechanism for quickly reconfiguring itself when nodes fail or leave the network. Backbone formation and reorganization techniques render CC a solution for hierarchical organization of WSNs that efficiently deals with node removal/failure. We have compared CC with DMAC, a previous proposed protocol for clustering and backbone formation and reorganization. Simulations results show that CC is effective in reorganizing clusters and backbones very quickly and with remarkably lower overhead than DMAC. We intend to investigate possible extensions of CC aimed at making it resilient to nodal mobility and to the addition of new nodes to the network.

VI. AKNOWLEDGEMENTS

This paper is based upon work partially supported by the MIUR International FIRB RBIN047MH9 (Tecnologia e scienza per le reti di prossima generazione) and the EU Integrated Project e-SENSE (IST-4-027227-IP).

REFERENCES

[1] A. Marcucci, M. Nati, C. Petrioli, and A. Vitaletti. Hierarchical routing and data aggregation in wireless sensor networks. Technical report, Università di Roma "La Sapienza", Roma, Italy, October 2003.

[2] S. Basagni, A. Carosi, and C. Petrioli. Sensor-DMAC: Dynamic topology control for wireless sensor network. In *Proceedings of the* 60*th IEEE Vehicular Technology Conference, VTC 2004 Fall*, volume 4, pages 2930–2935, Los Angles, CA, September 26–29 2004.

[3] S. Basagni. Distributed clustering for ad hoc networks. In A. Y. Zomaya, D. F. Hsu, O. Ibarra, S. Origuchi, D. Nassimi, and M. Palis, editors, *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, pages 310–315, Perth/Fremantle, Australia, June 23–25 1999. IEEE Computer Society.

[4] S. Basagni, M. Mastrogiovanni, A. Panconesi, and C. Petrioli. Localized protocols for ad hoc clustering and backbone formation: A performance comparison. *IEEE Transactions on Parallel and Distributed Systems, Special Issue on Localized Communication and Topology Protocols for Ad Hoc Networks*, 17(4):292–306, April 2006.

[5] S. Basagni, A. Carosi, and C. Petrioli. Mobility in wireless sensor networks. In A. Boukerche, editor, *Algorithms and Protocols for Ad Hoc and Sensor Networks*. Wiley's & Son, Inc., New York, NY, 2007. To appear.

[6] R. Ghosh and S. Basagni. Mitigating the impact of node mobility on ad hoc clustering. *Wiley InterScience's Wireless Communications & Mobile Computing, WCMC*, 8(3), March 2008. To appear.

[7] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM Computer Communication Review*, 27(2):49–64, April 1997.

[8] Y. Fernandess and D Malkhi. *k*-clustering in wireless ad hoc networks. In *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing, POMC 2002*, pages 31–37, Toulouse, France, October 30–31 2002.

[9] P. Tosic and G. Agha. Maximal clique-based distributed group formation for autonomous agent coalitions. In 3*rd International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2004*, New York, NY, USA, August 19–23 2004. IEEE Computer Society.

[10] R. Gupta and J Walrand. Approximating maximal cliques in ad-hoc networks. In *Proceedings of PIMRC 2004*, volume 1, pages 365–369, Barcelona, Spain, September 5–8 2004. Proceedings of the 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2004.

[11] K. Sun, P. Peng, P. Ning, and C. Wang. Secure distributed cluster formation in wireless sensor networks. In *Proceedings of the* 22*nd Annual Computer Security Applications Conference, ACSAC 2006*, pages 131–140, Washington, DC, USA, 2006. IEEE Computer Society.

[12] R. Gupta, J. Walrand, and O. Goldschmidt. Maximal cliques in unit disk graphs: Polynomial approximation. In *Proceedings of INOC 2005*, Lisbon, Portugal, March 2005. International Network Optimization Conference, INOC 2005.

[13] S. Basagni, C. Petrioli, and R. Petroccia. Clique clustering. Technical Report 09/2006, Dipartimento di Informatica, Università di Roma "La Sapienza", Roma, Italy, August 2006.

[14] I. Chlamtac and A. Faragó. A new approach to the design and analysis of peer-to-peer mobile networks. *Wireless Networks*, 5(3):149–156, May 1999.

[15] The VINT Project. *The ns Manual*. http://www.isi.edu/nsnam/ns/, 2002.