Adaptive Hierarchical Network Structures for Wireless Sensor Networks*

Dimitrios Amaxilatis[1,2], Ioannis Chatzigiannakis[1,2], Shlomi Dolev[3], Christos Koninis[1,2], Apostolos Pyrgelis[1,2], and Paul G. Spirakis[1,2]

[1] Computer Technology Institute & Press (CTI), Patras, Greece
[2] Computer Engineering and Informatics Department, University of Patras, Greece
[3] Department of Computer Science, Ben-Gurion University of the Negev, Israel
{amaxilat,ichatz,koninis,pyrgelis,spirakis}@cti.gr, dolev@cs.bgu.ac.il

Abstract. Clustering is a crucial network design approach to enable large-scale wireless sensor networks (WSNs) deployments. A large variety of clustering approaches has been presented focusing on various aspect such as minimizing communication overhead, controlling the network topology etc. Simulations on such protocols are performed using theoretical models that are based on unrealistic assumptions like ideal wireless communication channels and perfect energy consumption estimations. With these assumptions taken for granted, theoretical models claim various performance milestones that cannot be achieved in realistic conditions. In this paper, we design a new clustering protocol that adapts to the changes in the environment and the needs and goals of the user applications. We provide a protocol that is deployable protocol in real WSNs. We apply our protocol in multiple indoors wireless sensor testbeds with multiple experimental scenarios to showcase scalability and trade-offs between network properties and configurable protocol parameters. By analysis of the real world experimental output, we present results that depict a more realistic view of the clustering problem, regarding adapting to environmental conditions and the quality of topology control. Our study clearly demonstrates the applicability of our approach and the benefits it offers to both research & development communities.

Keywords: Algorithm Engineering, Clustering, Self-Stabilization, Implementation, Protocols, Software Design, Cross-layer, Cross-platform.

1 Introduction

During the last decade, *wireless sensor networks* (WSNs) gained the interest of computer science, industries and academia, not only from theoretical but also from practical perspectives [22]. Consisting of spatially distributed autonomous sensor-equipped devices, WSNs allow the cooperative monitoring of physical or environmental conditions (e.g., temperature, light, pollutants, etc.), enabling a multitude of applications in both urban and rural contexts.

* This work has been partially supported by the European Union under contract numbers ICT-2008-215270 (FRONTS) and ICT-2010-258885 (SPITFIRE).

D. Simplot-Ryl et al. (Eds.): ADHOCNETS 2011, LNICST 89, pp. 65–80, 2012.

Current WSN technologies used by the vast majority of off-the-shelf sensor nodes allow short range message exchanges. They employ flat network organization structures for message exchanges, data aggregation and actuators operation. Thus, they typically allow the operation of a few dozens of nodes.

Many of the proposed applications assume large node populations densely deployed over sizable areas. Although thus far, we have only a few examples of large-scale deployments of such systems, we are currently seeing great advances, as signified by research projects such as CitySense [12] and SmartSantander [29].

It is therefore important that future WSN have scalable network structures that achieve appropriate levels of organization and integration. This organization and integration needs to be achieved seamlessly and with appropriate levels of flexibility, in order to be able to accomplish their global goals and objectives. And it needs to be done in a proactive way to meet the current or anticipated needs of their "users". For this reason, they need to adapt to the changes in their environment and change their internal organization by communicating, cooperating and forming goal-driven sub-organizations.

Since [4], grouping sensor nodes into clusters has been widely pursued by the research community in order to achieve network scalability and fault-tolerance. A large variety of approaches has been presented focusing on different performance metrics. Some have been proposed as stand alone methods (e.g., [19]), others incorporated as sub protocols in larger solutions designed to solve more specific problems such as query execution, aggregation, localization etc. (e.g., [20,32]).

Unfortunately, even though all of them have many potential applications, extremely few software implementations for real sensor nodes is available to the community [33,20,19]. Furthermore, none of them has been widely adopted by the community. This is partially because cluster formations remain static throughout the execution of the network. Thus, it is difficult to react to "external changes" that affect the topology of the network (e.g., due to node failures) or to "internal changes" requested by the application (e.g., to reduce cluster sizes). Sudden variations of service requests or environmental physical conditions or of motion of nodes disrupts the system from serving its goals.

Clearly, technology expects future WSN to be dependable and adaptive to: the user needs, sudden changes of the environment and specific applications characteristics. This means that the system continues to operate in a set of desired states with maintained, or gracefully degraded or even improved quality of service. In order to design such systems we adopt the concept of [3] for self-organization that has been widely mentioned in the scope of distributed computing and peer to peer networks. We consider self-stabilizing distributed algorithms for cluster definition in communication graphs of bounded degree processors as well as for hierarchical distributed snapshots. We observe that as far as dynamic changes are limited within a cluster they do not affect the rest of the network.

We implement our solution by following a component-based design. We totally avoid implementing our algorithm as a monolithic, stand-alone piece of code. Thus, our code promotes **exchangeability** of different modules that interact using

well-defined interfaces. The modules can be easily integrated as sub-protocols in other problems such as energy conservation, routing, role assignment, security etc.. Furthermore, we use the Wiselib [8]: a code library, that allows implementations to be OS-independent. It is implemented based on C++ and templates, but without virtual inheritance and exceptions. All implemented algorithms are **platform independent** as they can be compiled on a number of different hardware platforms (e.g., TelosB, iSense, ScatterWeb) and **OS independent** as they can be automatically used in systems implemented using C (Contiki), C++ (iSense), and nesC (TinyOS).

We conduct a thorough evaluation using an experimental testbed environment. For all cases, our results indicate that our approach adapts to the external and internal changes. The results of the evaluation also indicate that our implemented code achieves high **scalability** and **efficiency**. To the best of our knowledge, this work is among the very few that conducts experiments and assess the practicality of a clustering approach in real WSN.

2 Previous Work

A large variety of clustering algorithms has been presented during the past years. In the relevant bibliography there exist several surveys and tutorials (e.g., [1,25,21,11]) that attempt to categorize and classify the various protocols based on the design choices and the mode of operation.

In terms of *Cluster-head selection* in some algorithms, like [18,6] each node is assigned a probability p of becoming a cluster-head. In algorithms like [27,5,32], some deterministic criteria like node connectivity, node identity and energy are respectively used for electing cluster-heads. Finally, algorithms like [9,14] are based on the combination of criteria in order to assign weights to nodes and decide the cluster-heads based on those.

In terms of *Node Grouping*, when a node is elected as cluster-head, it advertises itself to neighboring nodes in order for them to join its cluster. A node can decide to join a cluster based on various criteria. In most algorithms, e.g., [18,34,7], the main criterion for a node to join a cluster is the distance to the cluster-head. In other algorithms, like [31,28,9] the nodes decide to join a cluster-head based on some cluster-head attribute like remaining energy, time to live, etc..

In recent years, the concept of self-organization has been widely mentioned in the scope of distributed computing and peer to peer networks. Many works have claimed being self-organizing, but a mere fraction of these works also tries to give a specific definition of what self-organization really is. In [3] a framework for self-organization is proposed, including formal definitions of the self-organization concept and complementary proof techniques which can be used to prove that algorithms are indeed self-organizing.

Self-stabilizing and self-healing constructions of hierarchies, in the domain of sensor networks, appear in [35]. The authors divide the plane into hexagonal cells. In each cell, a head that corresponds with a cluster leader is elected. In [26], Wattenhofer and Moscibroda present an algorithm for computing a maximal independent set in radio networks where processors can broadcast their messages

asynchronously, but no collision detection mechanism is provided. Snapshot algorithms are used for recording a consistent global state of a distributed asynchronous system. A self-stabilizing snapshot algorithm was first introduced in [23], where repeated invocations of snapshots are used to ensure stabilization of a non-stabilizing algorithm. Following [23], several works have studied ways of achieving efficient snapshots in different models e.g., message passing, bounded links message passing and shared memory ([30,2,13]).

The initial point of our work is [15]. We are inspired by this protocol due to its inherent simplicity, ease of deployment, scalability and self-stabilizing properties. In this work, we move beyond this protocol's ([15]) abstract design. We provide a solution that resolves crucial issues such as bidirectionality and reliability of channel communication between the nodes, adaptive detection of changes to the network topology and the efficient usage of communication mechanisms. We propose specific algorithmic solutions and provide software components fine tuned for real WSN hardware. In contrast to [15] and to the majority of the previous related work, we evaluate the system in real testbeds to measure the performance of the resulting adaptive clustering protocol.

3 An Adaptive Hierarchical Network Structure

The main idea of the algorithm is to partition the nodes of the network into small clusters that are then merged to form bigger clusters and so on. Nodes continuously monitor the local topology. Based on this information, if they do not detect any cluster, they take the initiative to create a new one. If one or more clusters exist, they join one of these using some very simple criteria.

The clustering algorithm maintains the self-stabilizing properties analysed in [15]. It uses the network parameter k for controlling the size of the clusters. The value of k is set by the network operator during the deployment phase of the system and can be modified during the execution of the protocol. Essentially, the protocol adapts to the external requests by properly adjusting the cluster size so that they have a diameter of $2 \times k$. The adaptation to the new size requires $O(k)$ execution rounds (in [15] in order to measure the communication complexity they assume the existence of a global clock, however for the actual execution of the algorithm in real networks no such global clock is required).

3.1 Initialization Phase

Our algorithm follows the self-stabilization approach, so we do not assume any initialization phase. Hence, it is capable of starting from any configuration where the nodes of the network are set to any arbitrary state. Thus, some nodes may consider themselves as cluster heads, others may consider as members of non-existing clusters, etc.. Regardless of this initial arbitrary state, within a bounded number of steps, our algorithms converges to a stable configuration, i.e., a configuration where all nodes of the network participate in a valid cluster of $k - hop$ diameter. This is done regardless of the way that the devices are positioned

in the network area. In fact, as explained in the following section our algorithm will suitably adapt its operation to the physical topology of the network.

We do not assume any kind of global clock synchronization among the nodes of the network. Yet we assume that clocks of all nodes have similar (if not identical) drift rates. Thus they are capable of measuring the same amount of time for a given period.

In the sequel, for simplicity we assume that nodes are provided with unique identities. Still, in large-scale deployments this may not be guaranteed by the technical personnel involved during the network installation. Note that this problem can be easily solved by a very simple extension provided in [15] that uses random choice used to break symmetry between nodes (for further motivation, see the asynchronous version of the algorithm in [15]).

3.2 Neighbor Discovery

An important aspect of the algorithm is the ability to detect the current topology of the network. The purpose for detecting the topology is twofold. First, to discover the target nodes as they change their position within the network area. Second, to detect changes to the local connectivity so that the spreading of the traces is properly adjusted to frequent and/or significant topology changes.

A simple approach would be for each node to periodically broadcast beacon messages that include their unique id. For each received message beacon the algorithm considers the sender node as a neighbor. Similarly, if a node stops receiving beacons from a neighbor, it removes the node from the list of neighbors.

This simple approach has to deal with the fact that communication is carried out via a wireless channel. Naturally, the quality of the wireless channel is subject to a number of environmental effects and thus its quality varies over time. This means that some beacons from neighboring nodes may not be received, while the topology has not changed. The algorithm falsely translates this as if the topology has changed. Accordingly, the perceived neighborhood of the node is changed. Then, when the temporary degradation of channel quality is restored, the beacons are received again, and the nodes are reinstated in the neighborhood. Hence, the perceived neighborhood of the node is changed once again. This series of events is temporary, does not reflect the actual state of the neighborhood and forces the algorithm to take unnecessary actions for adaptation.

Our approach is to take into account the Link Quality Indicators (LQI) provided by the MAC layer for each received message beacon. The goal is to filter out the neighbors that have poor communication channel quality. The nodes will consider the broadcaster as a neighboring node only if it receives a number of consecutive beacon messages with LQI above a certain threshold. In order to prevent the occasional short channel degradation from negatively impacting the above strategy, we allow a node to miss a number of beacons within a given period of time before removing it (called the *timeout period*). We also set a second LQI threshold; message beacons with LQI below this threshold are dropped

(and therefore count towards the number of beacons that are missed). In Sec. 5.1, we report a preliminary set of experiments we conducted in order to fine tune these LQI thresholds and the timeout period.

Another crucial aspect of the neighbor discovery operations of the algorithm is the periodicity of the broadcast beacons. One would expect that there exists a linear correlation between the time required for the neighbor discovery to stabilize (i.e., correctly detect passing by nodes) and the beacon interval. Reducing the beacon interval period (i.e., increasing the rate of transmission) should lead to a quicker response to changes in the topology (i.e., shorter delays in detecting a mobile node and potential changes to the neighboring nodes). Interestingly, the experiments reported in Sec. 5.1 indicate that there is a lower bound in the beacon interval period beyond which the network is congested with messages leading to instability in the operation of the algorithm. The experiments indicate a suitable value around $1000ms$ and $2000ms$. Our experiments indicate that these values are very suitable for the proper operation of the algorithm. However, in some cases we may wish to have faster response times. For this reason we allow the mobile nodes to have a different beacon interval period than the rest of the static nodes. This way, we can set the beacon interval to be as low as $100ms$. This essentially allows us to have a very low response time without creating a large overhead on the wireless channel. In fact, the response time achieved is much lower than other RF technologies, like, e.g., Bluetooth devices that has an interval period of about `12sec` [17].

3.3 Leader Election Phase

Each node u maintains an internal list with all the leader nodes that are within $k - hop$ distance. This list is continuously broadcast to all neighboring nodes by piggy packing it in the periodic beacons of the neighbor discovery module.

A node u that has an empty list decides to nominate itself as a local leader and insert in the list the entry $\{id_u, dist_u = 0, null\}$.

When a node v that receives a list from a neighboring node u, it processes it as follows: for each entry $\{id_u, dist_u = 0, null\}$ it adds $\{id_u, dist_u = 1, v\}$; for each entry $\{id_x, dist_x, u\}$ it adds $\{id_x, dist_x + 1, v\}$. After processing the incoming list, it drops duplication entries and merges the conflicting entries (in which the id is the same) as follows: the node chooses the entry with the minimum id with the minimal dist (further ties are broken using the parent value).

Remark that the above update algorithm is a simplification of the one presented in [15]. Each node v maintains an internal array which consists of the most recent topology list received from each neighboring node u. The computation of v's topology list is done on the basis of this list. Furthermore, in the validation phase we also delete entries with $dist > k$. Consequently, v's list will reflect its neighborhood up to distance $k - hops$ from u. The correctness of the revised update algorithm is trivially preserved, and the convergence time is $O(k)$ rounds.

3.4 Clusters Construction Phase

As soon as a node nominates itself as a local leader it enters a waiting period of $O(k)$ period of time. Then it waits for the self-stabilizing update algorithm to collect the other identifiers and notify for the leader identity all nearby nodes within at most $O(k)$ rounds. If there does not exist a node u with distance less than k from v, with lower id than v, then v is a stable leader and initiates the cluster construction phase. If another node u is identified (With lower id) then v exits the waiting period and becomes passive.

Next, each active local leader starts a breadth-first search to identify all nearby nodes and invite them in its cluster. Nodes receiving the search message of local leader u respond by joining the cluster of the leader. The leaders define the cluster structure and since each node v may follow a different local leader in its neighborhood, if v decides to join the cluster formed by node u it sends back to u a response message. This process requires an additional $O(k)$ rounds.

The above algorithm is self-organizing based on the arguments used in [15] and the convergence time is $O(k)$ rounds.

4 Component-Based Implementation

4.1 Generic Implementation Using Wiselib

More often than not, in Theoretical Computer Science, researchers tend to design an algorithm in an abstract way. This happens because an algorithm should be able to be used in many different situations and it is up to the developer to decide the way it should be turned into code for a real system. Algorithm Engineering requires the algorithm developer to actually implement algorithms. This step from theory into practice is often considered hard and requires programming skills in addition to knowledge in algorithm theory. Almost every time the developer finds many limitations in the ways she can operate within the given hardware and software specifications. These problems are further augmented when implementing algorithms for wireless sensor networks due to the extremely limited resources and also due to the heterogeneous nature (both in terms of hardware and software). This also explains why many theorists, having only little Software Engineering experience, never engage in Algorithm Engineering. This situation is particularly alarming in distributed embedded systems.

We decided to implement our algorithm using Wiselib [8]: a code library, that allows implementations to be OS-independent. It is implemented based on C++ and templates, but without virtual inheritance and exceptions. Algorithm implementations can be recompiled for several platforms and firmwares, and even for simulators without the need to change the code. In its current version Wiselib can interface with systems implemented using C (Contiki), C++ (iSense), nesC (TinyOS), Android (via the C/C++ NDK) and iPhone OS.

An important feature of Wiselib are the already implemented algorithms and data structures. Since different kind of hardware uses different ways to store data (due to memory alignment, inability to support dynamic memory, etc.), it is

important to use these safe types as much as possible since they have been tested before on most hardware platforms. In its current version, Wiselib includes about 60 Open Source implementations of standard algorithms including Localization algorithms, Cryptographic schemes, Distributed data structures etc..

Wiselib also runs on the simulator Shawn [16] and TOSSIM [24], hereby easing the transition from simulation to actual devices. This feature allows us to validate the faithfulness of our implementation and also get results concerning the quality of our algorithms without time consuming deployment procedures and harsh debugging environments. Furthermore, apart from ordinary sensor node targets or simulation environments, it is also possible to run Wiselib code directly on a PC. The PC therefore acts as a sensor node, but without any code space or execution speed limitations. On the one hand, there is basic OS functionality provided, such as a timer for event registration, or a clock providing the current time. On the other hand, it is possible to connect an IEEE 802.15.4 device to the PC, so that the PC can directly communicate with other sensor nodes.

4.2 Components Description

The component-based design that we propose is depicted in Fig. 1. We partition the logic of the clustering algorithm into three pieces with clear boundaries in terms of functionality provided. Each partition is designed so that it can progress its work in a relatively independent manner while ensuring the correct functionality of the algorithm. Clean interfaces are provided so that the partitions can easily communicate, fast and without heavy information exchange.

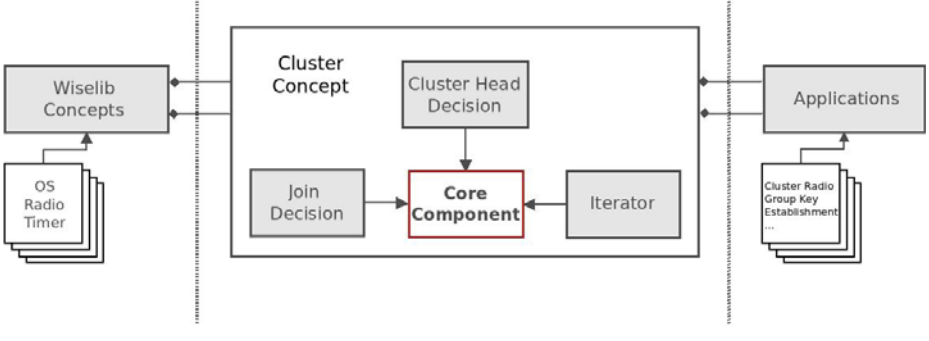


Fig. 1. Basic components and relation with Wiselib

Cluster-Head Decision (CHD). The first partition that we propose is related to the cluster-head selection process. We wish to implement the leader election mechanism as a single, stand-alone, software component. The component uses *call-backs* to the NEIGHBORHOOD DISCOVERY component so that it is re-executed whenever a change is detected in the node's neighborhood.

Join Decision (JD). The second partition is related to the methodology by which nodes decide to join cluster-heads. This component constructs the necessary payloads for the JoinRequest/JoinDeny/JoinAccept messages and it determines if a node will join a cluster when a JoinRequest message is received. The decision is based on the distance of the leader node and the ID of the node as described in Sec. 3.4.

Iterator (IT). The third partition is related to the organization of the nodes while clustering decisions are made by each node. This component is responsible for categorizing and storing neighbors into nodes that have already joined the cluster, nodes that have not joined the cluster yet and nodes that have joined another cluster. Collected information is maintained in *membership tables* by the IT component. These tables are of crucial importance for the algorithms that will be executed on top of the clustering – they are necessary to ensure cross-layering. This component also monitors the node's neighborhood and updates the *membership tables* based on observed changes to the network. This information can be used from the other components. For example, changes in the neighborhood that indicate that a node has left a cluster can trigger a new join decision process from the JD component.

The above three components are used by the main component which we call the **Core Component (CC)**. It is the kernel of our architecture that controls and coordinates all other components so that clusters are properly formed and maintained. The CC provides a public interface for other algorithms to take advantage of the resulting network organization. In the following we present the life-cycle of CC when forming a new cluster.

1. CHD is invoked to determine if the node will become a cluster head or not.
2. If the node is a cluster-head: JD is invoked to send JoinRequest messages to nearby nodes and invite them to the cluster. The JoinRequest messages can be sent to all available nodes using Broadcast messages or to selected nodes using the selected nodes ids.
3. Upon receiving a *Join Request* message, CC isolates the message's payload and passes it to JD.
   If JD decides to join, a JoinAccept message is sent to the originator of the JoinRequest message, IT is notified of the address so for it to be saved as the node's *Cluster-head*.
   If JD decides not to join, a JoinDeny message is generated along with a payload from JD and passed to the originator of the JoinRequest message.
4. If a JoinDeny message is received, its payload is passed to JD to be examined, in case the neighborhood's conditions are of interest and the IT is notified in order to keep track of which neighbors have joined the cluster and which have not.
5. When all nodes have been examined the membership tables are generated by the IT and the process of cluster formation completes.

4.3 Implementation Details

In the following, we present a Wiselib concept for each one of four basic components. The design goal of the concept is provide clear interfaces so that the implementation can be easily used by other algorithms with minimum effort.

Core Component (CC) Concept. The CC concept takes as template parameters a set of components types such as RADIO, TIMER and DEBUG that are needed for sending messages, registering events and optionally printing debug messages. The most important parameters are the types for the CHD, JD and IT which the CC will use for the clustering algorithm. The first method that initializes the module also provides instances of the components that the module will use. Then we have two methods for enabling and disabling the module, which is useful when it should only be run in certain points in time. After the module is enabled, the FIND_HEAD() method is called and starts the cluster formation. Next, we have a method for setting the parameters of the algorithm, which also sets the parameters for every other component. Then, we have a method for registering a callback in order to get notifications upon events. Finally, CC provides a set of functions to access useful information such as the cluster id, the parent node(if any) etc.

The CC components also provides a public interface *that implements the Wiselib concept of Clustering* and thus provides the cluster's ID, the ID of the cluster-head, and also allows to register a function callback in order to be able to deliver events to external components whenever an change to the cluster occurs, e.g., when the node joined a new cluster, or a neighbor from different cluster was discovered, or a new cluster was formed etc..

Cluster Head Decision (CHD) Concept. In the CHD concept we have a method for setting the parameters (e.g., the probability value that the module will use). Additionally, there is the method for calculating if the current node is a cluster-head and a method to get this result.

Join Decision (JD) Concept. In the JD concept we have a method that gives the hop count from the cluster-head, after the node has joined a cluster. It also provides methods that set the payload for specific types of messages. The minimum requirement is three methods, for the JOINREQUEST, the JOINACCEPT and the JOINDENY messages. Finally, we have the method JOIN that is called with a new JOINREQUEST payload, and decides if it is going to join the cluster.

Iterator (IT) Concept. For IT, we provide methods for getting the cluster id and the parent of the node. Moreover, the NEXT_NEIGHBOR() method allows iterating through the neighborhood of the node. If the neighborhood information is not available, we can register a callback function that the Iterator will call to inform us about changes in the neighborhood.

5 Real Experiments

Recently, experimentally-driven research has become an instrumental tool in designing and optimizing novel networking applications. While simulations are still

important tools, they suffer from several imperfections as they make artificial assumptions on radio propagation, traffic, failure patterns and topologies. Especially in the domain of wireless sensor networks, which are embedded into the environment, applications strongly depend on real-world processes that are often a result of complex interactions and are extremely difficult to model accurately. In order to design robust applications, developers need appropriate tools and methods for testing and managing their applications on real hardware in large-scale deployments. However, testbeds are expensive to set up and to maintain, hard to reconfigure for a different experiment and usually feature a fixed number of nodes. A possible approach to deal with these issues is to federate smaller-scale testbeds to form a virtual unified laboratory. WISEBED[1] [10] provides such a federation of testbeds consisting of heterogeneous sensor nodes (such as TelosB, Mica2, iSense or Sun Spot equipped with different sensors) and a collections of tools and methods to cope with implementing protocols and applications for heterogeneous networks.

We ended up using the WISEBED testbeds at UNIGE, CTI and UZL that are comprised of iSense nodes. An iSense node provides an IEEE 802.15.4 compliant radio, a 32-bit RISC controller running at 16MHz, 96kbytes of memory, a highly accurate clock and a switchable power regulator. We used 26 iSense nodes in UNIGE, 20 iSense nodes in CTI and 20 iSense nodes in UZL. In all experiments we set the transmission power set at $-6dB$ to enforce one-hop neighborhoods in room resolution. Data samples are collected every one second via the USB connectivity. The debugging was encoded out-of-band and it did not affect the experiments.

Due to space limitations we here report the results from the UNIGE experiment. Similar results hold for the other testbeds.

5.1 Assessing Channel Quality and Its Effect on the Performance of the Neighborhood Discovery Module

We conduct some preliminary experiments in order to fine tune the neighbor discovery operation of the algorithm. This essentially includes the appropriate adjustment of the two LQI threshold pairs (see Sec. 3.2) and the periodicity of the broadcast beacon. As expected the stricter the LQI thresholds, the smaller and more stable the neighborhood sizes will be. While when relaxing the LQI thresholds, the neighborhood sizes increase but also the system become prone to channel quality fluctuations. We tested two different pairs of LQI thresholds: $(35, 75)$ and $(55, 95)$. The resulting neighborhood sizes are depicted in Fig. 2. We conclude that the LQI threshold pair $(55, 95)$ is more suitable for the experimental testbed used. For this range the algorithm generates stable neighborhoods within a short period of time and the resulting topology is dense enough to allow the proper spreading of the traces (as it shall be reported in the sequel).

[1] `http://www.wisebed.eu`

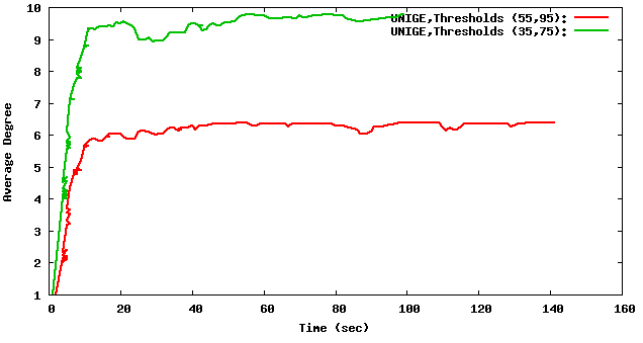


Fig. 2. Average neighborhood size with different LQI thresholds

We also examine the impact of beacon interval period and the neighbor timeout period in the detection of neighboring nodes. In Fig. 3 we test four different sets of beacon intervals. We measure the number of changes in the detected neighborhoods (i.e., the events) as the experiment evolves. It is evident that as the beacon interval increases, the neighbor discovery experiences fewer fluctuations. We believe that a beacon interval of $2000ms$ and above is a good trade-off between adaptivity and responsiveness to topology changes and induced overhead on the wireless medium.

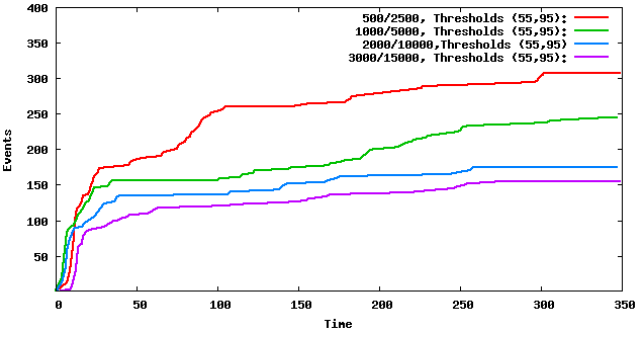


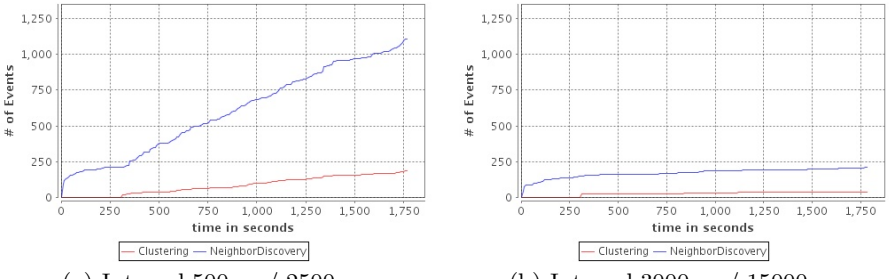
Fig. 3. Neighbor discovery for various beacon interval and timeout periods

5.2 Assessing the Speed and Quality of Adaptation

One would expect there exists a linear correlation between the time required for the module to stabilize (i.e., correctly detect the neighboring nodes) and the beacon interval. Reducing the beacon interval period (i.e., increasing the rate of transmission) should lead to a quicker response to changes in the topology (i.e., shorter delays in detecting changes to the neighboring nodes). Surprisingly, the results

show that our intuition was wrong, the experiments reveal that for small beaconing values (less than $1000ms$) the time the Neighbor discovery module needs to stabilize is longer. This leads to a larger number of events generated. This is caused by the excess traffic generated due to the short beacon interval, which itself creates interference that leads to losses of message beacons. Thus, many neighbors are falsely removed and then re-added to the neighborhoods. Compared to the $1000ms/5000ms$, the $500ms/2500ms$ Beacon Interval/Neighbor Timeout needs about 30% more time to stabilize and generates almost double more events.

In order to assess how this *period of stabilization* affects the performance of the other modules of the network layer. Essentially, we wish to investigate if a high rate of events prevents the other algorithms from stabilizing and thus functioning properly. We run 30 minute experiments using short beacon interval periods of $500ms/2500ms$ and long beacon interval periods of $3000ms/150000ms$. As observed in Fig. 4, for the case of $500ms$ period, we observe that the total number of events. The Neighbor discovery module wrongfully reports changes in the topology for such sort beacon interval and this leads the Clustering module to constantly attempt to adapt to the new state. However, when using $3000ms$ beacon interval, the communication channels reported by the Neighbor discovery module seems to be stable as the number of generated events is very limited.



(a) Interval 500ms / 2500ms (b) Interval 3000ms / 15000ms

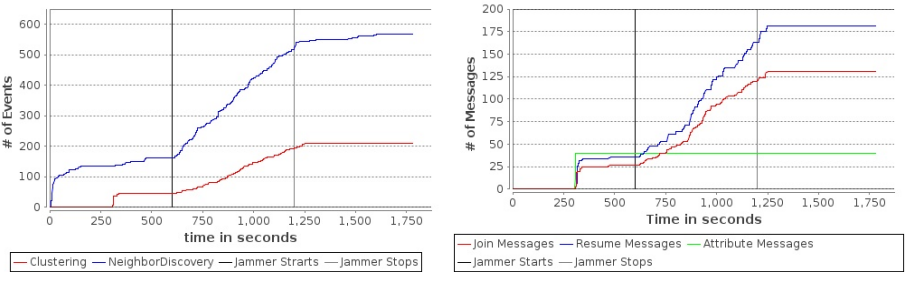
Fig. 4. Events generated by the Clustering mod. while reacting to events generaged by the Neighborhood discovery module

5.3 Assessing the Ability to Adapt to Channel Failures

Channel failures refer to a situation where a node is unable to successfully send most of its outgoing messages due to temporary noise on the wireless communication medium. We emulate such a behavior by introducing a node called "the Jammer" that continuously broadcasts big messages in order to create collisions, reduce link quality and in general reduce the message delivery rate. The Jammer has normal communication range, identical to all other nodes. We position it in such a way to disrupt almost 50% of the network.

The experiments conducted consisted of 3 stages. Firstly, the Adaptation and Clustering modules worked for 10 minutes to reach a stable state, then the Jammer was turned on for 10 minutes and finally the Jammer was turned off and the

network was left to stabilize again. As we can see in Fig. 5 the function of the Jammer heavily disrupts the smooth operation of both modules. During the channel disruption the Adaptation module continuously produces events and so does the Clustering module. Essentially, it creates the need to send more control messages in order to adapt to the new network state increasing the network traffic. When the disruption is over, network stabilizes and new events are rare.



(a) Total # Events generated (b) Messages generated

Fig. 5. Effect of channel failures on the performance of the Neighborhood discovery and Clustering modules

6 Conclusions

In this paper we have designed, implemented and extensively evaluated a clustering scheme for establishing adaptive hierarchical network structures for wireless sensor networks. We follow an experimental-driven research approach so that our system can be executed in real sensor networks. The extended experimental evaluation indicates that with proper fine tuning of the various algorithm's parameters, the resulting system can adapt to various internal and external events. In testbeds located in single-floor office spaces, stabilization was always reached within a very short period of time. In all cases the system returned to a stable state where all nodes participated in one of the formed clusters.

We believe that our experimental driven approach can be further optimized and for this we plan to conduct further experiments. We wish to examine mechanisms to adapt the protocol parameters as the system evolves dynamically. We wish to examine mechanisms that can adapt to concurrent events with heterogeneous performance parameters and in some cases conflicting goals. We also wish to combine our algorithm with specific application environments by exploiting the rich set of algorithm implementation provided by WISELIB pool of algorithms. One such application domain is the tracking of mobile assets in large-scale deployments. We believe that the hierarchical structure proposed here can be exploited to improve the scalability of the tracking process.

References

1. Abbasi, A.A., Younis, M.: A survey on clustering algorithms for wireless sensor networks. Comput. Commun. 30(14-15), 2826–2841 (2007)
2. Afek, Y., Dolev, S.: Local stabilizer. Journal of Parallel and Distributed Computing, Special Issue on Self-Stabilizing Distributed Systems 62(5), 745–765 (1997)
3. Anceaume, E., Défago, X., Gradinariu, M., Roy, M.: Towards a Theory of Self-organization. In: Anderson, J.H., Prencipe, G., Wattenhofer, R. (eds.) OPODIS 2005. LNCS, vol. 3974, pp. 191–205. Springer, Heidelberg (2006)
4. Baker, D., Ephremides, A.: The architectural organization of a mobile radio network via a distributed algorithm. IEEE Transactions on Communications 29(11), 1694–1701 (1981)
5. Baker, D.J., Ephremides, A.: A distributed algorithm for organizing mobile radio telecommunication networks. In: ICDCS, pp. 476–483. IEEE Computer Society (1981)
6. Bandyopadhyay, S., Coyle, E.J.: An energy efficient hierarchical clustering algorithm for wireless sensor networks. In: INFOCOM (2003)
7. Banerjee, S., Khuller, S.: A clustering scheme for hierarchical control in multi-hop wireless networks. In: INFOCOM, pp. 1028–1037 (2001)
8. Baumgartner, T., Chatzigiannakis, I., Fekete, S.P., Koninis, C., Kröller, A., Pyrgelis, A.: Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks. In: Silva, J.S., Krishnamachari, B., Boavida, F. (eds.) EWSN 2010. LNCS, vol. 5970, pp. 162–177. Springer, Heidelberg (2010)
9. Chatterjee, M., Das, S.K., Turgut, D.: Wca: A weighted clustering algorithm for mobile ad hoc networks. Cluster Computing 5(2), 193–204 (2002)
10. Chatzigiannakis, I., Fischer, S., Koninis, C., Mylonas, G., Pfisterer, D.: WISEBED: An Open Large-Scale Wireless Sensor Network Testbed. In: Komninos, N. (ed.) SENSAPPEAL 2009. LNICST, vol. 29, pp. 68–87. Springer, Heidelberg (2010)
11. Chen, Y.P., Liestman, A., Liu, J.: Clustering algorithms for ad hoc wireless networks. Ad Hoc and Sensor Networks 30, 2826–2841 (2007)
12. Citysense - An Open, Urban-Scale Sensor Network Testbed, `http://www.citysense.net/`
13. Cournier, A., Datta, A., Petit, F., Villain, V.: Enabling snap-stabilization. In: Proc. of the 23rd International Conference on Distributed Computing Systems, pp. 12–19 (2003)
14. Ding, P., Holliday, J., Celik, A.: Distributed Energy-Efficient Hierarchical Clustering for Wireless Sensor Networks. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 322–339. Springer, Heidelberg (2005)
15. Dolev, S., Tzachar, N.: Empire of colonies: Self-stabilizing and self-organizing distributed algorithm. Theoretical Computer Science 410, 514–532 (2008); FRONTS-TR-2008-22
16. Fekete, S.P., Kröller, A., Fischer, S., Pfisterer, D.: Shawn: The fast, highly customizable sensor network simulator. In: Proceedings of the Fourth International Conference on Networked Sensing Systems, INSS 2007 (2007)
17. Hay, S., Harle, R.: Bluetooth Tracking without Discoverability. In: Choudhury, T., Quigley, A., Strang, T., Suginuma, K. (eds.) LoCA 2009. LNCS, vol. 5561, pp. 120–137. Springer, Heidelberg (2009)
18. Heinzelman, W.R., Chandrakasan, A.P., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: 33rd IEEE Hawaii International Conference on System Sciences (HICSS 2000), p. 8020 (2000)

19. Iwanicki, K., van Steen, M.: Multi-hop Cluster Hierarchy Maintenance in Wireless Sensor Networks: A Case for Gossip-Based Protocols. In: Roedig, U., Sreenan, C.J. (eds.) EWSN 2009. LNCS, vol. 5432, pp. 102–117. Springer, Heidelberg (2009)
20. Iwanicki, K., van Steen, M.: On hierarchical routing in wireless sensor networks. In: Proceedings of the Eighth ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2009), IP Track, San Francisco, CA, USA, pp. 133–144 (April 2009)
21. Jiang, C., Yuan, D., Zhao, Y.: Towards clustering algorithms in wireless sensor networks: a survey. In: WCNC 2009: Proceedings of the 2009 IEEE Conference on Wireless Communications & Networking Conference, pp. 2009–2014. IEEE Press, Piscataway (2009)
22. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. John Wiley & Sons (2005)
23. Katz, S., Perry, K.: Self-stabilizing extensions for message-passing systems. In: Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, pp. 91–101 (1990)
24. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire tinyos applications. In: 1st ACM International Conference on Embedded Networked Sensor Systems (SENSYS 2003), pp. 126–137 (2003)
25. Mamalis, B., Gavalas, D., Konstantopoulos, C., Pantziou, G.: RFID and Sensor Networks: Architectures, Protocols, Security and Integrations. In: Clustering in Wireless Sensor Networks. Taylor & Francis Group (2009)
26. Moscibroda, T., Wattenhofer, R.: Maximal independent sets in radio networks. In: PODC 2005: Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing, pp. 148–157. ACM, New York (2005)
27. Parekh, A.: Selecting routers in ad hoc wireless networks. In: Proccedings of ITS, Rio-de-Janeiro, Brazil, pp. 420–424 (1994)
28. Selvakennedy, S., Sinnappan, S.: An adaptive data dissemination strategy for wireless sensor networks. IJDSN 3(1), 23–40 (2007)
29. SmartSantander - A unique in the world city-scale experimental research facility, `http://www.smartsantander.eu/`
30. Varghese, G.: Self-stabilization by counter flushing. SIAM Journal on Computing 30(2), 486–510 (2000)
31. Ye, M., Li, C., Chen, G., Wu, J.: An energy efficient clustering scheme in wireless sensor networks. Ad Hoc & Sensor Wireless Networks 3(2-3), 99–119 (2007)
32. Younis, O., Fahmy, S.: Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. IEEE Trans. Mob. Comput. 3(4), 366–379 (2004)
33. Younis, O., Fahmy, S.: An experimental study of routing and data aggregation in sensor networks. In: Proceedings of the IEEE International Workshop on Localized Communication and Topology Protocols for Ad Hoc Networks (IEEE LOCAN), pp. 50–57 (2005)
34. Youssef, A.M., Younis, M.F., Youssef, M., Agrawala, A.K.: Distributed formation of overlapping multi-hop clusters in wireless sensor networks. In: GLOBECOM. IEEE (2006)
35. Zhang, H., Arora, A.: Gs3: Scalable self-configuration and self-healing in wireless networks. In: Symposium on Principles of Distributed Computing, pp. 58–67 (2002)