An Experimental Study of Routing and Data Aggregation in Sensor Networks

Ossama Younis
Department of Electrical & Computer Engineering
University of Arizona, Tucson, AZ 85721, USA
E-mail: younis@ece.arizona.edu

Sonia Fahmy
Department of Computer Science
Purdue University, West Lafayette, IN 47907, USA
E-mail: fahmy@cs.purdue.edu

***Abstract*— Several sensor network applications, such as environmental monitoring, require data aggregation to an observer. For this purpose, a data aggregation tree, rooted at the observer, is constructed in the network. Node clustering can be employed to further balance load among sensor nodes and prolong the network lifetime. In this paper, we design and implement a system, iHEED, in which node clustering is integrated with multi-hop routing for TinyOS. We consider simple data aggregation operators, such as AVG or MAX. We use a simple energy consumption model to keep track of the battery consumption of cluster heads and regular nodes. We perform experiments on a sensor network testbed to quantify the advantages of integrating hierarchical routing with data aggregation. Our results indicate that the network lifetime is prolonged by a factor of 2 to 4, and successful transmissions are almost doubled. Clustering plays a dominant role in delaying the first node death, while aggregation plays a dominant role in delaying the last node death.**

I. Introduction

Several sensor network applications require an aggregated data value to be reported regularly. For example, in a habitat monitoring application, an average or maximum of the reported humidity values may be sufficient for the observer. For this purpose, a data aggregation tree, rooted at the observer, is constructed for *in-network* aggregation. Data aggregation reduces the communication overhead in the network, thus saving the sensor scarce energy resources. In addition, aggregation reduces channel contention and packet collisions.

For large-scale networks, node clustering has been proposed for efficient organization of the sensor network topology [1], [2], [3], [4], and prolonging the network lifetime. In a clustered network, a 2-tier hierarchy is constructed where cluster heads form an overlay responsible for data forwarding, while other nodes (which we refer to as "regular nodes") only report their data to their heads.[1] Energy savings are achieved by: (1) periodically re-clustering the network to select more energy-abundant cluster heads to participate in routing; and (2) allowing nodes at the lower tier to sleep most of the time since they do not participate in the routing infrastructure.

In this paper, we investigate the integration of node clustering and tree-based aggregation and routing in a real sensor network setting. We quantify the impact of clustering and aggregation on the network lifetime and the number of successfully transmitted measurements. We experiment with source-driven applications, where nodes periodically send reports to a fixed observer. (We discuss the case of mobile observers in [5]). In particular, we consider an application that uses a data aggregation operator, such as average (AVG), maximum (MAX), sum (SUM), or count (COUNT). Prior to constructing the routing tree, the network is clustered to identify a set of cluster heads that have higher average residual energy than their peers. Only cluster heads proceed to discover the path to the root of the tree (the observer) by constructing a breadth-first spanning tree.

Constructing a spanning tree for data forwarding was proposed for multi-hop routing in TinyOS [6], [7]. We integrate *HEED* clustering [2] with data aggregation in the *Multi-HopRouter* [8] to implement integrated HEED (iHEED) in TinyOS. We select the HEED clustering protocol because it terminates in a constant number of iterations, and elects cluster heads that are well-distributed in the network field. HEED does not require special node capabilities, such as location-awareness, does not make assumptions about node distribution, and operates correctly when nodes are not synchronized. iHEED can serve both source-driven applications (where sensors periodically report their readings) and data-driven applications (where an observer queries the network). To the best of our knowledge, our work is one of the earliest *implementations and testbed measurements* of *clustering protocols* in sensor networks. Our experiments using iHEED show that clustering plays a dominant role in delaying the first node death, while aggregation plays a dominant role in delaying the last node death.

The remainder of this paper is organized as follows. Section II gives a brief description of the HEED clustering protocol [2]. Section III introduces our network model and research challenges. Section IV formulates an energy model based on our hardware platform. Section V discusses the iHEED system design details. Section VI gives empirical results of iHEED performance on sensor motes. Section VII briefly surveys related work. Finally, Section VIII concludes the paper and discusses plans for system extensions.

II. HEED Clustering

In this section, we briefly describe the Hybrid, Energy-Efficient, Distributed (HEED) clustering protocol [2]. HEED assumes that sensor nodes do not have any special capabilities,

[1] A hierarchy may contain more than two tiers by recursive clustering.

0-7803-9466-6/05/$20.00 ©2005 IEEE

such as being GPS-equipped, and that all nodes to be clustered are equally important. The goal of HEED is to prolong the network lifetime, where the network lifetime is defined as the time until the first (or last) node in the network depletes its energy. We consider more practical definitions in Section VI-B. To attain this goal, HEED uses a probabilistic approach to elect cluster heads with high residual energy (compared to regular nodes) in a constant number of iterations.

A node is mapped to exactly one cluster, and must be able to communicate with its cluster head via a single hop using an intra-cluster transmission range, R_c. R_c corresponds to a power level P_c. Inter-cluster routing uses a higher transmission range, R_t ($R_t > R_c$), corresponding to a power level P_t. Inter-cluster routing may either be pro-active (i.e., table-driven) or reactive (e.g., Directed Diffusion [9]) according to the type of the application. Inter-cluster routing on data aggregation trees is the primary focus of this work.

Cluster head selection is based on two parameters: A primary parameter (node residual energy) is used to select an initial set of cluster heads, and a secondary parameter is used to break ties. A tie occurs when two nodes within range R_c from each other announce their willingness to become cluster heads. We propose a technique for estimating residual energy during network operation in Section IV. The secondary parameter can be set to an estimate of the intra-cluster communication "cost," which is a function of cluster density or neighbor proximity.

A node initially sets its probability to become cluster head $CH_{prob} = C_{prob}\frac{E_{residual}}{E_{max}}$, where $E_{residual}$ is the estimated residual energy of the node, E_{max} is a reference maximum energy, and C_{prob} is a small constant fraction used to limit the number of initial cluster head announcements. CH_{prob} is not allowed to fall below a small probability, p_{min}, to ensure constant time termination. During each iteration, a node arbitrates among the cluster head announcements it has received to select the lowest cost cluster head. If it has not received any announcements, it elects itself to become a cluster head with probability CH_{prob}. If successful, it sends an announcement indicating its "willingness" to become cluster head. The node then doubles its probability CH_{prob}, waits for a short iteration interval t_c, and begins the next iteration. A node stops this process one iteration after its CH_{prob} reaches 1. If a node elects to become a cluster head, it raises its transmission power to P_t for inter-cluster communication.

We have shown in [2] that HEED terminates in $N_{iter} = O(1)$ iterations, where $N_{iter} \leq \lceil log_2 \frac{1}{p_{min}} \rceil + 1$. In addition, the clustered network remains connected under a certain density model when $R_t \geq 6R_c$ (this is a loose upper bound). We have also proven that the probability of two cluster heads lying within the cluster range R_c of each other is very small.

III. System model

A. Platform

The platform we use in this work is the Berkeley Mica2 and Mica2Dot sensor motes [10] running TinyOS [7]. The Mica2 mote has a 7.38 MHz Atmel processor, while the Mica2Dot has a 4 MHz Atmel microprocessor. Both types have 128 KB program memory, 4 KB RAM, and 512 KB non-volatile storage. The two types also have the same radio properties. The radio is a Chipcon SmartRF CC1000, with 916 MHz frequency, FSK modulation with data rate 38.4 kBaud (19.2 Kbps), Manchester encoding, and linear RSSI (received signal strength indicator). Output power is digitally programmable by setting the PA_POW register. A minimum setting of PA_POW = 0x02 corresponds to a power output of -20 dBm (10 μW), while the default value PA_POW=0x80 corresponds to a power output of 0 dBm (1 mW). In our experiments, we use the documented power consumption values from the Chipcon CC1000 data-sheet.

We have performed measurements in our lab that indicate that the Mica2Dot motes have smaller transmission ranges compared to Mica2s using the same transmission power levels. This is *despite* the fact that they both use the same radio model and antenna. During our experiments, Mica2 motes were much more predictable than Mica2Dot motes.

B. Application

We consider the class of applications that utilizes in-network data aggregation. An example application is radiation-level monitoring around a nuclear plant, where the maximum value is of particular interest for the safety of the plant and the surrounding environment. Several projects, such as the Habitat Monitoring on Great Duck Island [11], can utilize the approach proposed in this work for data aggregation.

We study network support for the basic data aggregation operators: AVG, SUM, MIN, MAX, and COUNT. TinyDB [12] can be directly used on top of our clustered multi-hop network to provide query processing capabilities and data aggregation to applications. To assess energy savings, we experiment with a scenario where sensor nodes periodically sense the medium and forward their readings towards an observer on an energy-aware data aggregation tree (details are given in Section V). Nodes along the path from the leaves to the root aggregate data by forwarding only two values: (1) the data value D, which is the sum in case of AVG and SUM operators, or the maximum (minimum) in case of MAX (or MIN) operator, and (2) the number of aggregated sensor readings N. An observer can thus compute AVG by dividing D by N, SUM/MAX/MIN by using D, and COUNT by using N.

C. Challenges

Since we assume that node distribution in the field is random, some nodes may be on "popular" routing paths and rapidly deplete their energy, leaving areas in the field unmonitored. Periodic node clustering based on residual energy significantly mitigates this problem by electing nodes with higher remaining energy to perform the more demanding job of cluster heads. A 2-tier architecture may also reduce interference and collisions if different channels (or CDMA codes) are used for intra-cluster and inter-cluster communication Thus, routing and topology management should be based on both

shortest distance (or channel losses as proposed in the ETX metric [13]), as well as remaining energy.

A second challenge for data aggregation applications is the integration of clusters with data aggregation trees without degrading path quality. We propose applying HEED clustering prior to constructing the aggregation tree, and using only cluster heads to construct the aggregation tree. This organization is demonstrated in Fig. 1. Regardless of how the nodes are distributed in the field, HEED cluster heads are well-distributed. This helps in maintaining high path quality at the inter-cluster level.

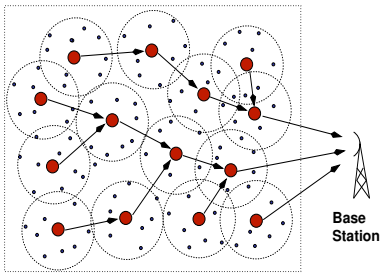


Fig. 1. A spanning tree of cluster heads rooted at the observer (base station)

A third challenge for energy-based topology management and routing protocols is the estimation of the remaining battery energy. Inspecting the analog-to-digital converter (ADC) for the battery voltage may not be useful due to its coarse granularity. In addition, the accuracy of the computed ADC result is not always guaranteed Therefore, we compute the remaining energy in the node by using a simple approach (described below) that exploits measurements in [14]. Our approach considers all sources of energy consumption, and defines a credit-point system (CREP) for updating the mote energy budget during on-line operation of the network. Our methodology is independent of the ADC hardware[2]. Observe that the CREP system is not a new model for measuring energy consumption of the sensor battery. It is a simple approach that compiles known physics formulae, and converts Joule consumption into point deduction for simpler computations. Some of the equations provided below are adopted in simulators, such as ns-2. The computations of the CREP system do not have to be highly accurate since a rough estimate of battery consumption will typically suffice to compute a reasonable cluster head probability.

IV. Dissipated Energy Estimation

Assume that a sensor uses a battery with maximum capacity A_b Amp-hr, and typical average voltage V_b. The maximum residual energy in the battery E_{max} is computed as:

$$E_{max} = V_b \times A_b \times 3.6 \times 10^3 Joule \quad (1)$$

We compute the dissipated energy for various sensor activities. In a sensor network, a node expends energy in: (1) processing, (2) sensing and actuation, (3) flash memory operations (read/write), and (4) communication (transmitting/receiving). According to the application, a node goes through a duty cycle when it performs some of the above activities and sleeps otherwise. For most applications, it is possible to accurately determine the active to sleep ratio for the sensor board since sensing is periodic. Processor, flash memory, and radio usage can also be estimated with some accuracy if independent of the data received. Inaccuracies in estimating the energy consumed in processing are negligible since it is well known that the energy consumed for communications is significantly higher

Let I_{pa} and I_{ps} denote the current drawn by the processor during the active and sleep periods, respectively. Let I_{mr}, I_{mw}, and I_{ms} denote the current drawn for memory read, write, and sleep, respectively. Let I_{sa} and I_{ss} denote the current drawn by the sensor board during active and sleep modes, respectively. Finally, let, I_{r_x}, $I_{t_x,R}$, and I_{cs} denote the current drawn by the radio for receive, transmit (to a range R), and sleep modes, respectively.

Consider the following active/sleep ratio for different components: processor $R_{pa} : R_{ps}$, flash memory $R_{mr} : R_{mw} : R_{ms}$, and sensor board $R_{sa} : R_{ss}$. Therefore, the effective current I_{eff} per unit time drawn from all components except the radio is computed as:

$$\begin{aligned} I_{eff} &= I_{pa}R_{pa} + I_{ps}R_{ps} + I_{sa}R_{sa} + I_{ss}R_{ss} \\ &+ I_{mr}R_{mr} + I_{mw}R_{mw} + I_{ms}R_{ms} \end{aligned}$$

The power consumed for all components other than the radio, P_o, is therefore computed as: $P_o = V_b \times I_{eff}$. Let $P_{t_x,R}$ be the power consumed for transmitting one packet of size k bytes (in Watt) with a range R, $P_{t_x,R}$ is computed as: $P_{t_x,R} = V_b \times I_{t_x,R} \times k \times 8$. The receive power consumption, P_{r_x}, is computed as: $P_{r_x} = V_b \times I_{r_x}$.

A. Credit-Point System (CREP)

Our credit-point system, CREP, assigns *points* to E_{max} instead of Joules. To be conservative, the points only represent a fraction of the computed E_{max} (say 90%). $E_{t_x,R} = P_{t_x,R} \times t_p$ points are deducted for each packet transmission, where t_p is the packet transmission time. Every period of time t_o, $E_o = P_o \times t_o$ points are deducted for energy consumption of components other than the radio, and $E_{r_x} = P_{r_x} \times t_{r_x}$ points are deducted for radio receive, where $t_{r_x} \leq t_o$. To avoid floating point computations and increase accuracy, points are integers with a finer granularity than the smallest granularity of energy consumption, as shown below.

Example: Consider Crossbow Mica2 nodes [10] with AA batteries. A conservative estimate of the AA battery capacity is 2.2 A-hr (an average estimate of AA capacity is about 2.4 A-hr). A Mica2 mote uses two AA batteries with effective average voltage V_b = 3V. Therefore, the total energy available for a Mica2 mote from 2 AA batteries, E_{max} = 2.2 × 3 × 3600 = 23760 J. We assume the processor's I_{pa} = 8 mA (1%), and I_{ps} = 15 μA (99%); the sensor's I_{sa} = $5mA$ (5%), and I_{ss} = 5 μA (95%); the flash memory's I_{mr} = 4 mA (0%), I_{mw} = 15 mA (0%), and I_{ms} = 2 μA (100%); the radio's $I_{t_x,R}$ = 16.8 mA, I_{r_x} = 10 mA (5%), and I_{cs} = 1 μA (95%). We assume

[2]A query power monitor was also proposed in the TinyDB query processor [12]. Its goal is different from our needs for the iHEED system.

that the sensors transmit at 0 dBM (1 mW). The time per bit transmission is 62.4 μsec as indicated in the measurement study in [14], and we assume t_o = 1 minute. We extract the parameters from the Mica2 data-sheet and the MPR/MIB user manual found at [10].

Using this information, E_o = (8 $\times$ 0.01 + 0.015 $\times$ 0.95 + 5 $\times$ 0.05 + 0.005 $\times$ 0.95 + 0.002 $\times$ 1) $\times$ 3 $\times$ 60 = 56 mJ. For a 30 byte-packet, the packet transmission time t_p = 0.0624 $\times$ 30 $\times$ 8 $\times$ = 14.97 msec. Thus, $E_{t_x,R}$ = $P_{t_x,R} \times t_p$ = 16.8 $\times$ 3 $\times$ 14.97 = 0.75 mJ per packet. E_{r_x} = 0.01 $\times$ 3 $\times$ 0.05 $\times$ 60 = 90 mJ, and E_{cs} (radio sleep) = $10^{-6}\times$ 3 $\times$ 0.95 $\times$ 60 = 171 μJ.

It is clear from the above calculations that the smallest energy (one packet transmission) is better expressed as multiple of 1 μJ (it can also be expressed as multiples of nJ or pJ to improve accuracy). Thus, the points in CREP are assigned as follows: The maximum credit limit (battery capacity) = $23,760 \times 10^6$ points, E_o = 56,000 points, $E_{t_x,R}$ = 750 points/packet, E_{r_x} = 90,000 points, and E_{cs} = 171 points.

In [5], we proved that the energy dissipated for worst case operation of HEED clustering is no larger than twice the energy dissipated for optimal clustering. We have verified this result empirically using uniform and non-uniform node distribution in the network field.

V. iHEED Implementation in TinyOS

In this section, we discuss the design details of the iHEED system.[3] iHEED extends the multi-hop router in [15] (initially proposed in [8]) by adding: (1) clustering logic that is executed prior to parent selection in the routing tree, and (2) a packet capture mechanism in the router for pushing data up the protocol stack to the data aggregation application, in case the node is a cluster head. The schematic design of the iHEED system is depicted in Fig. 2. iHEED adds about 420 lines of code to the TinyOS code. The packet size used is 29 bytes, which is the default in TinyOS. The main modules in the multi-hop router are:

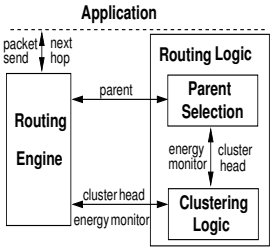


Fig. 2. Schematic diagram of the iHEED system

The Routing Engine: This is the main control unit in the iHEED router that is responsible for packet forwarding. The "Routing Logic" module is consulted for setting the next hop in the packet. For data aggregation, the "Routing Engine" module in a cluster head intercepts incoming packets from its cluster members or its descendants in the aggregation tree, and pushes them up the protocol stack to the application layer. This module is independent of the routing mechanism.

The Routing Logic: This module is responsible for providing a routing algorithm for constructing a connected graph. The "Routing Logic" consists of two main sub-modules: (1) the **Clustering Logic** which implements the clustering algorithm used to select a set of connected cluster heads. Aggregation tree construction follows network clustering, and considers only cluster heads in the routing infrastructure, and (2) the **Parent Selection** module which is responsible for estimating the link cost for each neighbor based on the "quality" of communications and its proximity to the base station. The quality of communication can be determined by considering data losses and link symmetry [8]. For example, a cluster head v having a neighbor u_1 that is 4 hops away from the base station, and another neighbor u_2 that is 6 hops away from the base station may prefer u_2 to u_1 as its parent if the loss rate for u_1 exceeds a specified minimum requirement (i.e., a metric like ETX [13] can be used). Each node maintains an internal estimate of the data sent and received from each of its neighbors to record the link quality to each of them.

1) **Cluster formation:** The initial proposal of the multi-hop router in [15] includes a timer (Timer1) used for sending out routing updates and triggering new routing computations. We augment the Routing Logic module with two additional timers for the clustering process: (1) clustering trigger timer (Timer2), and (2) clustering iteration timer (Timer3). When Timer2 expires, a node declares that it is not a cluster head (NON_CH) and it has no parent. At that point, we proceed to initialize a table of neighbors that are final cluster heads (FINAL_CH). This table is used to arbitrate among final cluster heads within the cluster range, after N_{iter} iterations have been executed. (The number of iterations, N_{iter}, was computed Section II.) Timer2 then triggers Timer3 to start iterating so that the node competes for cluster head candidacy. Whenever Timer3 expires, the steps discussed in Section II are followed in order to elect a cluster head or join a cluster.

If a node elects to become cluster head, a routing update message is forced by asynchronously triggering Timer1 to rapidly inform the neighbors. After the clustering process ends, routing update messages continue carrying information about a final cluster head to aid nodes that are newly deployed or have been sleeping for an extended period of time. After a node u elects a cluster head v, it invokes a method "joinClusterHead()" to use this head as its parent. This process is successful only if: (1) the link between u and v is symmetric using the intra-cluster range, and (2) v was able to find a path to the root, i.e., has determined its position in the aggregation tree. The pseudo-code for clustering logic initialization and timer actions can be found in [5].

2) **Clustering iteration interval t_c:** The clustering iteration interval t_c should allow neighboring nodes (within the cluster range) to exchange information about their status if they elect to become cluster heads. Three main parameters drive the choice of t_c: (1) the packet transmission time, t_p, (2)

[3]Our complete implementation can be downloaded from: http://www.cs.purdue.edu/homes/fahmy/

the number of neighbors n_g, and (3) the delays due to retransmissions, propagation, and queuing. Assuming that packets are lost with probability p, then $\lceil \frac{1}{1-p} \rceil$ transmissions will be required for successful packet transmission. The transmission interval should be multiplied by a constant c_q to account for propagation and queuing delays. Therefore, $t_c = n_g \times t_p \times \lceil \frac{1}{1-p} \rceil \times c_q$. For example, if t_p = 15 msec (as computed in the example of Section IV-A), n_g = 50, p = 0.15, and c_q = 2, then t_c should be set to 3 seconds.

3) **Triggering the clustering process:** It is difficult in practice for all nodes to start executing iHEED simultaneously because of clock drifts. Therefore, we use a simple approach to asynchronously trigger the clustering process in the network. A cluster head v whose Timer2 has expired *immediately* broadcasts a routing update packet to its cluster members and its neighbor cluster heads in the aggregation tree. The message contains information that v is not a cluster head anymore (NON_CH). Upon receiving this message, cluster members with v as their cluster head trigger their clustering process by re-initializing their Timer2. In addition, neighboring cluster heads *immediately* trigger their cluster members and neighbor cluster heads and so on. Hence, the clustering process diffuses throughout the entire network, though certain regions start slightly earlier than their neighboring regions. It is not essential that the entire network is simultaneously re-clustered: As long as every set of neighboring regions can start re-clustering within msec time difference, the network will still function correctly.

Observe also that triggering clustering does not require any additional overhead from a cluster head, except for a routing update message. For a realistic scenario where the battery lifetime is in the range of months and loads on cluster heads are balanced, the clustering process can be triggered at a coarse granularity, e.g., hours or days.

4) **Energy monitoring:** An energy monitor interface is added to the multi-hop router to manage the CREP system and supply information on the remaining energy to the clustering logic. As previously discussed, CREP points are deducted for data packet transmission, routing update packet transmission, radio receive, and energy consumption of other components. The conservative computation of points causes the CREP system to deplete its points while the battery is still operational. This is handled by the clustering logic, which uses a minimum probability for electing cluster heads when the remaining points are close or equal to zero. The pseudo-code for the energy monitor can be found in [5]. For radio receive, RECEIVE_COST points are deducted from the battery capacity. This cost may be fixed if the receive pattern of the radio is known. Otherwise, it will depend on the receive interval. For packet transmission, the cost is computed according to the power setting of the RF radio.

5) **Tree aggregation:** A cluster head aggregates the data packets received from its cluster members or tree descendants, and sends the aggregated value up to the root. To achieve this, we bind the packet interception at the Routing Engine with that of the application, thus pushing data up the protocol stack. The application manipulates data according to the aggregation operator, increments the count of data packets it has received within the epoch of time since its last send operation, and forwards the aggregate when the send timer (appTimer) expires. The pseudo-code for the application in the iHEED system can be found in [5].

6) **Detailed design:** Fig. 3 depicts a detailed description of the iHEED system. The figure is an extension of the *MultiHopRouter* in [15]. RoutingLogicM is the module that contains the clustering, and link estimation and parent selection (LEPS) algorithms. We show the new timers added for clustering, the use of the energy monitor interface, and the application interface with the Routing Engine to intercept packets coming to the node if it is an elected cluster head. The EnergyMonitor interface is also used by the Routing Engine to inform the application whether the battery is still operational. This information is only exploited by the application to stop data transmission. In future applications, the energy monitor can supply the application with valuable information about the battery status, which is used for notifying neighbors of possible death in the near future, and adjusting the rate or range of transmission.

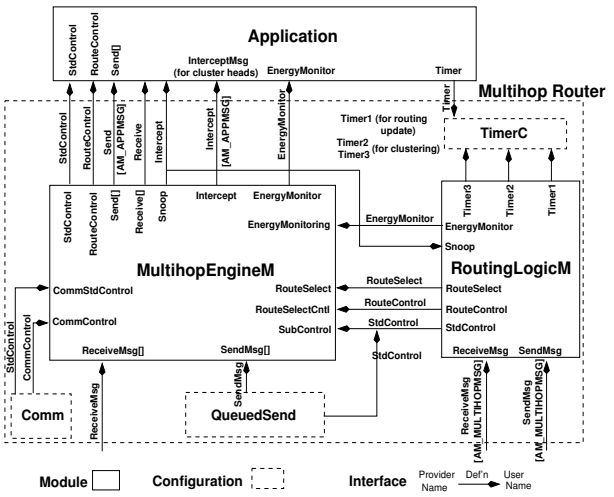


Fig. 3. Multi-Hop routing with clustering, aggregation, and energy control (an extended version of [15]). Arrows show interface provider/user relationships

The Comm interface, illustrated in Fig. 3, is responsible for packet capture and transmission. The Message ID is used to identify whether the packet is an application packet (AM_APPMSG) sent through the Routing Engine module, or a routing update packet (AM_MULTIHOPMSG) sent through the RoutingLogicM module. The QueuedSend interface is responsible for buffering the packets to be sent in sequence. Details of these interfaces are found in [6].

VI. Sensor Testbed Measurements

In this section, we evaluate the iHEED system on a testbed of Berkeley (Crossbow) Mica2 and Mica2Dot sensor motes. Our performance metrics are: (1) the sensor network lifetime,

(2) the number of successfully transmitted measurements, and (3) the overhead incurred by certain nodes (as described later).

A. Experimental Configuration

Our network setup is illustrated in Fig. 4. We conduct indoor experiments in a computer lab. We use 6 Mica2 and 4 Mica2Dot sensors distributed in an area of about 18 ft $\times$ 12 ft. The base station is also a Mica2 sensor attached to a MIB510 programming board, which is connected to the serial port (COM4) of a Pentium-III desktop running Windows XP. The capabilities of the motes are described in Section III-A. Let G_1 be the set of nodes $\{1,2,3\}$, G_2 be the set $\{4,5,6\}$, and G_3 be the set $\{7,8,9,10\}$. The nodes in each set are located within a circular area with 2 ft diameter. The average distances between the observer and G_1, G_2, and G_3 are 6 ft, 15 ft, and 4 ft, respectively. We place the set G_2 behind one of the lab partitions to create an obstacle and leave no line-of-sight between this group and the base station. This necessitates multi-hop communication between G_2 members and the base station through G_1 members.

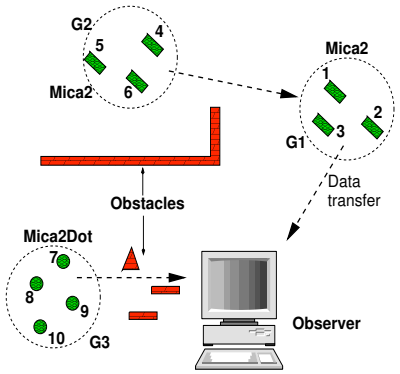


Fig. 4. The network testbed used in the experiments.

We use the following parameters: C_{prob} = 0.03, clustering iteration interval length (t_c) = 11 seconds, routing update frequency = 6/min, route recomputation frequency = 2/min, INTRA_CLUSTER_POWER = -20 dBm (PA_POW register = 0x02), and INTER_CLUSTER_POWER = -13 dBm (PA_POW register = 0x06). The choice of C_{prob} ensures a minimum of 7 iterations, which reduces the possibility of multiple cluster heads in the same region. t_c was selected such that node asynchrony does not affect the initial choice of cluster heads. The transmission ranges were selected such that three network regions are created and connected. The data rate is 0.5 pkts/sec. When the clustering process is in progress, each node aggregates its own data, and does not send it out until it finds a parent in the tree or a cluster head. This ensures that no data is lost during the clustering process. Using the clustering iteration interval and the initial clustering probability given above, the clustering process takes approximately one minute with a reasonably charged battery. In scenarios where the minimum CH_{prob} = 0.001, the clustering process takes close to 2 minutes. In real scenarios, clustering will be triggered every few hours, and data reporting will be triggered on the order of a few minutes. This implies that the clustering process is *within practical delay bounds for applications*. The time taken by the clustering process does not affect the application data flow since an old cluster head remains functional until a new cluster head takes over.

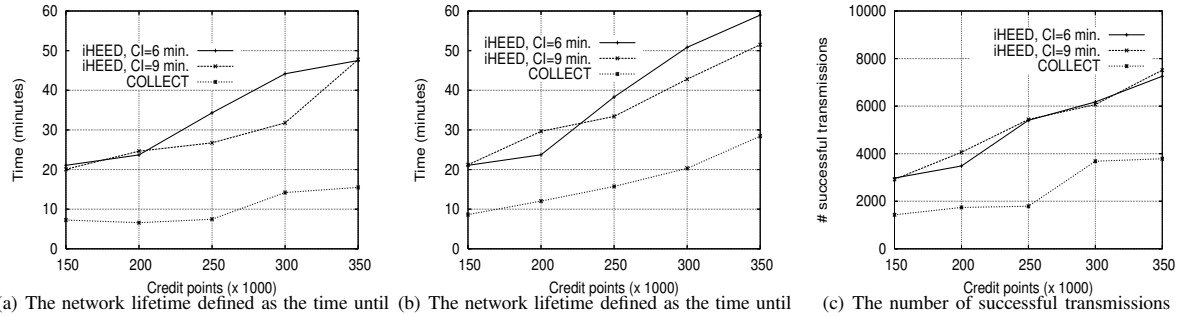
Our application uses the MAX operator. Our results are valid, however, for all similar operators, such as MIN and AVG. We conduct experiments using the CREP system presented in Section IV-A. We use values of drawn current from a recent measurement study on the Mica2 radio [14]. For PA_POW = 0x02, the drawn current = 5.3 mA, and for PA_POW = 0x06, the drawn current = 6.7 mA (approximately). The intra-cluster energy consumption $E_1 = 5.3 \times 3 \times 62.4 \times 29 \times 8 = 230\ \mu J$, and similarly the inter-cluster energy consumption E_2 = 291 μJ. Thus, we assign points in multiples of 1 μJ. We evaluate the iHEED system by comparing the performance of a data aggregation application using two different approaches. One approach uses multi-hop routing without clustering [15] (which we refer to as "COLLECT"), while the other uses a multi-hop routing data aggregation tree with node clustering (iHEED). Comparison with COLLECT was selected since it assesses the benefits of *both* clustering and aggregation, as explained below.

In our experiments, the initial battery capacity is assigned a fixed number of points (e.g., 200,000). For a clustered network, we deduct 230 points for each intra-cluster communication, and 291 points for each inter-cluster communication. We assume that nodes in the COLLECT approach use a transmission range similar to the inter-cluster range of the iHEED system.

This data aggregation application implies the following about energy consumption. First, processor energy consumption is almost similar in the two evaluated approaches. This is because the main processing overhead is in maintaining the routing table and preparing routing updates. The clustering process is infrequently invoked, and it involves exceedingly simple operations, except for the random number generation in case no cluster head announcements are heard. Second, the sensor radios are either: (1) in the receive mode if they are not transmitting; or (2) synchronized for sleep and wakeup, as in TinyDB [12] (see [5] for a discussion of the node duty cycle). In either case, the energy consumed in reception is independent of the number of received messages for both iHEED and COLLECT. Third, our application does not require flash memory operations. Therefore, the main parameter contributing to energy consumption in this application is the number of transmitted packets and the power level used for packet transmission.

B. Network Lifetime

The most common definition of network lifetime is the time until the first (or last) node in the network depletes its energy. In a multi-hop network, network connectivity is the primary determinant of network lifetime. That is, if in the set of nodes V, only a subset of the nodes $V' \in V$ can reach the observer in one hop (full-duplex), then the network practically



(a) The network lifetime defined as the time until first node death

(b) The network lifetime defined as the time until the network gets disconnected

(c) The number of successful transmissions

Fig. 5. Performance of iHEED vs. COLLECT

"dies" when nodes in V' deplete their energy because of disconnection from the observer. Therefore, network lifetime in multi-hop networks is defined as the time until the first (or last) node in V' depletes its energy. Using the above definition and our configuration illustrated in Fig. 4, the set $G_1 = \{1,2,3\}$ represents V' (subset of critical nodes). This is because G_2 members cannot reach the observer except through G_1.

Fig. 5(a) shows the network lifetime for the first node death definition for iHEED versus COLLECT. Results indicate that the first node death is delayed by a factor of up to 4 with the iHEED system. This significant improvement is attributed to the periodic re-clustering of the network that pushes each node in and out of the routing overlay to reduce its energy consumption. Fig. 5(b) shows the network lifetime for the second definition (last node death). The figure illustrates that the observer in the iHEED system remains connected to the network at least twice as long as the case with no clustering. As expected, node death in a clustered network speeds up after the first node death. The network lifetime is prolonged for the smaller clustering interval (CI), since more frequent clustering tends to distribute energy consumption more evenly among nodes. In summary, clustering plays a dominant role in delaying the first node death, while aggregation plays a dominant role in delaying the last node death.

C. Successful Transmissions

In this experiment, we measure the number of successful transmissions, since data loss may occur during the re-clustering operation. By successful data transmission, we mean that a sensor reading is carried all the way to the observer. Since the number of the aggregated sensor readings is updated in each packet on its way to the observer, the observer can easily compute the number of successful transmissions. This is a good metric since it implicitly accounts for losses due to the wireless medium.

Fig. 5(c) depicts the number of successful transmissions for both techniques. iHEED at least doubles the number of successful transmissions. This is a consequence to the prolonged network lifetime. The figure also illustrates that different clustering intervals do not result in significant differences in the number of received transmissions. This is rather surprising since the network lifetime is longer for the smaller CI. The longer lifetime advantage is offset by the fact that frequent clustering may result in data loss during tree construction. This effect is minimal in iHEED since a node does not send its data until a cluster head (or a parent) is found.

Although successful transmissions in iHEED significantly outnumber those in COLLECT due to the prolonged network lifetime, it is also important that the network using iHEED gives a consistent throughput comparable to that of COLLECT. We compute the network throughput by dividing the total number of packets successfully received at the root by the network lifetime. In [5], we show that the average throughput in iHEED is indeed comparable to that of COLLECT.

D. Overhead

The overhead in our application is defined as the energy consumed for routing updates, clustering, and packet forwarding towards the root. In iHEED, the clustering process only requires a few routing updates to carry cluster head announcements. Therefore, routing updates constitute the primary overhead in iHEED. In this experiment, we report the maximum overhead on any node in G_1 using iHEED, and compare it to the overhead of each node in G_1 using COLLECT. Fig. 6 shows that the maximum overhead in iHEED is less than one half of the average overhead in COLLECT. This is expected since packet forwarding in COLLECT consumes significant energy from nodes in the critical set. In the COLLECT experiments, most of the nodes that cannot directly reach the base station tend to use the same parent in the tree. This results in quickly depleting energy from this parent, which explains why the first node death in COLLECT is much faster than that in iHEED.

VII. Related Work

TinyOS [6], developed at UC Berkeley, was proposed as the operating system for small sensors. TinyOS provides network abstractions to facilitate communications. A number of approaches for multi-hop communications exist, including tree routing, ad-hoc routing, and broadcast and epidemic protocols [16]. TinyOS beaconing (AMROUTE) is the earliest tree routing proposal for TinyOS in which the root sends periodic

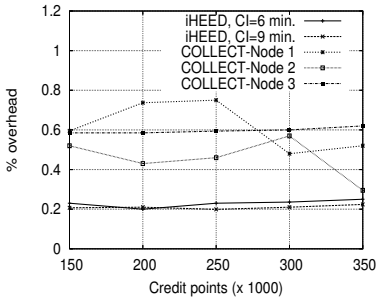


Fig. 6. Overhead of tree construction, tree maintenance, and clustering

beacons to construct the routing tree. Non-beaconing multi-hop routing was later implemented in TinyOS, e.g., mh6 [15] and its MultiHopRouter implementation. An evaluation of the impact of data aggregation on energy conservation in sensor networks was presented in [17]. TinyDB [12] and Cougar [18] were proposed for efficient database querying and aggregation in sensor networks. TinyDB focuses on the same class of operators (AVG, MAX, etc.) as in this study. TinyDiffusion [19] also exploits in-network data aggregation as necessary. Our approach supports both source-driven and data-driven applications, and thus can be used to construct an underlying structure for both TinyDB and TinyDiffusion. Support for other aggregation operators, such as median and histograms, was considered in [20]. In [21], [22], a schedule is computed for aggregating data to maximize the network lifetime. AIDA [23] studies application-independent data aggregation and proposes a mechanism for concatenating network units using an adaptive feedback scheme. For topology management techniques in ad-hoc networks, the reader is referred to [24].

VIII. Conclusions

We have presented iHEED: a system that integrates energy-efficient node clustering with data aggregation trees in sensor networks. iHEED prolongs the network lifetime and reduces contention on communication channels. We proposed a simple credit-point system (CREP) for tracking the energy dissipated in different sensor components. CREP assists in electing more energy-capable cluster heads. We implemented iHEED into TinyOS. Our experiments on a sensor network testbed demonstrate that by using clustering and data aggregation, the network lifetime is prolonged by a factor of 2 to 4, the number of successful transmissions is almost doubled, and the maximum overhead is reduced to less than half (for the studied application). Future extensions of this system will consider multiple cluster head overlays for reliability, efficient node duty cycles, and message authentication. We also plan to incorporate more operators, such as MEDIAN and VARIANCE.

Acknowledgments

We are extremely grateful to Saurabh Bagchi (Purdue University) for lending us sensor testbed nodes, and to Sam Madden (MIT), Fan Ye (IBM), and Victor Shnayder (Harvard University) for several useful discussions.

References

[1] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "Initializing Newly Deployed Ad-hoc and Sensor Networks," in *Proceedings of ACM MOBICOM*, September 2004.

[2] O. Younis and S. Fahmy, "Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach," in *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004, an extended version appeared in IEEE Transactions on Mobile Computing, 3(4), Oct-Dec 2004.

[3] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, October 2002.

[4] H. Chan and A. Perrig, "ACE: An Emergent Algorithm for Highly Uniform Cluster Formation," in *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, January 2004.

[5] O. Younis and S. Fahmy, "Energy-Efficient Routing and Data Aggregation in Sensor Networks: An Experimental Study," Purdue University, Tech. Rep. CSD-TR-04-031, December 2004.

[6] "TinyOS," http://www.tinyos.net, 2005.

[7] J. Hill, "System Architecture for Wireless Sensor Networks," Ph.D. dissertation, University of California at Berkeley, 2003.

[8] A. Woo, T. Tong, and D. Culler, "Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks," in *Proceedings of ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2003.

[9] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *Proceedings of ACM MOBICOM*, 2000.

[10] "CrossBow," http://www.xbow.com/, 2005.

[11] "Habitat Monitoring on Great Duck Island," http://www.greatduckisland.net/, 2005.

[12] S. Madden, "The Design and Evaluation of a Query Processing Architecture for Sensor Networks," Ph.D. dissertation, MIT, 2004.

[13] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *Proceedings of ACM MOBICOM*, September 2003.

[14] V. Shnayder, M. Hempstead, B.-R. C. G. Werner, and M. Welsh, "Simulating the Power Consumption of Large-Scale Sensor Network Applications," in *Proceedings of ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2004.

[15] "Multihop Routing for TinyOS," http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html, 2004.

[16] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The Emergence of Networking abstractions and Techniques in TinyOS," in *Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[17] L. Krishnamachari, D. Estrin, and S. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[18] Y. Yao and J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks," in *ACM SIGMOD record*, 2002.

[19] "Tiny Diffusion," http://www.isi.edu/scadds/software/, 2004.

[20] N. Shrivastava, C. Buragohian, A. Agrawal, and S. Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks," in *Proceedings of ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2004.

[21] K. Kalpakis, K. Dasgupta, and P. Namjoshi, "Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks," in *IEEE International Conference on Networking (ICN)*, 2002.

[22] K. Dasgupta, K. Kalpakis, and P. Namjoshi, "An Efficient Clustering-based Heuristic for Data Gathering and Aggregation in Sensor Networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2003.

[23] T. He, B. Blum, J. Stankovic, and T. Abdelzaher, "AIDA: Adaptive Application-Independent Data Aggregation in Wireless Sensor Networks," *ACM Transactions on Embedded Computing Systems*, vol. 3, May 2004.

[24] P. Santi, *"Topology Control in Ad Hoc and Sensor Networks"*. John Wiley and Sons, 2005.