2004

Energy-Efficient Routing and Data Aggregation in Sensor Networks: An Experimental Study

Ossama Younis

Sonia Fahmy
Purdue University, fahmy@cs.purdue.edu

Report Number:
04-031

ENERGY-EFFICIENT ROUTING AND DATA AGGREGATION IN SENSOR NETWORKS: AN EXPERIMENTAL STUDY

Ossama Younis
Sonia Fahmy

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #04-031
November 2004

Energy-Efficient Routing and Data Aggregation in Sensor Networks: An Experimental Study

Ossama Younis and Sonia Fahmy
Department of Computer Sciences, Purdue University
250 N. University Street, West Lafayette, IN 47907–2066, USA
e-mail: {oyounis,fahmy}@cs.purdue.edu

Abstract— **Several sensor network applications, such as environmental monitoring, require data aggregation to an observer (e.g., a base station). For this purpose, a data aggregation tree rooted at the observer is constructed in the network to reduce communication overhead and facilitate faster and more reliable results. Node clustering can be employed for this purpose, 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. In iHEED, sensor nodes are clustered prior to constructing the data aggregation tree. We consider simple data aggregation operators, such as AVG or MAX. We perform experiments on a sensor network testbed to quantify the benefits of integrating hierarchical routing with data aggregation. Our results indicate that, by using reduced intra-cluster transmission power and exploiting intra-cluster and inter-cluster data aggregation, network lifetime is prolonged by a factor of 2 to 4, and successful transmissions are almost doubled. The overhead of the clustering process is subsumed by tree construction and maintenance overhead.**

Index Terms— **sensor networks, implementation, clustering, energy efficiency**

I. Introduction

Networked embedded systems provide an important opportunity for supporting applications, such environmental monitoring or military field surveillance. In these applications, tiny sensors are deployed in vast fields to continuously report measured parameters, such as temperature, pressure, humidity, light, or chemical activity. Reports transmitted by these sensors are collected by observers, such as base stations, that are typically situated outside the network field. The unattended nature of these networks makes it impossible to re-charge node batteries. Therefore, the goal of most research in sensor networks has been to design energy-efficient protocols at all layers [1].

Several sensor network applications require an aggregated data value that is reported regularly. For example, in habitat monitoring where humidity is being measured, an average or maximum of the reported values may be sufficient at the observer. In military fields, where chemical activity or radiation is measured, the maximum value may be required to trigger an action or alert the troops. For this purpose, a data aggregation tree (e.g., a spanning tree) is constructed for in-network aggregation. The tree is rooted at the observer, which is the final destination of the aggregate report. Data aggregation reduces the communication overhead in the network, thus saving the sensor scarce energy resource. In addition, aggregation results in less channel contention and packet collisions. Consequently, data can reach its destination faster and more reliably.

For large-scale networks, node clustering is proposed for efficient organization of the sensor network topology. Clustering balances the traffic load and resource utilization and consequently prolongs the network lifetime [2], [3], [4], [5]. 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 periodically re-clustering the network to select more energy-abundant nodes to act as cluster heads, and form the network routing infrastructure. Rotating the role of cluster head among nodes also results in keeping the level of residual-energy uniform across the network, thus maintaining most of the nodes alive for extended periods of time, compared to a non-clustered network. In a multi-hop, non-clustered sensor networks, certain nodes may die very quickly because of their presence on "popular" paths, especially if the node distribution is non-uniform.

In this paper, we investigate the integration of node clustering and data aggregation trees in a real sensor network setting. We quantify the impact of using clustering on network lifetime and number of successfully transmitted measurements. We consider source-driven applications, where nodes periodically send reports to a fixed observer (we discuss the case of mobile observers in Section V). In particular, we consider an application that uses a data aggregation operator, such as average (AVG), maximum (MAX), minimum (MIN), sum (SUM), or count (COUNT). Prior to constructing the data aggregation tree, the network is clustered to identify a set of cluster heads that have higher average residual energy than their peers. Only cluster heads then proceed to discover the path to the root of the tree (the observer) by constructing a breadth-first spanning tree. Therefore, a cluster head acts as an aggregation point for its cluster members, as well as its child cluster heads in the data aggregation tree.

– This research has been sponsored in part by NSF grant ANI-0238294 (CAREER) and the Schlumberger Foundation technical merit award.

[1] A hierarchy can contain more than two tiers by recursively clustering the higher tier.

Constructing a spanning tree for data forwarding was proposed for multi-hop routing in TinyOS [6], [7], [8]. We use *HEED* clustering [2] in conjunction with data aggregation and integrate it with the *MultiHopRouter* [9] to implement a clustered, multi-hop router (iHEED) in TinyOS for data aggregation applications. We selected the HEED clustering protocol because it terminates in a constant number of iterations and selects cluster heads that are well-distributed in the network field [2]. This gives HEED an advantage over protocols whose termination is dependent on network diameter [10], [11], [12], [13] or dependent on the number of nodes [3]. HEED also does not require special node capabilities, such as location-awareness, does not assume specific node distribution, and works correctly when nodes are not synchronized. iHEED can serve both source-driven (where sensors are periodically reporting their readings) and data-driven (where an observer queries the network for some data of interest) applications. To the best of our knowledge, our work is one of the earliest implementations and testbed measurements of clustering protocols in sensor networks.

The contributions of this work can be summarized as follows:

- Identifying the practical challenges for building an energy-efficient routing infrastructure in the higher tier of a clustered network, using the residual energy parameter as the metric for cluster head selection.
- Designing and implementing the *iHEED* system, which integrates node clustering with data aggregation tree construction.
- Demonstrating iHEED advantages in terms of network lifetime and successful data transmissions using experiments on a testbed of sensor motes [14].
- Giving recommendations for the design of reliable routing systems in the presence of an intelligent MAC layer, significant packet losses, sleeping nodes, and in hostile environments.

The remainder of this paper is organized as follows. Section II gives a brief description of the HEED clustering protocol [2] and its properties. Section III introduces the hardware platform used, the class of data aggregation applications considered, the basic challenges in implementing a clustering protocol in TinyOS systems, and the iHEED system design details. Section IV gives empirical results of the iHEED implementation on sensor motes, and the effect of clustering on network performance. Section V discusses design and implementation aspects related to deploying clustering in large-scale networks. Section VI briefly surveys related work. Finally, Section VII concludes the paper and discusses plans for system extension.

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, such as being GPS-equipped, and that all n nodes to be clustered are equally important. The goal of HEED is to prolong the network lifetime.[2]. To attain this goal, HEED uses a probabilistic approach to elect cluster heads in a constant number of iterations. The elected cluster heads must have higher residual energy than regular nodes (not cluster heads) in the network. They must also be well-distributed in the network area to form a uniform routing infrastructure as demonstrated below.

HEED identifies a set of cluster heads which covers the entire sensor field. Each node v_i, where $1 \leq i \leq n$, is mapped to exactly one cluster c_j, where $1 \leq j \leq n_c$, and n_c is the number of clusters ($n_c \leq n$). A regular node 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. If the application is source-driven (nodes send periodic reports to an observer), then a proactive routing approach, such as DSDV can be used. If the application is data-driven (sensors only respond to queries), then reactive routing approaches, such as Directed Diffusion [15] can be used. 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 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. The HEED primary parameter is the node residual energy. We propose a technique for estimating residual energy during network operation in Section III-C. 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. For example, a cluster head with smaller degree may be favored to one with a larger degree to balance load.

Let the clustering process duration, T_{CP}, be the time interval taken by the clustering protocol to cluster the network. Let the *network operation interval*, T_{NO}, be the time between the end of a T_{CP} interval and the start of the subsequent T_{CP} interval. Clustering is triggered every $T_{CP} + T_{NO}$ seconds to select new cluster heads. A node initially sets its probability to become cluster head, CH_{prob}, as follows:

$$CH_{prob} = C_{prob} \times \frac{E_{residual}}{E_{max}} \quad (1)$$

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 then begins

[2]In [2], the network lifetime was defined as the time until the first (last) node in the network depletes its energy. We consider more practical definitions in Section IV-B

the next iteration. A node stops this process one iteration after its CH_{prob} reaches 1. Thus, if no announcements are received by a node until CH_{prob} reaches 1, a node will elect to become a cluster head and will exit the clustering process, raising its transmission power to R_t for inter-cluster communication.

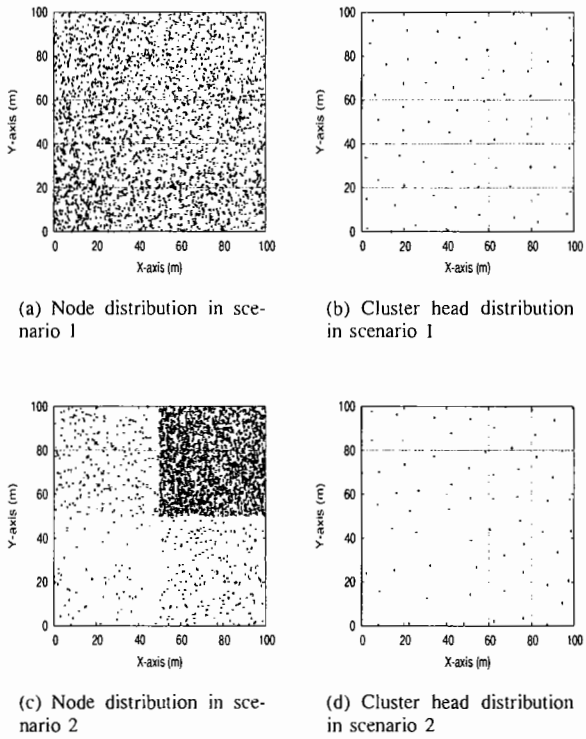
We have shown in [2] that HEED terminates in $N_{iter} = O(1)$ iterations, where $N_{iter} \leq \lceil log_2 \frac{1}{p_{min}} \rceil + 1$. In addition, the clustered network remains connected under a certain density model and when $R_t \geq 6R_c$.[3] We have also proven that the probability of having two cluster heads within cluster range R_c of each other is very small.

We now demonstrate the independence of the output cluster head distribution and the node distribution in the field. We experiment with two scenarios, where 3000 nodes are distributed in a 100 m × 100 m field with a cluster range R_c = 10 m. C_{prob} is set to 5%, E_{max} is set to 1 Joule, and each node starts with a random $E_{residual}$ that is Uniform[0,1]. In the first scenario, depicted in Fig. 1(a), nodes are uniformly distributed in the field. Fig. 1(b) shows that the elected cluster heads are well-distributed in the network field, with 3-4 nodes in each cell of 20 m × 20 m. This agrees with the cell occupancy analysis in [2]. In the second scenario, nodes are dispersed such that the bottom left, top left, bottom right, and top right quarters have 2%, 9%, 9%, and 80% of the nodes, respectively. Fig. 1(c) depicts the node distribution in this scenario. Fig. 1(d) shows the elected cluster head distribution, which is similar to that in the first scenario. This is because we set the HEED clustering parameters here to energy and an estimate of communication cost which is based on proximity (in addition to the cluster range). We do not use density or load as primary or secondary clustering parameters in this experiment. A cluster head v_1 in a dense area will expend more energy than another one v_2 in a less dense area because it has to serve more cluster members. However, this is compensated for by the fact that v_1 will elect as cluster head less frequently than v_2 because of the abundance of v_1 neighbors which compete to carry out this role. Well-distributed cluster heads are very important for paths to the observer to be (almost) optimal. There may be a redundancy in the first or last hop since a node has to send and receive through its cluster head. Once a cluster head obtains the cluster data, it forwards the aggregate data towards the observer, similar to typical multi-hop routing using the same transmission range. We assume that the observer is situated outside the network area.

III. iHEED Implementation in TinyOS

In this section, we discuss the iHEED system in detail. First, we describe the class of applications studied in this paper. Although most of our work is useful for configuring any energy-efficient sensor network architecture, we only consider data aggregation applications in this paper. Second, we discuss the challenges of energy-efficient clustering in sensor networks that affect our design and methodology. Finally, we describe the hardware platform of motes that we used in this work and our system design details.

[3]This is a loose upper bound.



(a) Node distribution in scenario 1

(b) Cluster head distribution in scenario 1

(c) Node distribution in scenario 2

(d) Cluster head distribution in scenario 2

Fig. 1. Cluster head distribution in two scenarios: (1) uniform node distribution, and (2) non-uniform node distribution

A. Application

We consider a 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. Another example is humidity measurement in a field (e.g., for habitat monitoring or cultivation). In addition, the average temperature is of interest for studying the best conditions for raising cattle or crop productivity. Several projects, such as the ZebraNet wildlife tracker [16] or the Habitat Monitoring on Great Duck Island [17], 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 [18] 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 send out their readings. These readings are forwarded towards an observer (base station) on an energy-aware data aggregation tree (details are given in Section III-E). 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 (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. In future studies, we

plan to incorporate more operators, such as MEDIAN and VARIANCE, in addition to multi-dimensional data. An initial study of efficient aggregation for the MEDIAN operator can be found in [19].

B. Challenges

From the network perspective, energy scarcity is the primary challenge that drives protocol design. To tackle this challenge, topology management becomes key for modifying the routing infrastructure in order to keep most of nodes operational as long as possible. An empirical study in [4] showed that periodic node clustering can prolong the network lifetime by a factor of four (for a specific radio model) compared to direct communication with the base station. Since nodes are randomly deployed in the field, their distribution on the ground can be arbitrary. Thus, some nodes may be on "popular" paths and rapidly deplete their energy, leaving areas in the field unmonitored. Periodic node clustering based on residual energy reduces this effect significantly by electing nodes with higher remaining energy to perform the more demanding job of cluster heads, leaving lower-energy nodes to only perform sensing. Clustering also enables nodes to communicate with smaller power ranges at the intra-cluster level for more energy savings. A 2-tier architecture also reduces interference and collisions if different channels (or CDMA codes) are used for intra-cluster and inter-cluster communication. Thus, routing should be based on both shortest distance (number of hops), 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. Therefore, a cluster head collects and aggregates data from its cluster members, as well as from its descendants in the aggregation tree. This organization is demonstrated in Fig. 2. The benefit of using HEED as the underlying cluster structure is three-fold: (1) whenever clustering is triggered, cluster heads are elected that have higher residual energy, (2) regardless of how the nodes are distributed in the field, HEED generates a well-distributed set of cluster heads as shown in Section II. This helps in maintaining high path quality at the inter-cluster level, and (3) cluster heads act as initial points of data aggregation, thus reducing the scale of communications.

A third challenge for energy-based topology management and routing protocols is the estimation of the remaining battery energy. One possibility is to inspect the analog-to-digital converter (ADC) for the battery voltage. This may not be useful because of the coarse granularity of the ADC result. In addition, the accuracy of the computed ADC result is not always guaranteed, which means that an erroneous value may lead to an inefficient choice of cluster head. Therefore, we compute the remaining energy in the node by using a simple approach described below. We also exploit the measurements provided in [20]. We consider all sources of energy consumption, and propose a credit-point system (CREP) for manipulating the mote energy budget during on-line operation

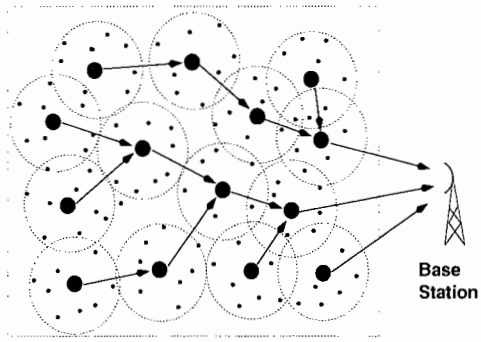


Fig. 2. A spanning tree of cluster heads rooted at the base station

of the network.[4] In Section III-C below, we give a general analytic model for CREP computations. Later, we will use this model for energy monitoring in our Mica2+Mica2dot (motes from Crossbow [14]) testbed. Note that we are *not* trying to propose an accurate CREP system. We are merely computing a rough estimate of the the remaining energy that can be computed using a unified methodology across all motes. This methodology is independent of the ADC hardware.

C. Dissipated Energy Estimation

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

$$E_{max} = V_b \times A_b \times 3.6 \times 10^3 Joule \quad (2)$$

We compute the dissipated energy for various sensor activities. In a sensor network, a node expends energy in four main activities: (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. Flash memory usage can also be estimated with some accuracy if it is independent of the data received. Processor active to idle period ratio can also be estimated accurately if it is independent of received data, or can be based on an estimate of the maximum amount of received data otherwise. This inaccuracy in estimating the processor consumed energy is negligible since it is well known that the energy consumed for transmitting/receiving data is relatively higher than processing energy [21]. The communication active and sleep periods can be estimated if the transmission pattern is fixed. However, it is usually difficult to estimate the amount of received packets during active periods, especially that the radio remains listening for extended periods of time if it is part of the routing infrastructure.

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},

[4] A query power monitor was also proposed in the TinyDB query processor [18]. Its purpose and approach are different from our needs for the iHEED system.

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_{rx}, I_{tx}, and I_{cs} denote the current drawn by the radio for receive, transmit, and sleep modes, respectively.

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

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

Consider a time period t_o. The energy consumed for all components other than the radio, E_o, can be computed as follows:

$$E_o = V_b \times I_{eff} \times t_o \tag{3}$$

Let E_{tx} be the energy consumed for transmitting one packet of size k bytes (in Joule), and let t_b be the bit transmission time. E_{tx} can be computed as follows:

$$E_{tx} = V_b \times I_{tx} \times t_b \times k \times 8 \tag{4}$$

Similarly, the energy consumed for receiving one packet (in Joule), E_{rx} can be computed as:

$$E_{rx} = V_b \times I_{rx} \times t_b \times k \times 8 \tag{5}$$

If we assume that the radio is always in the receive mode while it is not transmitting, then E_{rx} should be computed in a similar manner to E_o. That is, $E_{rx} = V_b \times I_{rx} \times (t_o - t_{tx})$, where t_{tx} is the total time during which the radio was in the transmission mode. t_{tx} can be computed by multiplying the number of packets transmitted during a period t_o by the time taken for one packet transmission.

Our credit-point system, CREP, assigns points to E_{max} instead of Joule. To be conservative, these points should be a fraction of the computed E_{max} (say 90%). E_{tx} points are deducted for each packet transmission operation, assuming a fixed packet length. Every period of time t_o, E_o points are deducted for energy consumption of components other than the radio, and E_{rx} points are deducted for radio receive. To avoid floating point computations and increase accuracy, points are integers with a finer granularity than the smallest granularity of energy consumption. For example, if the smallest granularity of the computed energy consumption values is in mJ, then the points are given in multiples of μJ.

Example: Consider a Crossbow Mica2 nodes [14] with AA batteries. A conservative estimate of the AA battery capacity is 2.2 mA-hr (an average estimate of AA capacity is about 2.4 mA-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. The current drawn by all components during active/sleep modes are listed in Table I. We assume that the sensors transmit at 0 dBM (1 mW). The table also gives the percentage of time each component is in active and sleep modes, estimated according to typical data values. The time per bit transmission is 62.4 μsec as indicated in the measurement study conducted in [20], and we assume t_o = 1 minute. We extract the rest of the parameters from the Mica2 data-sheet and the MPR/MIB user manual found at [14].

TABLE I

MICA2 MOTE CURRENT CONSUMPTION (ACCORDING TO DATA SHEET) AND PERCENTAGE OF TIME FOR ACTIVE AND SLEEP MODE (ACCORDING TO A HYPOTHETICAL APPLICATION)

Component/Mode	Current	Percentage
Processor (active) I_{pa}	8 mA	1
Processor (sleep) I_{ps}	15 μA	99
Sensor (active) I_{sa}	5mA	5
Sensor (sleep) I_{ss}	5 μA	95
Flash memory (read) I_{mr}	4 mA	0
Flash memory (write) I_{mw}	15 mA	0
Flash memory (sleep) I_{ms}	2 μA	100
Radio (transmit) I_{tx}	16.8 mA	-
Radio (receive) I_{rx}	10 mA	-
Radio (sleep) I_{cs}	1 μA	0

Using the information in Table I, E_o = (8 × 0.01 + 0.015 × 0.95 + 5 × 0.05 + 0.005 × 0.95 + 0.002 × 1) × 3 × 60 = 56 mJ. Now assume that during a t_o period of 1 minute, the sensor transmitted 20 packets of 30 bytes length each. Therefore, E_{tx} = 16.8 × 3 × 0.0624 × 30 × 8 = 0.75 mJ per packet. The total transmission time within the t_o interval, t_{tx} = 0.0624 × 30 × 8 × 20 = 300 msec. Therefore, E_{rx} = 0.01 × 3 × (60-0.3) = 1.8 J.

It is clear from the above calculations that the least energy (one packet transmission) is better expressed as multiple of 1 μJ. Thus, the points in CREP can be assigned as follows: The maximum credit limit (battery capacity) = 23760×10^6 points, E_o = 56000 points, E_{tx} = 750 points, and E_{rx} = 1.8×10^6 points.

D. Platform

The hardware platform we use in this work is the Berkeley Mica2 and Mica2Dot sensor motes [14] that run TinyOS [6], [8]. 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 also carried out some experiments in our lab to determine the correspondence between transmitted power (the value of PA_POW) and the transmission range. The results are shown in Table II. These measurements were taken when a clear line-of-sight was there between the sender and receiver motes. The results are usually specific to the environment where they are carried out. Therefore, we do not recommend these results

TABLE II

TRANSMISSION POWER LEVELS AND THEIR CORRESPONDING INDOOR MEASURED RANGES FOR CHIPCON RF CC1000 RADIO ON MICA2 AND MICA2DOT MOTES

Mote	PA_POW	Output power (μW)	Range (ft)
Mica2	0x01	-	≤ 0.5
	0x02	10	≤ 2
	0x05	32-40	≤ 6
	0x06	50	$\leq 8-9$
	0x0A	100	≤ 20
Mica2Dot	0x01	-	≤ 0.5
	0x02	10	$\leq 1-2$
	0x05	32-40	$\leq 3-4$
	0x06	50	≤ 6
	0x0A	100	≤ 10

to be used directly. In addition, the values when PA_POW=1 were not documented. That is why we just report the range results without specifying any corresponding consumed power value. From the table, it appears that the Mica2Dot mote covers smaller transmission ranges compared to those of Mica2s using the same transmission power levels. This is *despite* the fact that they both use the same radio model and antenna. The smaller range may be due to the inability of the Mica2Dot radio chip to draw as much current as the Mica2 radio. During our experiments, Mica2Dot motes sometimes exhibited unpredictable behavior, especially with respect to the transmission range capabilities. Mica2 motes have exhibited much more consistent behavior.

E. System Design

In this section, we discuss the design details of the iHEED system. iHEED extends the multi-hop router implementation and functionality in [22] (initially proposed in [9]) 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. 3. The main modules in the multi-hop router are:

The Routing Engine: This module is the main control unit in the iHEED router. It is responsible for examining whether the packet should be forwarded on to the parent of the tree, or pushed up the protocol stack. For setting the next hop in the packet, the "Routing Logic" module should be consulted. 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. The application can thus manipulate this data according to its own logic. This module is independent of the routing mechanism, and is responsible for sending out packets coming from the application layer.

The Routing Logic: This module is responsible for providing a routing algorithm for data forwarding. It is therefore responsible for structuring the network into a connected graph, maintaining information about neighbor nodes in a neighbor table, and sending routing update messages for tree construction and maintenance. 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. Thus, a cluster head can determine the "best" parent in the aggregation tree. The quality of communication can be determined by considering data losses and link symmetry [9]. 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. 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.

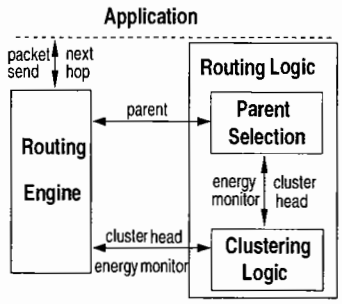


Fig. 3. Schematic diagram of the iHEED system

We now describe the details of incorporating HEED into the routing infrastructure, the process of energy monitoring, and the methodology for tree aggregation.

1) **Cluster formation:** The initial proposal of the multi-hop router in [22] 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 to serve the clustering process: (1) clustering trigger timer (Timer2), which is triggered every T_{CP}+ T_{NO} interval (defined in Section II), and (2) clustering iteration timer (Timer3), which is triggered every few seconds to start the next iteration of the clustering process. The number of iterations, N_{iter}, was computed Section II. 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. 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.

During the clustering process, information about node candidacy for becoming a cluster head is embedded within a routing update message, along with the secondary cost used by the HEED algorithm. If the node elects to become cluster head, a routing update message is forced out 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 is included in Fig. 4. For brevity, we do not detail the entire pseudo-code of the clustering process that is executed in Timer3.Expire() since it can be found in [2].

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) 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 III-C), n_g = 50, p = 0.15, and c_q = 6, then t_c should be set to 9 seconds.

3) **Triggering the clustering process:** The clustering process is triggered by timer expiration as described above. For efficient clustering, nodes should start the clustering process simultaneously. This is difficult in practice because clock drift causes the network nodes to be unsynchronized. 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. This is shown in the *RoutingLogic.Receive(pkt)* method in Fig. 4. In addition, neighboring cluster heads *immediately* trigger their cluster members and neighbor cluster heads and so on. Hence, the clustering process diffuses through the entire network, though certain regions start slightly earlier than their neighboring regions. A node whose Timer2 expires before it receives a trigger from its cluster head starts the clustering process independently. If its cluster head Timer2 still does not expire before all the clustering iterations are executed, then the node will likely elect the same cluster head again, given that their link is still symmetric. In addition, a node within its clustering process ignores any received cluster trigger packets.

The fine granularity of packet transfer (msec per packet) and the coarse granularity of a clustering iteration (seconds) allows the network to rapidly converge to a stable state. For example, for a network of 100,000 nodes and average cluster size of 100 nodes, the worst case diameter is linear in the number of cluster heads, i.e., 100,000/100 = 1000 hops. For

Fig. 4. The iHEED system pseudo-code: Multi-hop routing

RoutingEngine.Send(pkt) (application packet)
1. IF (forwardBufferList is not full)
2. Send(pkt)
3. EnergyMonitor.reduceRemainingEnergy(SEND_OP)

RoutingEngine.Receive(pkt) (application packet)
1. IF (pkt.nextHopAddress = myLocalAddress)
2. signal packetIntercept(pkt)
3. Forward(pkt)
4. Else Snoop(pkt) // save information about sender
5. EnergyMonitor.reduceRemainingEnergy(RECEIVE_OP)

Forward()
1. IF (forwardBufferList is full) return FAIL
2. nextHop ← Consult RoutingLogic
3. Send(pkt)
4. IF (nextHop ≠ myLocalAddress)
5. EnergyMonitor.reduceRemainingEnergy(SEND_OP)

RoutingLogic.Initialize()
1. Initialize neighborTable
2. myParent ← NULL
3. inClusteringProcess ← NULL
4. Timer1 ← Timer(ROUTING_UPDATE_INTERVAL)
5. Timer2 ← Timer(CLUSTERING_UPDATE_INTERVAL)
// Timer1 expiration triggers routing updates

event **Timer2.Expire()** //(clustering triggered)
1. inClusteringProcess ← TRUE
2. myState ← NON_CLUSTERHEAD
3. myTentativeClusterHead, myFinalClusterHead ← NULL
4. Compute CH_prob as in Eq. 1
6. sendRouteUpdate()
7. setRFPower(INTRA_CLUSTER_POWER)
8. Timer3 ← Timer(ITERATION_INTERVAL)

event **Timer3.Expire()**
// Clustering logic goes here.
// Timer3 is re-triggered N_{iter}-1 times
// Clustering done: inClusteringProcess=FALSE,
// RFPower=INTER_CLUSTER_POWER

RoutingLogic.Receive(pkt) (routing update packet)
1. updateNeighborInformation(pkt.address)
2. IF (inClusteringProcess)
3. update myTentativeClusterHead and myFinalClusterHead
4. ELSE IF (pkt.address=myFinalClusterHead and pkt.state = NON_CLUSTER_HEAD)
5. signal Timer2
6. EnergyMonitor.reduceRemainingEnergy(RECEIVE_OP)

EnergyMonitor.reduceEnergy(deductionReason)
1. IF (deductionReason = RECEIVE_OP)
2. totalEnergy ← totalEnergy - RECEIVE_COST
3. ELSE IF (deductionReason = SEND_OP)
4. totalEnergy ← totalEnergy - energyCost(getRFPower())
// Other deduction reasons go here.

a transmission speed in the order of msec, the entire network can be triggered in the range of a few seconds. This is within the granularity of one or two clustering iterations. It is not important, however, that the entire network is simultaneously re-clustered. As long as every set of neighboring regions can start re-clustering within msec time difference (which corresponds to the granularity of one-hop inter-cluster packet transmission), the network can 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.

4) **Energy monitoring:** An energy monitor interface is added to the multi-hop router to manage the CREP system and provide information on the remaining energy to the clustering logic. The remaining energy is used as the primary parameter in probabilistic cluster head election as described in Section II. Points are deducted for data packet transmission, routing update packet transmission, radio receive, and energy consumption of other components. The conservative assignment and computations of points causes the CREP system to lose all points while the battery is still operational. This is handled well by the clustering logic, which uses a minimum probability for electing cluster heads when the remaining points are close or equal to zero.

Fig. 4 gives pseudo-code for the energy monitor. 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. Smaller current values drawn for the shorter clustering range result in lower consumed energy, and hence fewer deducted points.

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. Fig. 5 gives the pseudo-code for the application in the iHEED system. Data aggregation occurs when the *packetIntercept(pkt)* event is triggered.

Fig. 6 depicts a detailed description of the iHEED system. The figure is an extension of the *MultiHopRouter* in [22]. 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 currently not exploited in our application, except to stop data transmission. In future applications, the energy monitor can supply the application with valuable information about the battery status, which can be used for notifying neighbors of possible death in the near future, and adjusting the rate or range of transmission.

Fig. 5. The iHEED system pseudo-code: Application

```
Initialize()
1. T_A ← APPLICATION_TIMER_RATE
2. appTimer ← Timer(T_A)
3. collectedData ← 0
4. numCollectedPoints ← 0

Stop()
1. appTimer.Stop()
2. ApplicationControl.Stop()

event appTimer.Expire()
1. reading ← data from ADC (sensing)
2. IF (node is cluster head)
3.     collectedData ← collectedData + reading
4.     numCollectedPoints ← numCollectedPoints + 1
5. ELSE
6.     collectedData ← reading
7.     numCollectedPoints ← 1
8. sendData()
9. IF (node address ≠ BASE_STATION)
10.    collectedData ← 0
11.    numCollectedPoints ← 0

sendData()
1. nextHop ← multihopRouter.getParent()
2. Send Packet(collectedData, numCollectedPoints) to nextHop

event packetIntercept(pkt)
1. IF (node is cluster head)
2.     collectedData ← collectedData + pkt.collectedData
3.     numCollectedPoints ← numCollectedPoints +
                            pkt.numCollectedPoints
```

The Comm interface, illustrated in Fig. 6, 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 can be found in [23]. iHEED adds about 420 lines of code to the *MultiHopRouter* [22]. The packet size used is 29 bytes, which is the default in TinyOS.

IV. 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). We start by describing our experimental configuration. Then,

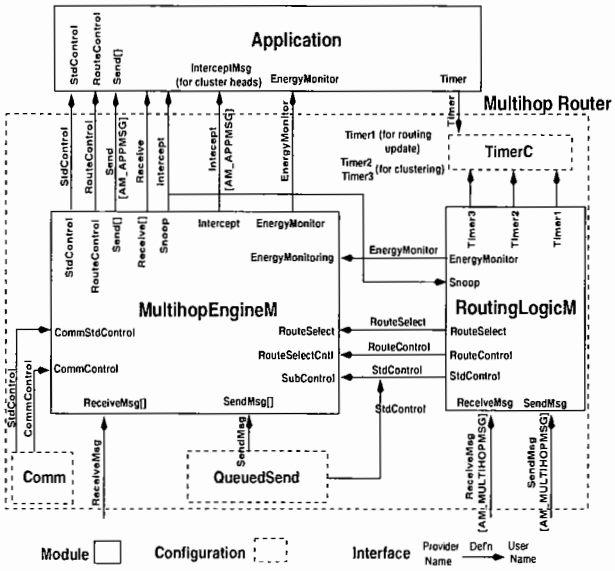


Fig. 6. Multi-Hop routing with clustering, aggregation, and energy control (an extended version of [22]). Arrows show interface provider/user relationships

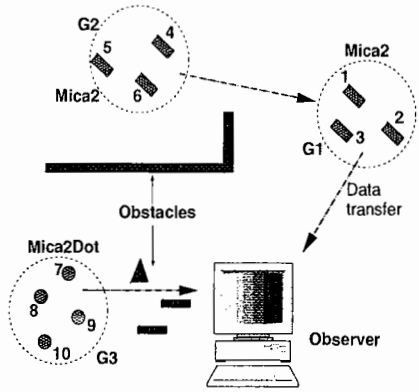


Fig. 7. The network testbed used in the experiments.

we evaluate each of the above parameters by running a set of experiments under different battery capacities. We summarize our findings at the end of the section.

A. Experimental Configuration

Our network setup is illustrated in Fig. 7. We conduct indoor experiments in a computer lab. We use 6 Mica2 and 4 Mica2Dot sensors distributed in an area of about 18 ft × 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 configuration of the motes is described in Section III-D, and the electric current consumption for different components is given in Table I. 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 of 1.5 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. G_1 and G_2 include Mica2 motes, while G_3 includes Mica2Dot motes. 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. There are also a few small obstacles between G_3 members and the base station. However, the pulses from that set can typically reach the base station because of their proximity.

We use the following parameters: C_{prob} = 0.03, clustering iteration interval length (t_c) = 11 seconds, routing update frequency = 6/min, route recalculation frequency = 2/min, INTRA_CLUSTER_POWER = -20 dBm (PA_POW register = 0x02), INTER_CLUSTER_POWER = -13 dBm (PA_POW register = 0x06). The data rate is 0.5 pkts/sec, i.e., the appTimer timer of the application expires every 2 seconds. 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. According to the speed of the clustering process and the rate of data reporting, the observer received aggregated reports from different regions in the network. 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 typically will be triggered every few hours 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.

We run experiments on the network using our simple CREP system presented in Section III-C. We use values of drawn current from a recent measurement study on the Mica2 radio [20]. 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 a multi-hop routing tree with no clustering (which we refer to as "COLLECT"), while the other uses a multi-hop routing data aggregation tree with node clustering (iHEED). In COLLECT, the latest implementation of *MultiHopRouter* for TinyOS [22] is used and augmented with an application that collects data at the base station. In the iHEED system, a cluster head v aggregates data packets from its cluster members and its descendants in the aggregation tree. When its appTimer expires, v sends one packet representing the aggregated data (including its own reading) to its parent in the tree.

For deducting points in this application, three issues must be taken into consideration. First, processor energy consumption can be assumed to be 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, we

assume that 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 [18] (see Section V for a discussion of the node duty cycle). In either case, the energy consumed in reception is independent of the number of received messages. Third, our application does not require flash memory operations (i.e., zero energy consumption for this component). Therefore, the main parameters contributing to energy consumption in this application are the number of transmitted packets and the power level used for packet transmission.

In our experiments, we assign the initial battery capacity as 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. Below, we present a performance comparison of the iHEED and COLLECT approaches. For iHEED, we perform our experiments at two different inter-clustering intervals (CIs), where CI is the interval between two successive Timer2 expirations (clustering trigger). We examine CI = 6 min, and CI = 9 min. Different clustering intervals have different effects on network lifetime and overhead as described below.

B. Network Lifetime

There are several possible definitions of network lifetime. The most common definition is the time until the first (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 "dies" when nodes in V' deplete their energy because of disconnection from the observer. Therefore, network lifetime in multi-hop networks can be defined as the time until the first (last) node in V' depletes its energy.

Using the above definition and our configuration illustrated in Fig. 7, the set $G_1 = \{1,2,3\}$ represents the V' subset of critical nodes. This is because the G_2 members cannot reach the observer except through G_1. We observed that G_3 sometimes also reach the observer through G_1 because of the small obstacles between its members and the observer. Hence, we measure the network lifetime in our application as the lifetime of nodes 1, 2, and 3.

Fig. 8 shows the network lifetime for the first node death definition for the application with and without clustering and data aggregation. 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. 9 shows the network lifetime for the second definition (last node dies). The figure illustrates that the observer in the iHEED system remains connected to the network at least twice as much when no clustering is applied. This order of magnitude difference in the two cases indicates an important fact: node death in a clustered network speeds up after the first node death. This is obvious since clustering balances energy consumption among nodes. We have noticed that in some cases, the entire set G_1 dies within a period of a few seconds.

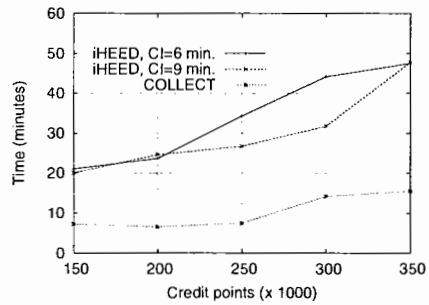


Fig. 8. The network lifetime defined as the time until first node death

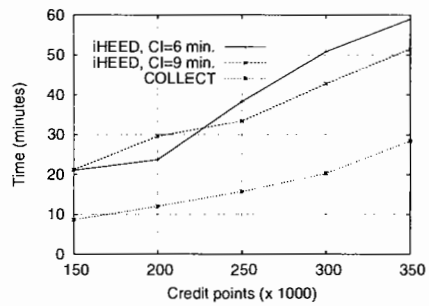


Fig. 9. The network lifetime defined as the time until the network gets disconnected

We can also see that the network lifetime is longer for the smaller CI interval. This is attributed to the fact that more frequent clustering tends to distribute energy consumption more evenly among nodes. This is further supported by the fact that our clustering protocol has low overhead.

C. Successful Transmissions

The goal of clustering is to prolong the network lifetime. Node clustering may, however, result in data loss during transient periods of network re-clustering. In this experiment, we measure the number of successful transmissions. Successful data transmission indicates that a sensor reading is carried to the root of the tree. In iHEED, the packet containing this reading does not appear at the root. Instead, the reading is aggregated at the parent (cluster head) of this node and is propagated all the way up to the root. Since the number of these aggregated sensor readings is reported and updated in each packet going to the root, then it is possible for the root to compute the number of successful transmissions.

Fig. 10 depicts the number of successful transmissions for the iHEED and COLLECT approaches. Using clustering and data aggregation improved the successful transmissions by at least a factor of 2. This is a direct consequence of 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 a bit surprising since the network lifetime is a bit longer for the smaller CI. The longer lifetime advantage is balanced, however, by the fact that frequent clustering may result in data losses during tree construction. This effect is minimal in our iHEED system because a node aggregates its data without sending that data until a cluster head is found. A cluster head also aggregates the data it is receiving until it finds a parent in the routing tree.

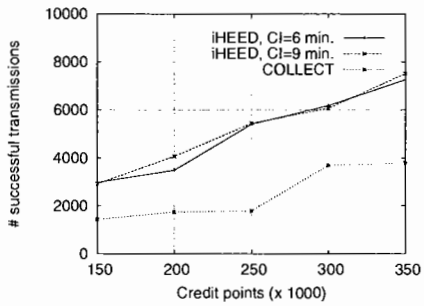


Fig. 10. The number of successful transmissions

Although the successful transmissions in iHEED significantly outnumber those in COLLECT due to the prolonged network lifetime, it is also important that the network using iHEED provide 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. Fig. 11 shows that the average throughput in iHEED is indeed comparable to that of COLLECT.

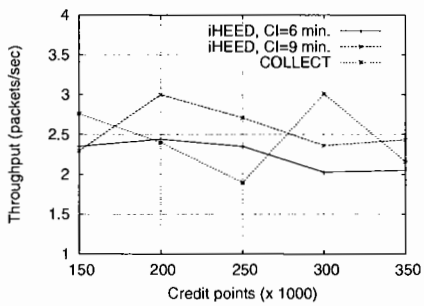


Fig. 11. Network throughput (packets/sec) measured at the observer

In our iHEED experiments, we noticed that data aggregation sometimes contributed more than clustering to the lifetime and throughput improvements. This is primarily attributed to the small-scale of our testbed. We expect the clustering effect to be dominant in large-scale networks.

D. Overhead

The overhead in our application is defined as the energy consumed for routing updates, clustering, and data packet forwarding toward the root of the aggregation tree. We compare the overhead imposed on the critical nodes {1,2,3} for the iHEED and COLLECT systems. Note that there are fewer data packet forwarding operations in iHEED: only cluster heads forward aggregate measurements. In addition, the clustering process only requires a few routing updates to carry cluster head announcements. Therefore, routing update is the main 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. 12 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.

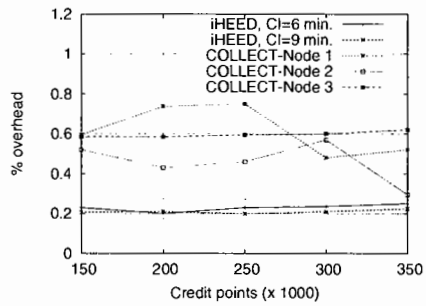


Fig. 12. Overhead of tree construction, tree maintenance, and clustering

E. Summary of Experimental Results

Our experiments indicate that integrating clustering with data aggregation results in significant performance improvements. The network lifetime is prolonged by a factor of up to 4 for the first node death definition, and a factor of 2 for the network disconnection definition. The number of successful transmissions is approximately doubled, while the throughput is similar to that of COLLECT. The average overhead imposed on the critical nodes in iHEED is about half of that of COLLECT.

Note that these results represent small-scale experiments in which the cluster head to regular node ratio is 1:2 to 1:3. We expect more significant performance improvements for larger cluster sizes.

V. Node Clustering in Large-Scale Networks

Our testbed experiments use only 11 nodes (including the base station), as other motes available to us (older and newer models) do not intercommunicate with Mica2 motes. In this section, we consider issues related to deploying node clustering in large-scale sensor networks. In particular, we discuss (1) network robustness and resilience in harsh environments, (2) how packet losses, mobility of observers, and the node duty cycle affect the iHEED system, (3) how clustering provides an opportunity for spatial-reuse, thus increasing the network capacity, and (4) scalability of routing in a clustered network.

We give recommendations for future extensions of iHEED to cope with different conditions and scenarios.

Fault-tolerance: iHEED operates best in environments where node failures are mainly due to energy depletion. In hostile environments, such as military fields, nodes may be destroyed due to other reasons, such as explosions. At the inter-cluster level, iHEED can easily handle such conditions by exploiting the routing updates for tree maintenance. At the intra-cluster level, however, a regular node needs to keep exchanging heartbeat messages with its cluster head to be able to determine whether it is alive. Upon detecting cluster head failure, several actions can be taken. The simplest action is that the node switches its state to cluster head and discovers its inter-cluster neighbors. This is not a favorable solution because it will be taken by all the nodes of the cluster of the failing head. Another solution is for a node to go through a cluster head discovery phase (as newly deployed nodes do) to join an operating cluster head. If the environment is very harsh, multiple cluster head overlays can be constructed to provide backup routing infrastructures, as proposed in [24].

Resilience to adversary attacks: Constructing a routing tree and clustering a network is vulnerable to many adversary attacks [25]. For tree construction, an adversary can send false routing updates with a strong signal to trick nodes into false shorter paths to the observer (the HELLO flood attack). The same attack can be applied to trick nodes into selecting a bogus cluster head that has a very low cost. iHEED is not susceptible to this attack since a node typically selects a parent based on link symmetry estimation, and not just the received signal strength. That is, assume that during clustering the adversary A uses a transmission range R_a, and a node v uses the clustering range R_c, where $R_c < R_a$. Through exchanged routing updates, v will realize that A can not hear v signals, and thus does not consider it to be a viable parent. The problem of false routing updates in general requires message authentication. Problems with compromised nodes are much harder to defeat. Most approaches to handle this problem propose in-network collaboration to detect the compromised nodes [26].

Effect of packet losses: The clustering protocol implemented in iHEED is resilient to packet losses during the clustering process. Packet losses, however can lead to less efficient choices of cluster heads. This effect is mitigated by the number of iterations that HEED clustering takes. For example, assume that each packet has a probability p of being lost. If a node v announces its willingness to become cluster head in iteration N_i, where $N_{iter} - N_i = 2$, then the probability that neighbor u will not hear this announcement is p^3. This is because a node re-sends this announcement in each remaining iteration, and packet losses during these iterations are typically independent.

Sleeping nodes: If the radio of a sensor node has to be on for extended periods of time for data relaying, this may result in significant energy dissipation, as we have discussed in Section III-C. To mitigate this problem, the radio of the sensor should go through a duty cycle that involves a power save mode (sleep mode). This was addressed in TinyDB [18]. where nodes in the tree synchronize their radio sleep and wakeup periods to reduce energy consumption. Synchronization accuracy may be critical, however, to avoid data losses. If nodes are not synchronized or data transmission is asynchronous, then nodes involved in the routing infrastructure have to be listening most of the time. Such energy dissipation is significantly scaled down when a clustered network is used. This is because nodes other than those in the cluster head overlay can go through their duty cycle independently and report their readings to their heads whenever they are awake. Since the number of regular nodes in the network is at least an order of magnitude larger than that of cluster heads in typical cases (reasonably dense network and reasonable power levels), this will lead to tremendous energy savings.

Mobile observers: Node mobility may increase the overhead of tree maintenance. Node clustering, however, is independent of the mobility of the *observer* (the root of the aggregation tree). In order not to spend significant network resources maintaining the tree to keep up with the pace of the observer, an alternative approach can be used to collect data. An application defines two routing modes in the network: (1) the multi-hop tree mode (which we described in this work) and (2) the reactive routing mode, which uses techniques such as dynamic source routing (DSR) or Directed Diffusion [15]. An observer can select a fixed cluster head in the network as the tree root for data collection as it moves in the field. Then, by using the reactive routing mode, the observer can query this root later for data update. This facilitates constructing only one tree for data collection, while being able to pull data from that tree anywhere in the field at any particular time. The SEAD protocol [27] proposed a more flexible approach for non-clustered topologies. In SEAD, the observer can join an already existing data dissemination tree through any node on the tree (which is called an access node). SEAD uses minimum energy paths to collect data from the delegated access node for further energy savings.

MAC layer and network capacity: Using shorter transmission ranges and enabling multi-hop communication can significantly increase the network capacity. However, interference remains problematic even in multi-hop networks when node density is high. One effective approach to avoid interference is to assign different CDMA codes or use different channels for transmission to maximize parallelism. This is difficult in our distributed ad-hoc environment, though. Fortunately, the clear distinction between inter-cluster and intra-cluster communication can alleviate the interference between the two communication types by assigning a different code for each. In case the cluster sizes are small, nodes can be synchronized with their cluster head (e.g., using the RBS protocol [28]) and a TDMA schedule can be used for intra-cluster communications.

Scalability versus complexity: Scalability is the primary goal for dense sensor network applications. Constructing a hierarchical structure, as in iHEED, helps reduce the communication overhead and save resources but at the expense of increasing processing and memory overhead. We conjecture (Section IV) that the clustering overhead in iHEED is insignificant in terms of both processing and memory. This is true for data aggregation applications, but may not be justified

for applications where nodes are cheap and easily replaceable, or applications where redundant deployment can effectively overcome any node failure.

VI. Related Work

TinyOS [23], [6], created at UC Berkeley, was proposed as the operating system for small sensors. TinyOS provides network abstractions to facilitate communications. For single hop communications, the active messages abstraction has been used for identifying message types and triggering actions based on this identifier [29]. For multi-hop communications, several approaches were proposed, such as tree routing, ad-hoc routing, and broadcast and epidemic protocols [7]. Tree routing is the most suitable for data aggregation. TinyOS beaconing (AMROUTE) is the earliest tree routing proposal for TinyOS in which the root sends periodical beacons to construct the routing tree. Non-beaconing multi-hop routing was later proposed and implemented in TinyOS, e.g., mh6 [22] and its MultiHopRouter implementation.

Data aggregation is crucial for several applications. An evaluation of the impact of data aggregation on energy conservation in sensor networks was presented in [30]. TinyDB [18] and Cougar [31] 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. (These operators were also the focus of other studies, such as [32], [33].) TinyDB assumes that nodes synchronize their sleep and wakeup periods for power management. TinyDiffusion [34] is the implementation of Directed Diffusion [15], a protocol that matches observer interests with sensor published data. TinyDiffusion also exploits in-network data aggregation as necessary. Both TinyDB and TinyDiffusion are designed to serve data-driven applications, where the observer queries the network for specific data of interest. Our approach provides networking support for 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, is considered in [19]. In [35], a schedule is computed for collecting and aggregating data in order to maximize the network lifetime. This work is extended in [36] to use clustering in order to scale down the overhead. AIDA [37] studies application-independent data aggregation and proposes a mechanism for concatenating network units using an adaptive feedback scheme. AIDA was evaluated on Mica motes and was demonstrated to reduce the end-to-end transmission delay by 80%. ESPDA [38] uses clustering to discard redundant data before being forwarded to the observer. If the data is encrypted, the cluster head just discards similar patterns. SEAD [27] proposes fixing aggregation points in the network. A mobile observer can access aggregated values by querying any nearby node (access point) on the aggregation tree. Aggregation timing policies are studied in [39] in order to evaluate their impact on the accuracy of aggregated data. The work in [40] proposes a mechanism for selecting the minimum number of aggregation points in the network in order to save energy consumption.

Several clustering protocols have been proposed for ad-hoc and sensor networks over the last few years, including [4], [2], [5], [10], [41]. Another recently proposed approach is [42], which selects a dominating set of nodes that covers the entire network area. The effect of different communication paradigms (single hop versus multi-hop) on the performance of clustering protocols is studied in [43]. Approaches that perform routing in clustered ad-hoc networks while supporting mobility can be found in [44], [13]. To the best of our knowledge, our work is one of the earliest implementations and testbed measurements of clustering protocols in sensor networks.

VII. Conclusion

In this paper, we presented iHEED, a system that integrates node clustering with data aggregation trees in sensor networks. Our approach prolongs the network lifetime, reduces contention on the communication channels, and rapidly aggregates and relays data to the observer. We proposed a simple credit-point system for tracking the energy dissipated in different components, such as processor, flash memory, sensor, and radio. By estimating the remaining energy in the sensor mote battery, more energy-capable cluster heads can be selected. We implemented iHEED and incorporated it 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.

This work can be extended in several directions. For robustness under unexpected failures, multiple cluster head overlays can be constructed, instead of only one. Route update messages should be authenticated to avoid adversary attacks in malicious environments. Detecting compromised nodes is also important in such environments. Techniques such as Statistical En-Route Filtering (SEF) [26] can be employed on the cluster head overlay to detect compromised nodes and exclude them from the data aggregation tree. In heterogeneous networks, where node capabilities are different, new cost definitions can result in deriving new parameters for the clustering process. For example, if sensors are used to collect images, then the sensors that have capabilities for image compression can be favored as cluster heads. Finally, larger scale experiments and comparisons with other routing approaches are planned for the near future.

Acknowledgments

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

References

[1] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, "Instrumenting the World with Wireless Sensor Networks," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001), Salt Lake City, Utah*, May 2001. [Online]. Available: http://citeseer.nj.nec.com/estrin01instrumenting.html

[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 appears in IEEE Transactions on Mobile Computing, vol. 3, issue 4, Oct-Dec, 2004.

[3] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "Initializing Newly Deployed Ad-hoc and Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, September 2004.

[4] 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.

[5] 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.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System Architecture Directions for Networked Sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.

[7] 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)*, April 2004.

[8] J. Hill, "System Architecture for Wireless Sensor Networks," in *Ph.D. Thesis*, May 2003.

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

[10] S. Banerjee and S. Khuller, "A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks," in *Proceedings of IEEE INFOCOM*, April 2001.

[11] S. Basagni, "Distributed Clustering Algorithm for Ad-hoc Networks," in *International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, 1999.

[12] C. R. Lin and M. Gerla, "Adaptive Clustering for Mobile Wireless Networks," in *IEEE J. Select. Areas Commun.*, September 1997.

[13] M. Gerla, T. J. Kwon, and G. Pei, "On Demand Routing in Large Ad Hoc Wireless Networks with Passive Clustering," in *Proceeding of WCNC*, 2000.

[14] "Crossbow," http://www.xbow.com/, 2004.

[15] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, 2000.

[16] "The ZebraNet Wildlife Tracker," http://www.princeton.edu/ mrm/zebranet.html, 2004.

[17] "Habitat monitoring on great duck island," http://www.greatduckisland.net/, 2004.

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

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

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

[21] K. Barr and K. Asanovic, "Energy Aware Lossless Data Compression," in *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys'03)*, May 2003.

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

[23] "TinyOS," http://www.tinyos.net, 2002.

[24] O. Younis, S. Fahmy, and P. Santi, "Robust Communications for Sensor Networks in Hostile Environments," in *the Twelfth International Workshop on Quality of Service (IWQoS'04)*, June 2004.

[25] C. Karlof and D. Wagner, "Secure Routing in wireless Sensor Networks: Attacks and Countermeasures," in *IEEE Workshop on Sensor Network Protocols and Applications*, May 2003.

[26] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-route Filtering of Injected False Data in Sensor Networks," in *Proceedings of IEEE INFOCOM*, March 2004.

[27] H. S. Kim, T. Abdelzaher, and W. H. Kwon, "Minimum-Energy Asynchronous Dissemination to Mobile Sinks in Wireless Sensor Networks," in *ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2003.

[28] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time synchronization Using Reference Broadcasts," in *Proceddings of OSDI*, 2002.

[29] P. Buonadonna, J. Hill, and D. Culler, "Active Message Communication for Tiny Networked Sensors," http://www.cs.berkeley.edu/ jhill/cs294-8/ammote.ps.

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

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

[32] J. Zhao, R. Govindan, and D. Estrin, "Computing Aggregates for Monitoring Wireless Sensor Networks," in *Proceedings of the First IEEE International Workshop on Sensor Networks and Applications (SNPA)*, 2003.

[33] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate Aggregation Techniques for Sensor Databases," in *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, 2004.

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

[35] 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.

[36] 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.

[37] 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.

[38] H. Cam, S. Ozdemir, and P. N. D. Muthuavinashiappan, "Espda: Energy-efficient and secure pattern-based data aggregation for wireless sensor networks," *IEEE Sensors*, vol. 2, October 2003.

[39] I. Solis and K. Obraczka, "The Impact of Timing in Data Aggregation for Sensor Networks," in *IEEE International Conference on Communications*, June 2004.

[40] J. Al-Karaki, R. Ul-Mustafa, and A. Kamal, "Data Aggregation in Wireless Sensor Networks - Exact and Approximate Algorithms," in *IEEE Workshop on High Performance Switching and Routing*, 2004.

[41] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh, "Max-Min D-Cluster Formation in Wireless Ad Hoc Networks," in *Proceedings of IEEE INFOCOM*, March 2000.

[42] F. Kuhn and R. Wattenhofer, "Constant-Time Distributed Dominating Set Approximation," in *ACM Symposium on Principles of Distributed Computing (PODC)*, July 2003.

[43] V. Mhatre and C. Rosenberg, "Design Guidelines for Wireless Sensor Networks Communication: Clustering and Aggregation," *Ad-hoc Networks Journal*, To appear.

[44] B. McDonald and T. Znati, "Design and Performance of a Distributed Dynamic Clustering Algorithm for Ad-Hoc Networks," in *Annual Simulation Symposium*, 2001.