Computing Aggregates for Monitoring Wireless Sensor Networks

Jerry Zhao and Ramesh Govindan
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
Email: {zhaoy,ramseh}@usc.edu

Deborah Estrin
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095
Email: destrin@cs.ucla.edu

***Abstract*— Wireless sensor networks involve very large numbers of small, low-power, wireless devices. Given their unattended nature, and their potential applications in harsh environments, we need a monitoring infrastructure that indicates system failures and resource depletion. In this paper, we briefly describe an architecture for sensor network monitoring, then focus on one aspect of this architecture: continuously computing aggregates (sum, average, count) of network properties (loss rates, energy-levels *etc.*, packet counts). Our contributions are two-fold. First, we propose a novel tree construction algorithm that enables energy-efficient computation of some classes of aggregates. Second, we show through actual implementation and experiments that wireless communication artifacts in even relatively benign environments can significantly impact the computation of these aggregate properties. In some cases, without careful attention to detail, the relative error in the computed aggregates can be as much as 50%. However, by carefully discarding links with heavy packet loss and asymmetry, we can improve accuracy by an order of magnitude.**

I. Introduction

Wireless sensor networks will consist of large numbers of small, battery-powered, wireless sensors. Deployed in an ad-hoc fashion, those sensors will coordinate to monitor physical environments at fine temporal and spatial scales [1]–[3]. Wireless sensor networks will be autonomously deployed in large numbers. Energy-efficiency is a key design criterion for these sensor networks.

A monitoring infrastructure will be a crucial component of a deployed sensor network. Such an infrastructure indicates node failures, resource depletion, and other abnormalities. Our first contribution is an *architecture* for sensor network monitoring infrastructures, one that consists of three classes of software. The first class of software *continuously* collects aggregates of network properties (we call them network *digests*) in the background. Triggered by sudden changes in these properties, *scans* can be invoked to provide global, yet aggregated, views of system state. Such views can indicate the location of performance problems or impending failure within the network. *Dumps* can then be used to collect detailed node state to debug the problem. These three pieces of software are invoked at different spatial and temporal scales, and will allow accurate, yet low-overhead sensor network monitoring.

Our second contribution is the design of protocols to continuously compute network digests. Abstractly, a digest is defined by an *digest function* $f(v_1, v_2, \cdots, v_n)$, where v_i is the value contributed by each node i. In this paper, we consider v_i's that represent some aspect of network operation: node energy level, degree of connectivity, volume of traffic seen *etc.* A key property of the class of aggregates we are interested in is *decomposability* [4], [5]. f is *decomposable* by a function g if it can be expressed as:

$$f(v_1, \cdots, v_n) = g(f(v_1, \cdots, v_k), f(v_{k+1}, \cdots, v_n))$$

Decomposable digest functions include *min*, *max*, *average*, and *count*. The median, for example, is *not* decomposable.

Aggregation has been discussed in different contexts such as large scale databases [6], active networks [7], and wireless sensor network applications [5], [8], [9]. However, computing digests for sensor networks poses unique design challenges. Digests are computed *continuously* and from the *entire network*. Furthermore, computing digests represents background activity and not the sensing task done by the applications (in contrast to queries that compute the average temperature of a region, for example). Finally, prior aggregation schemes have been designed to deliver aggregates on-demand to a small number of users outside the network; we argue that digests should be continuously distributed throughout the entire network. This will allow users low-latency access to digests from any node within the network. In addition, it may also enable applications to tailor their performance based on the values of digests (*e.g.*, shift to a different mode of operation when the average energy level falls below a certain threshold).

A. Our Approach

These observations lead to two key constraints in the design of protocols for digest computation. First, digest protocols must be *aggressively* energy-efficient, far more so than other components of the system. Second, because there isn't a natural initiator for a digest (*e.g.*, a user node) the routing structures for digest computations must be autonomously derived.

To achieve aggressive energy-efficiency, we propose to piggyback digest computation messages on neighbor-to-neighbor communication. We observe that many proposed sensor network protocols for medium access [10] and for topology control [11], [12] include periodic beaconing. Digests, being

small by definition, can easily be piggybacked on such communication. While not itself a new idea, this approach *seems almost necessary* to achieve very low energy expenditures for digest computation. This approach trades-off latency for energy savings. We quantify this trade-off in a later section.

We observe that some decomposable digest functions like *min* and *max* can be computed using a technique we call *digest diffusion*[1]. For example, suppose we are interested in computing a digest that represents the value of minimum energy at any node in the network; call this value E_{min}. Each node periodically broadcasts to its neighbors (*e.g.* by piggybacking on other messages) its own energy, as well as its current estimate of E_{min}. Each node also sets its estimate of E_{min} to the lower of its own energy-level and the lowest among the estimates heard from its neighbors. After a few iterations (intuitively, a number proportional to the network diameter), all nodes converge to the right E_{min}. In other words, E_{min} diffuses out to the network. (This is, of course, a simplified description. We describe our protocol more fully in Section IV).

Thus, digest diffusion can be used to evaluate one class of digests, and satisfy two important requirements we discussed above. First, every node ends up with an estimate of the digest. Second, these computations do not need to be explicitly initiated by some external action (*e.g.* by injecting a query into the system).

Not all digest functions can be computed using iterative diffusing computations. For example, the *average* and *count* functions, being non-idempotent, can be particularly sensitive to duplicates. Our simple diffusing computation can easily deliver duplicate data to a node. To compute this class of digests, though, we observe that computing a *min* or a *max* using an iterative diffusing computation *results in a tree that spans the entire network*. As we show in Section IV, we can use this tree to compute this class of digests, by propagating digest values to the root of the tree. Note that this tree is not constructed by user initiation, but as a by-product of computing a *min* or a *max* digest[2]

Finally, we note that some digests are particularly sensitive to packet loss. *Count* is an example of this. The packet loss on a wireless link can be significant is anecdotally well known. However, not much work has gone into quantifying the extent of loss, until recently. Morris *et al.* show the prevalence of links with heavy loss and asymmetry in 802.11 environments [13]. Our own experiments confirm this in Section V for a network of *motes* (the sensor node platform). Even in fairly benign environments, we observe widespread and time-varying occurrences of heavy link loss and asymmetry. We also demonstrate that simple implementations of the *count* digest can exhibit severe error in these environments. We then show that a careful implementation that selectively avoids links with heavy loss and asymmetry can improve the accuracy of *count* computation, sometimes by an order of magnitude.

[1]This is in contrast to *directed diffusion* [8]. a data-centric routing paradigm.

[2]The notion here is that the network will be continuously computing several kinds of digests, depending on the needs of the particular deployment. We think at least one of them will be a min or max digest, and that can form the basis for computing other digests.

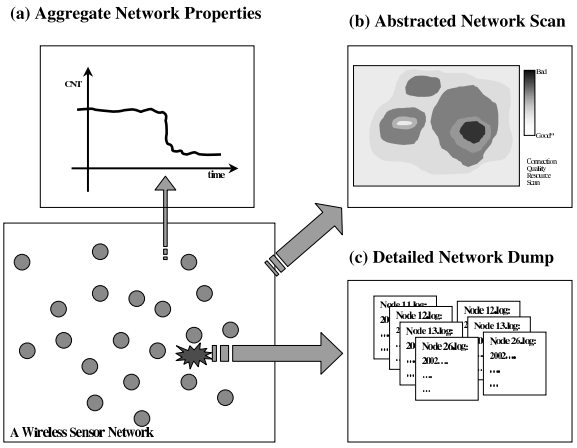


Fig. 1. Monitoring wireless sensor networks

To our knowledge, this paper is the first to articulate an architecture for sensor network monitoring. Our paper fleshes out a very practical implementation of one component of this architecture, discussing how real-world artifacts can seriously impact the performance of the monitoring system.

B. Paper Organization

The rest of the paper is structured as follows. The next two sections describe an infrastructure to monitor wireless sensor networks in details, and give a brief definition of aggregate network properties. Section IV describes our approach that enables energy-efficient computation of aggregate properties. Section V describes link quality estimation and rejection algorithms to reduce the negative impacts of packet loss on the performance of the computation process. The performance of our design is evaluated by implementation on a testbed and a simulator in Section VI and VII. We conclude our work with related work and discussion of strength and limitation of our approach.

II. Monitoring Wireless Sensor Networks: An Architecture

While the main focus of this paper is a specific set of diagnostic tools for sensor networks (digests), we describe in this section our vision for how these tools fit into a coherent architecture for monitoring sensor networks. This architecture, which is quite different from the classical SNMP [14] architectural model (centralized collection of per-device statistics), is motivated by the need for energy-efficient communication in sensor networks.

Our architecture is distinguished by three levels of monitoring, where each level consists of a class of tools. Each level is distinguished from the next in the spatial or temporal scale at which the corresponding tools are invoked. This is illustrated in Figure 1.

The first component consists of tools such as *dump*. Upon user's request, *dump* collects detailed node state or logs over

the network for diagnosis. For example, we could dump the raw temperature readings from some sensors to debug the collaborative event detection algorithm between nearby nodes. *Dump* can be implemented as an application upon directed diffusion [8]. Because the amount of data per node may be large, *dump* should be invoked only at small spatial scales (*i.e.*, from a few nodes), and only when there is a reasonable certainty of a problem at those nodes.

To guide system administrators to the location of problems, we envision the second class of tools that we call *scans*. Scans represented abstracted views of resource consumption throughout the entire network, or throughout a significant section of the network. Thus, this class of tools has a significantly greater spatial extent than dumps. One example of a scan is the *escan* [15]. To compute an *escan*, a special user-gateway node initiates collection of node state, for instance residual energy supply level, from every node in the system. Instead of delivering the raw data to user node, *escan* computation takes advantage of in-network aggregation. Residual energy level data from individual nodes are combined into more compact forms, if and only if those nodes are nearby and have similar energy level. By pushing the data processing into the network, *escan* constructs an approximate system-wide view of energy supply levels with much less communication cost compared to centralized collection. From such a global view, users are able to isolate those nodes upon which they can invoke tools such as *dump*.

Clearly, the energy cost of collecting an escan can be significant, and our third class of tools, *digests*, can help alert users to error conditions (partitions, node deaths) within the network. As we have described before, a digest is an aggregate of some network property. For example, the size of network i.e. the number of nodes, can indicate several system health conditions: Sudden drop in the network size can be taken as hint for massive node failure or network partitioning. In the paper, we show how to collect aggregates efficiently, accurately, and continuously. Digests, like escans, also span the entire network, or a large spatial extent. However, unlike escans, they are continuously computed. Digests are not intended to isolate network problems, merely to tell users when to invoke network-wide scans.

III. Definitions, Assumptions, and Models

We assume that the sensor network consists of n nodes deployed in an ad-hoc manner. Nodes have unique identifiers. Nodes may crash due to failures or resource depletion and new nodes may join the network. Nodes are static or move infrequently. Each node can communicate with its neighbors within certain range. Communication between nodes may be lost due to noise or collision. We do not assume a specific MAC or routing protocol, but do assume the radio capability to broadcast messages to neighbors.

Recall that a digest function is denoted by $f(v_1, v_2, \cdots, v_n)$, where v_i is the value contributed by sensor node i. Additionally, f is *decomposable* by a function g:

$$f(v_1, v_2, \cdots, v_n) = g(f(v_1, \cdots, v_k), f(v_{k+1}, \cdots, v_n))$$

A decomposable digest function is one in which the final result can be calculated from partial results. The values v may either be scalars or vectors. For example, to compute average residual energy supply level in the network, we can define $v =< s, c >$ and aggregate function

$$g_{AVG}(v_1, v_2) =< v_1.s + v_2.s, v_1.c + v_2.c >$$

where s and c are the sum of energy level and c is the node count, respectively. The average value is derived from the final result $v.s/v.c$.

The problem of digest computation is: Each node i provides a value v_i as its contribution to the digest function f, where v_i may change over time. The goal of the digest computation mechanism is for each node in the network to contain a continuous estimate for the current value of f. In this paper, we limit the digest functions we consider to V_{MAX}, V_{AVG}, V_{SUM} and V_{CNT} , which respectively denote the maximum, average, and sum of $v_1, v_2, \cdots v_n$, and number of the nodes in the network.

There is a specific rationale for our choice of digest functions, since these functions are qualitatively different from each other. Using terminology from [5], a digest function is *monotonic* if and only if, when two partial results $r_1$and r_2 are combined by a function $r = g(r_1, r_2)$, the result r satisfies $\forall i = 1, 2\ r \succeq r_i$ for an ordering relationship $\succeq$. It is *exemplary* if the final result can be determined from one single contribution value. In our set of digest functions, V_{MAX} is monotonic and exemplary, while V_{CNT} is monotonic but not exemplary, and V_{SUM} and V_{AVG} may not be monotonic (if negative values are allowed) and are certainly not exemplary. Finally, as we shall argue later, the loss sensitivity of V_{AVG} may be different from that of V_{MAX}.

IV. Computing Digests

In this section, we discuss techniques for computing digest functions for sensor network monitoring.

A naive, centralized, approach to compute digest functions is to have each node send its value to a designated head node H. H computes the final result from all the values received. This approach does not scale well with network size. First, there is possible message implosion at nodes near H. Second, it can incur heavy processing work load at H to aggregate values from all nodes. Third, H represents single point of failure.

Our approach leverages *in-network* aggregation. Each node computes a partial result of the digest function, and passes that result to other neighboring nodes (we describe the exact technique in the next two sections). For this, we leverage the fact that our digest functions are all decomposable. In-network aggregation has better energy-efficiency characteristics; communication overhead is less, and the computation is evenly distributed.

A standard way of computing these digest functions using in-network processing is to use a hierarchy and propagate the digest up to the root, computing partial values along the way. Such an approach is exemplified by the approach of Gupta

et al. [4], where node location is leveraged to construct a "Grid Box" hierarchy. However, their approach for computing aggregates requires leader election within grid boxes, and other maintenance overhead. One requirement for our monitoring application is that digest computation has to be aggressively energy-conserving. Another approach, with similar drawbacks from the perspective of monitoring, is the idea of recursive clustering elections [16], [17].

Instead of using more heavyweight hierarchy and clustering techniques, we use a two-pronged approach for computing digests.

- We note that some of our digests can be computed by a scheme we call *digest diffusion*.
- Digest diffusion implicitly builds a tree. We use this tree to compute digest functions by propagating partial results up the tree towards the root.

We now describe these in some more detail.

A. Digest Diffusion

We note that monotonic and exemplary digest functions can be computed efficiently by localized information exchanges between one-hop neighbors. We call this technique *digest diffusion*. We now describe digest diffusion for V_{MAX}. Initially, each node i sets its perceived maximum value $m_i = v_i$, source of maximum $s_i = i$, hop distance $h_i = 0$ and periodically sends a tuple $M = (m_{i,}, s_i, h_i)$ to its neighbors. Upon receiving a message (m_j, s_j, h_j) from neighboring node j with $m_j > m_i$, node i sets $m_i = m_j$, $s_i = s_j$, $h_i = h_j + 1$ and parent $p_i = j$. If $m_j = m_i$, it further checks if $s_j > s_i$, which guarantees strict monotonicity. Node i may switch its parent node from j to node k, when k provides the same maximum value but a shorter hop distance $h_k < h_j$. Gradually within $O(d/(1-p))$ steps (d is the diameter of the network, p is packet loss rate per link.), all nodes agree on a node s with the maximum value v.

This fusion based approach is simple but efficient. It is fully distributed and requires no base-station or user node to initiate the computation. The computation converges in a time proportional to the network diameter. It is energy-efficient and scales well with network size since the overhead at each node is constant over time. The information exchanged between neighbors is small and can easily be piggybacked[3] on other neighbor-to-neighbor communication (*e.g.*, beacons sent by MAC protocols or protocols for topology adaptation[4]).

B. Computing Other Digests

However, digest diffusion *cannot be used to compute non-exemplary digests*, such as V_{AVG}. One of the fundamental reasons is that when a node tries to aggregate the V_{AVG} partial results from its neighbors, it is difficult to determine if there are any overlaps between those results. For example, in Figure 4, node E tries to aggregate the partial results for V_{AVG} from C and D. however without explicit knowledge whether values from A, B have been accounted by C, D or both of them, it is impossible for E to aggregate correctly.

We note that digest diffusion implicitly constructs a tree whose root is the node that contributes to the value of the exemplary digest (*e.g.*, the node that has the maximum value in a V_{MAX} digest. Digest diffusion also computes a parent p_i for each node i (see Section IV-A). We call this tree the *digest tree*.

Other digest functions can be computed easily on this tree. For example, with the aggregation tree from V_{MAX} computation, it is straightforward to calculate compute V_{AVG}: node i periodically calculates a partial result from most recent reports from its children $c_1, c_2, \cdots, c_k$, for node count

$$n_i = \sum_{j=1}^{k} n_{c_j} + 1$$

and average value

$$a_i = \frac{\sum_{j=1}^{k} n_{c_j} \cdot a_{c_j} + v_i}{n_i}$$

It then sends out $< a_i, n_i >$ to its parent p_i along the tree. Hop by hop, the partial results are propagated up to the root, where the final result of V_{AVG} is calculated. It takes $O(d/(1-p))$ time to converge on the correct result, given the tree structure is stable. We may further reduce communication cost by *incrementally updating* the partial digests. Only those subtrees that have nodes whose values have changed beyond a certain threshold need to send their partial results. Finally, in a similar fashion, the root can propagate a computed digest *down* the tree such that all nodes can maintain a current estimate for the digest.

The digest tree construction process is fully distributed and robust. The tree migrates adaptively when the current root fails, since digest diffusion will try to find the new value for V_{MAX}. Not all metrics are suitable to construct the aggregation tree. For example, the maximum node link degree is a bad choice because the node with the maximum degree (maximum number of neighbors) may change frequently over time. A stable tree can avoid short-term errors in the computed digest values caused by root switching. A digest tree based on the *maximum coarse-grained residual energy level* of a node tends to hold still over relative long time period. When the current root node is exhausted, the protocol changes the root of the tree to the next most energy-rich node in the network.

C. Digest Tree Maintenance

Maintenance of the digest tree against topology changes such as node failure and addition is also combined within the process of updating V_{MAX}: Each node periodically broadcasts a message $M = (m, s, h)$ for updating V_{MAX} every T_0

[3]Of course, if the network is continuously computing many digests in parallel, then piggybacking does not make sense. In that situation, one can combine the information required from several digests into one message and achieve similar amortization benefi ts.

[4]The advantage of piggybacking is that it can avoid the header and framing costs associated with sending the information on a separate packet. In addition, in sensor networks, waiting to piggyback the information on other transmissions can save the cost of turning on and off radios (*e.g.*, if the MAC layer has turned off the radio for power saving) compared to sending the information immediately.

TABLE I

TIME-OUT VALUE V.S. SOFT-STATE STABILITY

$\frac{T}{T_0}$	$\phi = 90\%$	$\phi = 99\%$	$\phi = 99.9\%$
$p = 10\%$	1	2	3
$p = 20\%$	1.4	2.9	5.2
$p = 30\%$	1.9	3.8	5.7
$p = 50\%$	3.3	6.6	9.7

seconds, as described in the previous sub-section. Topology adaptation is through soft-state techniques. The parent node identifier expires if node A does not receive any message from its current parent after T_p seconds.

Node A then switches to another node (including A itself) which provides the largest value with the smallest hop distance during the last T_p seconds. Similarly, each node keeps a timer T_c for the partial result sent by its child. Additionally, a sequence number or time-to-live value from the root is placed into each message to avoid possible looping when the root node itself crashes.

It takes T_p seconds to detect a parent node failure or disconnection. The time-out value has to be carefully selected. Ideally, we would set $T_p = T_0$ for fastest response to topology changes. However, the stability of the tree is equally important. Considering existence of packet loss, setting $T_p = T_0$ leads to significant oscillation in the tree structure. We quantify how stable the parent-child relationship is as follows:

$$\phi(T) = 1 - p^{\frac{T}{T_0}}$$

where p is packet loss probability for the link. ϕ is the probability that a soft-state is refreshed within T seconds. The goal is to minimize T for fast adaption to topology changes, while having certain bound on ϕ so that the tree structure is stable. Table I describes the relation between T, ϕ, and p. Conservatively, we choose $T_p = 4T_0$ in our experiments, which keeps the tree relatively stable even when all the links suffers packet loss as much as 30%.

V. IMPACT OF PACKET LOSS

Packet loss can significantly impact the computation of some classes of digests. In this section, we quantify this phenomenon by observing packet loss rates in a deployed wireless network under relatively benign conditions. These results suggest that our design of digest computation must explicitly deal with packet loss, the subject of our next section.

Several factors affect packet loss over a wireless communication channel. The signal strength fading effect leads to low signal noise ratio over long distances. Environmental interference, which may be sporadic or constant, also contributes to packet loss. Packet collision between multiple transmitters, particularly the hidden terminal problem, is another factor. As a special case of packet loss, an asymmetric link arises between a pair of nodes when only one can directly communication with the other. The use of transmission range control protocols can result in nodes to transmit with different powers. Even with the same power setting, different receivers may

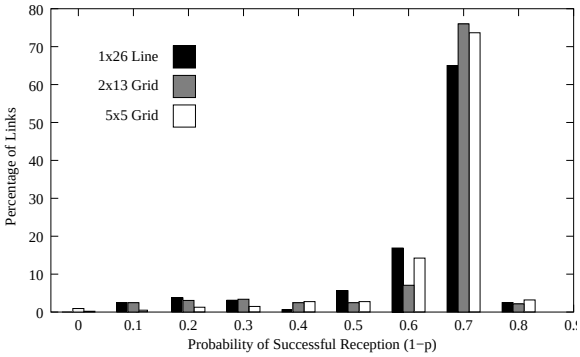


Fig. 2. Distribution of link quality

experience different levels of channel interference. Depending on those conditions, empirical studies show that heavy packet loss and link asymmetry can be quite common in wireless networks [13], [18].

Our measurements on a testbed consisting of motes (detailed description in Section VI) qualitatively confirm the same findings but in different environment. In our experiments, packet loss for each link is measured every minute for two hours in different topology settings. A link is defined as a good link if its average packet loss $p \leq 30\%$ or a bad one if $p \geq 80\%$. Figure 2 shows that common existence of links with heavy packet loss: Though majority of the links are good links, more than 10% of the links suffers average loss rate greater than 50%. To evaluate connectivity asymmetry, links between a pair of nodes are defined as *symmetric* if both are good links, and as *asymmetric* if one is a good link and the other is a bad link. A link pair are *relatively asymmetric* if their loss rate difference is greater than 35%. Table II indicates asymmetric links are also quite common. Statistics also shows that the packet loss of most links fluctuates over the time with an estimated variance of 9% to 17%.

TABLE II

Percentage OF SYMMETRIC AND ASYMMETRIC LINKS

Percentage	1x26 Line	2x13 Grid	5x5 Grid
Symmetric (2G)	72.5%	69.5%	81.1%
Asymmetric (1G1B)	12.5%	10.2%	7.2%
Symmetric Bad (2B)	3.8%	2.9%	3.1%
Relatively Asymmetric	17.5%	16.8%	12.3%

A. Impact on the Aggregation Tree

Not only is loss rate prevalent, it can also adversely affect the computation of digests, as this section shows.

In Section IV, we described a design for digest computation that involves constructing a digest tree. Figure 3 shows the results, on a 26-node linear topology, from the direct implementation of a V_{CNT} digest tree. Notice that the estimated count (shown by the dashed line) changes over time and results in significant error. An analysis of logs from the testbed reveals that the existence of heavy packet loss and link asymmetry *adversely* affects the tree construction protocol in Section IV.

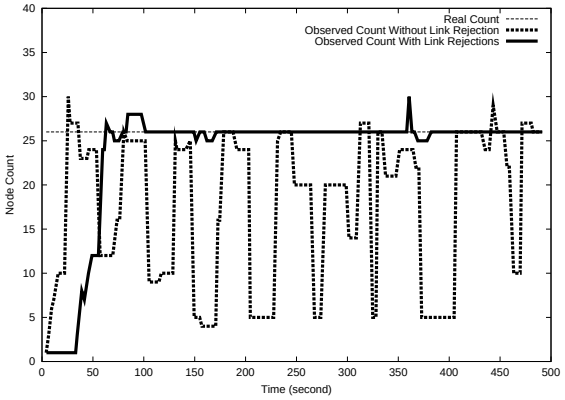


Fig. 3. A plot of CNT (1x26 network)

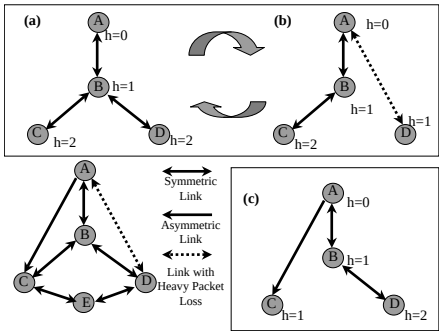


Fig. 4. Impact of packet loss on the aggregation tree

To understand this phenomenon, consider Figure 4 (a) and (b), where, from time to time, node D selects node A as its parent over B according to the shortest hop count rule. However since the link between D and A suffers heavy packet loss, the parent's soft-state at D soon expires and D switches back to B as its parent. Thus this branch "flapping" occurs all the time, and the final aggregate result is unstable. In another case of Figure 4(c), node C always chooses A as its parent. However, A is hardly able to hear any reports from C over the asymmetric link $A \rightarrow C$. The branch is stable but the partial result from C and its subtree is thus lost, which may lead to significant error if the subtree is large.

B. Link Quality Profiling and Rejection

So far, we have seen that:

- Packet loss and link asymmetry can be prevalent in wireless networks.
- That time-varying loss and asymmetry can result in oscillating digest tree branches, and thereby cause significant error in the computed digest.

To avoid this, we propose to selectively "blacklist" links with poor link quality or asymmetry from being on the tree. That is, when possible, each node will try to choose a parent with which it has "good" and symmetric communication. The challenge in doing this is to detect these links reliably, and adapt to time-varying conditions. We describe our approach to doing this, which involves using the digest tree construction messages to estimate losses to neighbors.

Recall that the digest tree construction message contains a sequence number. For each of its neighbors B, node A maintains a FIFO buffer $k_1, k_2, \cdots, k_m$ to store the sequence numbers in the most recently received beacons. Packet loss is estimated as $p_{AB} = 1 - m/(k_m - k_1)$. If no message is received from B for $10T_0$ seconds, the buffer is released. (*i.e.*, those links with $p > 90\%$ are ignored).

All incoming links are then qualified as "good" or "bad" according to the following rules: 1) Whenever $p_{AB} < \alpha$, link $A \rightarrow B$ is marked with "good". 2) Whenever $p_{AB} > \beta$, link $A \rightarrow B$ is marked with "bad". The heuristic is necessary given our observation that there exists fairly high variance in packet loss over the time. Assessment of link quality based on a single threshold still leads to frequent parent switching.

Each node now has a list L_{IN} of the good incoming links. To identify asymmetric links, each node periodically broadcasts L_{IN} to its neighbors. Instead of sending out the entire list, each item of L_{IN} is piggybacked into the digest computation message, in a round-robin fashion. The overhead of neighbor list exchange is then amortized over many messages. By listening if its identifier is mentioned by neighbors, each node can construct another list L_{OUT} of good outgoing links. To avoid the scenarios in Figure 4, only those neighbors in $L_{IN} \cap L_{OUT}$ are allowed to be parent candidates. With an upper bound of packet loss, the time-out values can easily be determined according to Table I to guarantee that tree is relatively stable.

VI. Experimental Evaluation

In Figure 3, the solid curve shows that with link profiling and rejection, the *computed digest is significantly more stable* than in an implementation that does not selectively choose tree links based on observed packet loss. This essentially validates our design, but we now describe our experiments in more detail, more carefully dissect our experimental findings, and quantify the performance difference that our scheme can bring about.

We implemented our scheme on the "mote" sensor platform [19]. Each node has a 4MHz Atmel microprocessor with 4 KB RAM, 128 KB code space and 512KB external EEPROM. Motes use TinyOS [20] which provides a MAC layer with a simple CSMA/collision avoidance protocol running on a 433MHz RFM radio transceiver at 40Kbps. In our experiments, we chose the transmission power setting such that the communication range is approximately 3-4 meters. Nodes are placed along a single line with inter-node distance of 1 meter. Each node has a degree of 4-6. Given only a limited number of nodes, our intention is to stress test our approach with the largest network diameter as much as possible. A single line formation does not reflect the full reality of a sensor field, but still captures the accumulated effect of multiple hops with loss and asymmetry on digest computation.

We implement three digests V_{MAX}, V_{CNT} and V_{SUM}. V_{AVG} is derived from V_{SUM}/V_{CNT}. The partial results for

TABLE III
DIGEST MESSAGE FOR MAX, CNT, SUM, AND AVG

id	$seqno$	V_{MAX}	Src_{MAX}	Hop_{MAX}	$Seqno_{MAX}$	V_{CNT}	V_{SUM}	$Parent$	$GoodNeighbour$

these three can be coded into a 18-byte message (Table III), which is periodically sent out every $T_0 = 6$ seconds with randomization. Note that partial results for V_{CNT} and V_{SUM} can be separated from V_{MAX} computation to further reduce the individual digest size, but we did not do this for simplicity. As proposed in Section IV, we choose timeout values $T_p = T_c = 4T_0$. We also choose sequence number buffer size $m = 5$, link quality profiling thresholds $\alpha = 25\%$, $\beta = 50\%$. These thresholds are reasonable values to identify the bad links, given the average packet loss and variance observed from Figure 2. The contributed value v_i for each node is uniformly distributed over the range $[0, 100]$. A node with the maximum value is intentionally placed on one corner. Partial results and node state are logged into EEPROM for post analysis. Each experiment takes around 2 hours. Experiments were repeated until 95% confidence intervals were achieved.

A. Communication Cost

The digest computation is based on periodic messages, thus each node consumes constant power transmitting $18/T_0 = 3$ byte/sec in our experiments. In addition, the energy expended for reception at each node is proportional to the node degree. Comparison between our approach to centralized solution is trivial as described in section IV: In-network aggregation can achieve an order of magnitude reduction on communication cost and thus leads to better energy-efficiency. Note that our choice of a small $T_0 = 6$ is primarily to reduce the experiment time. Larger value can be used in practice to further improve energy-efficiency.

B. Robustness to Packet Loss

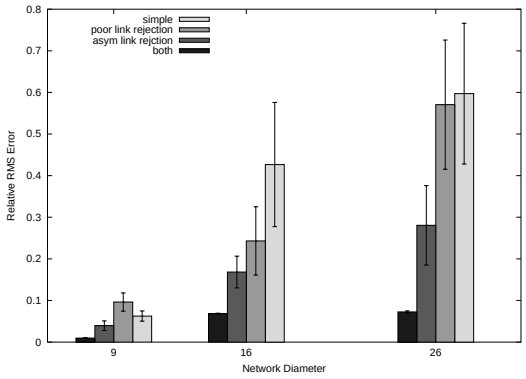
The primary characteristic to evaluate on our testbed is the robustness to packet loss and asymmetric links in the real world. Figure 3 shows that link rejection can reduce the error in digest computation dramatically. To *quantify* the performance improvement, we define the relative root mean square error in digest V as the follows:

$$\frac{1}{V}\sqrt{\sum_{t=1}^{T}(V_t - V)^2/T}$$

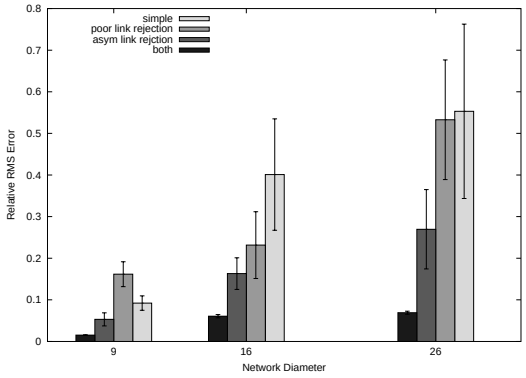
where V_t is the observed value at time t and V is th actual value.

We then compare our proposed solution against three schemes:

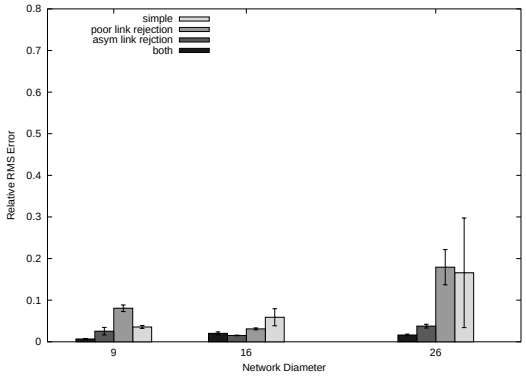
1) The digest computation algorithm without link rejection;
2) Scheme 1 plus rejection of poor incoming links;
3) Scheme 1 plus rejection of asymmetric (poor outgoing) links;



(a) Relative RMS Error for CNT



(b) Relative RMS Error for SUM



(c) Relative RMS error for AVG

Fig. 5. Relative Root Mean Square Error for 1x9, 1x16, 1x26 networks (Implementation)

4) Our proposed scheme: scheme 1 with both incoming and asymmetric link rejections turned on.

Figure 5 shows that the accumulated error over multiple hops increases significantly with larger network size. Without link rejection, the simple tree construction algorithm leads to error as much as 70% for V_{CNT} in a 1x26 configuration. However, with rejection of both poor incoming links and asymmetric links, the error can be reduced to less than 10% for the same network.

In Figure 5, the performance difference between scheme (2), (3) and (4) implies that both incoming link rejection and asymmetric link rejection are important to our design. A thorough analysis of logs shows that with only asymmetric link rejection (scheme 3), the oscillations depicted in Figure 4 (a)(b) are still quite common. On the other hand, rejection of poor incoming links (scheme 2) itself is sufficient to construct quite stable trees. However, in some cases, such a tree includes "stable" asymmetric links where partial results are constantly lost. Neither of them can achieve acceptable accuracy in digest computation by itself. It is interesting that scheme 3 outperforms scheme 2 in our experiments. Our explanation is that in scheme 3, that node B is a good outgoing neighbor of A implies that A can hear from B quite well, at least for those neighbor list broadcast messages. Thus the trees constructed in scheme (3) is more likely be a "good" aggregation tree compared to those in scheme (2).

Our experiments also show that different digests have different robustness characteristics. Monotonic exemplary digests such as V_{MAX} are the most robust digest with hardly no error all time except some transient errors when the node with maximum node crashes. Packet loss and topology changes hardly affect the final result given the network is still connected. V_{CNT} and V_{SUM} tend to be the most sensitive to packet loss since their accuracy rely on correctly collect the partial results from every node in the network. The robustness of V_{AVG} computation depends on the characteristics of the data set. In our experiments, V_{AVG} is more robust than V_{CNT} and V_{SUM} because the value set is from an uniform distribution. Even a fraction of samples leads to a good estimate for the average. In the next section, we will further evaluate by simulation the performance of digest computation with different distributions of contributed values.

C. Latency

Digest computation relies on piggybacking partial results on other periodical messages. The latency of response to value changes, node failures or topology changes is another important performance metrics. The response time to those scenarios is bounded by the convergence time of computation from the system initialization. We define the convergence time for V_{CNT} as the time that the moving average of the 10 recent results appears with less than 10% relative error. For V_{MAX}, we define the convergence time as the time that all nodes agree on the same V_{MAX}.

In Figure 6, the convergence time for V_{CNT} shows the trend approximately proportional to the network diameter. In addition, the convergence time t_{MAX} for V_{MAX} is smaller than t_{CNT} for V_{CNT}. We further conjecture that t_{MAX} is approximately half of t_{CNT}. Because from system initialization, V_{CNT} computation need t_{MAX} time for constructing the aggregation tree and approximately the same amount of time to propagate all the partial results back to the root.

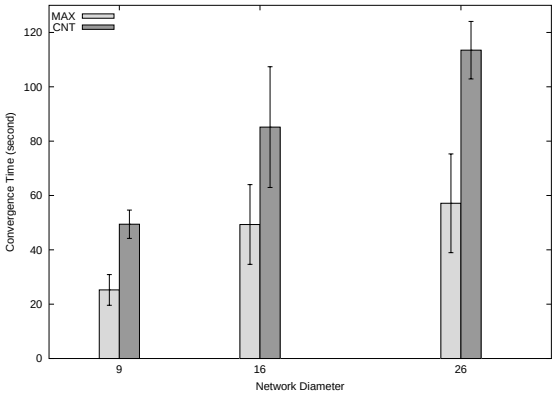


Fig. 6. Latency for MAX and CNT

VII. Simulation Results

We use the TinyOS simulator [20] to study the impact of scale on our conclusions. The simulator is modified the simulator with the following packet loss model: The packet loss over distance d is defined:

$$p(d) = \begin{cases} 0.2d/r & d \le r \\ 1.6d/r - 1.4 & r < d \le 1.5r \\ 1.0 & d > 1.5r \end{cases}$$

where r is the nominal transmission range. The packet loss slightly drops to 20% at r and then drops sharply to 100% at $1.5r$. This model is artificial but it reflects our observation from the testbed experiments. In addition, asymmetric links are generated by setting r for each node randomly from range of 2 to 4 meters. The system parameters such as T_0,T_p, α, β are used with the same values in the testbed experiments except otherwise noted. Each simulation runs for approximately 40 minutes.

A. Scalability

Once again, the communication and computation overhead at each node is constant over time. The primary metric to study scalability is the error that introduced by packet loss and asymmetric links in large networks. In this set of simulations, nodes are randomly placed on a $m \times m$ area, with an approximate density of 1 node per square meters. Figure 7 shows the errors in V_{AVG} and V_{CNT} for the networks with 100-900 nodes. With link rejection, the proposed digest computation protocol scales well with the network size and is robust enough to provide accurate results.

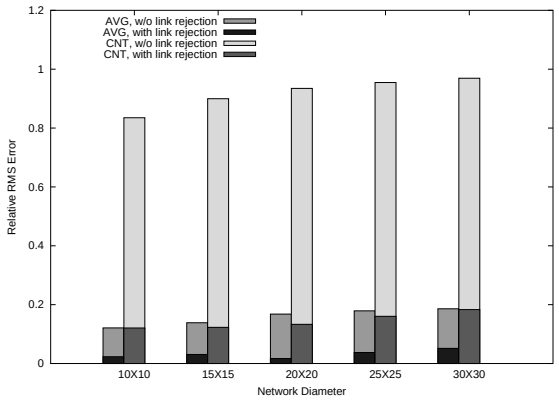


Fig. 7. Relative RMS Error for V_{AVG} and V_{CNT} (Simulation)

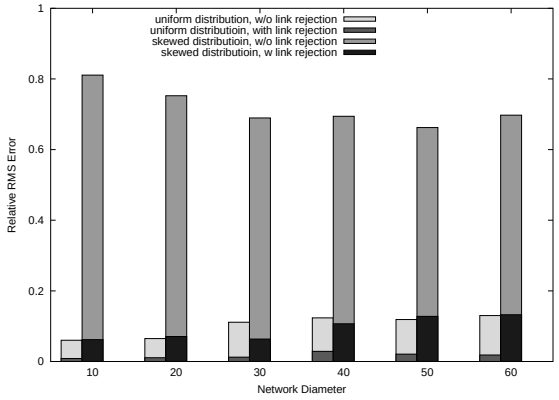


Fig. 8. Relative RMS Error in V_{AVG} for Different Data Sets

B. Sensitivity to data distribution

Our experiments on the testbed (Figure 5) and simulation (Figure 7) show that different digests have different levels of sensitivity to packet loss. In particular, it seems that V_{AVG} is significantly more robust than V_{CNT} and V_{SUM}. However, as we show in this section, the robustness of digest computation also depends on the distribution of the contributed values. We simulated digest computation with two different distribution models of the value for each node. The first one is the uniform model $[0, 100]$ that we use in previous experiments. The second one is a "skewed" distribution: 10% of the nodes have a value uniformly from $[90, 100]$, the rest nodes have value of 0 or 1. *i.e.* a small fraction (10%) of nodes contributes a large fraction (95%) of the sum.

We simulate V_{AVG} computation with different number of nodes on a linear formation. Figure 8 shows the different impacts of these two distributions on V_{AVG} computation: for the uniform distribution, the error tends to converge when the network size increases. From a uniformly distributed data set, a fraction of samples can provide an average estimate such that additional sample loss does not introduce more error. The "skewed" distribution behaves differently. Without link rejection, the large values from very few nodes tends to be lost due to high accumulated packet loss. Thus the result of V_{AVG} computation constantly suffers significant error for different network sizes. However, with link rejection, the digest computation protocol can reduce the error significantly.

VIII. Related Work

The problem of monitoring sensor networks is crucial and important. Recently, different protocols are proposed to discover node deaths [21], [22], compute the coverage and exposure bounds of wireless sensor networks [23], [24], provide remaining energy supply indication [15] or discover the topology of the network [25]. These approaches address various specific aspects of sensor network monitoring and are complementary to the digest aggregation tools in this paper. In addition, they can fit into our proposed architecture to provide a coherent monitoring system for wireless sensor networks.

Computation of aggregates has been discussed in database research community such as in [6]. Madden *et al.* recently proposed a generic framework to support aggregate queries from base-stations in sensor networks [5], [9]. Our solution eliminates any predefined base-stations or hierarchy, and provides an energy-efficient and robust aggregation with little extra overhead. In addition, we also address the impact of packet loss from empirical studies on a real wireless sensor network testbed, which turns out to be crucial to the accuracy of aggregate computation. Though this paper is not intended to address the design of generic sensor network applications, some techniques proposed here can be applied to efficiently compute global or regional aggregates for applications.

There are many proposed solutions to deal with packet loss and asymmetric links in mobile ad-hoc networks. To name a few, associativity-based routing protocol [26] uses a route stability metric for routing in mobile ad-hoc networks. The objective is to select long lived links according to the associativity of the nodes involved. In signal stability based adaptive routing [27], routing is based on both the signal strength and location stability. The link quality is estimated according to the signal strength of received beacons from its neighbors. In [28], a sub-layer called Sub Routing Layer between the network layer and the MAC layer is proposed to provide a bidirectional abstraction of any unidirectional network to the routing protocols. Our proposed link profiling and rejection technique is similar in spirit, but focuses on the context of lower-power wireless sensor network monitoring, given its even tighter energy efficiency requirement. Though mobility is not a primary challenge for sensor networks, our approach does tolerate certain level of node mobility.

Another orthogonal class of approaches is to improve scalability with randomized sampling techniques. For example, in the context of multi-cast group size estimation [29] [30] [31], only a small fraction of multi-cast participants send out replies to the querying node by suppressing replies from the others. However, depending on the statistical nature of the data set, these centralized solutions may require high sample probability to achieve reasonable accuracy. In addition, they assume that the communication cost to disseminate any message to the network is constant, which does not hold for

multi-hop sensor networks where energy-efficiency is crucial. We are starting to investigate the trade-offs to incorporate such techniques into digest computation.

IX. Conclusions and Future Work

In this paper, we propose an architecture to monitor wireless sensor networks with different levels of detail, and focus on the design of computing network digests. Digests represent continuously computed summaries of network properties and can serve to indicate the need for more detailed, but perhaps energy-intensive, monitoring.

We have implemented digests, and presented our evaluation from a medium-scale testbed of motes. Carefully dealing with heavy packet loss and asymmetric links can significantly reduce the error of digest computation, as we have shown. We presented a simple scheme that selectively avoids adding links with heavy loss or asymmetric links to the digest tree.

We would like to continue our experiments on a larger scale testbed and further evaluate our design. Ultimately, we intend to make a suite of monitoring tools available that will foster larger scale experimentation in sensor networks.

References

[1] D. Estrin, R. Govindan, and J. Heidemann, "Embedding the Internet," *Communications of the ACM*, vol. 43, no. 5, pp. 39–41, May 2000, (special issue guest editors).

[2] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocols for wireless microsensor networks," in *Proceedings of the Hawaii International Conference on Systems Sciences*, Jan. 2000.

[3] P. Varshney and C. Burrus, *Distributed detection and data fusion*, New York, Springer, 1997.

[4] I. Gupta, R. van Renesse, and K. Birman, "Scalable fault-tolerant aggregation in large process groups," in *Proc. Conf. on Dependable Systems and Networks*, 2001.

[5] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad hoc sensor networks," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[6] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *J. Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 29–53, 1997.

[7] D. Raz and Y. Shavitt, "New models and algorithms for programmable networks," in *Computer Networks*, Feb. 2002, vol. 38(3), pp. 311–326.

[8] 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*, Boston, MA, USA, Aug. 2000, pp. 56–67.

[9] S. R. Madden, R. Szewczyk, M. J. Franklin, and D. Culler, "Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks," in *Proceedings of the Workshop on Mobile Computing Systems and Applications*, 2002.

[10] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the IEEE Infocom*, June 2002.

[11] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Rome, Italy, July 2001, pp. 70–84.

[12] A. Cerpa and D. Estrin, "ASCENT: Adaptive self-configuring sensor networks topologies," in *Proceedings of the IEEE Infocom*, New York, USA, June 2002.

[13] D. De Couto, D. Aguayo, B. Chambers, and R. Morris, "Performance of multihop wireless networks: Shortest path is not enoughi," in *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, New Jersey, USA, Oct. 2002.

[14] J. Case, M. Fedor, M. L. Schoffstall, and C. Davin, "Simple Network Management Protocol," in *Internet Request For Comments 1908*, Nov. 1996.

[15] Y. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, Mar. 2002.

[16] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, WA, USA, Aug. 1999, pp. 174–185.

[17] P. Tsuchiya, "The Landmark Hierarchy: A new hierarchy for routing in very large networks," in *ACM Computer Communication Review*, Aug. 1988, pp. 35–42.

[18] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: An experimental study of low-power wireless sensor networks," in *Technical Report UCLA/CSD-TR 02-0013,Computer Science Department, UCLA*, July 2002.

[19] M. Horton, D. Culler, K. Pister, J. Hill, R. Szewczyk, and A. Woo, "MICA, the commercialization of microsensor motes," in *Sensors Magazine*, Apr. 2002, pp. 40–48.

[20] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for network sensors," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, USA, Nov. 2000, pp. 93–104.

[21] S. Chessa and P. Santi, "Comparison based system-level fault diagnosis in ad-hoc networks," in *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, Oct. 2001.

[22] J. Staddon, D. Balfanz, and G. Durfee, "Efficient tracing of failed nodes in sensor networks," in *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, USA, Sept. 2002, pp. 122–130.

[23] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava, "Coverage Problems in Wireless Ad-hoc Sensor Networks," in *Proceedings of the IEEE Infocom*, 2001.

[24] S. Meguerdichian, F. Koushanfar, G. Qu, and M. Potkonjak, "Exposure In Wireless Ad Hoc Sensor Networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, July 2001, pp. 139–150.

[25] B. Deb, S. Bhatangar, and B. Nath, "A topology discovery algorithm for sensor networks with applications to network management," in *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, USA, Sept. 2002.

[26] C. Toh, "Associativity-based routing for ad-hoc mobile networks," in *IEEE Personal Communications Magazine*, 1997.

[27] R. Dube, C.D. Rais, K. Wang, and S.K. Tripathi, "Signal stability based adaptive routing for ad-hoc mobile networks," in *IEEE Personal Communications Magazine*, Feb. 1997.

[28] V. Ramasubramanian, R. Chandra, and D. Mosse, "Providing A Bidirectional Abstraction for Unidirectional Ad-Hoc Networks," in *Proceedings of the IEEE Infocom*, June 2002.

[29] J. Nonnemacher and E. W. Biersack, "Optimal multicast feek backup," in *Proceedings of the IEEE Infocom*, San Fransisco, USA, Mar. 1998, pp. 964–971.

[30] T. Friedman and D. Towsley, "Multicast session memebership size estimation," in *Proceedings of the IEEE Infocom*, New York, USA, Mar. 1999, pp. 965–972.

[31] S. Alouf, E. Altman, and P. Nain, "Optimal on-line estimation of the size of a dynamic mutilcast group," in *Proceedings of the IEEE Infocom*, New York, USA, June 2002.