USENIX Association

Proceedings of MobiSys 2003:
The First International Conference on
Mobile Systems, Applications, and Services

San Francisco, CA, USA
May 5-8, 2003

Energy-Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks

Sagnik Bhattacharya, Hyung Kim, Shashi Prabh, Tarek Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

Abstract

In recent years, large distributed sensor networks have emerged as a new fast-growing application domain for wireless computing. In this paper, we present a distributed application-layer service for data placement and asynchronous multicast whose purpose is power conservation. Since the dominant traffic in a sensor network is that of data retrieval, (i) caching mutable data at locations that minimize the sum of request and update traffic, and (ii) asynchronously multicasting updates from sensors to observers can significantly reduce the total number of packet transmissions in the network. Our simulation results show that our service subsequently reduces network energy consumption while maintaining the desired data consistency semantics.

1. Introduction

Sensor networks are ad hoc wireless networks made of large numbers of small, cheap devices with limited sensing, computation, actuation, and wireless communication capabilities. Such a network, for example, can be dropped from the sky on a disaster area to form collaborative teams of programmable nodes that help with rescue operations. Sensor networks are made possible by advances in processor, memory and radio communication technology, which enable low-cost mass-production of sensor-equipped wireless computing nodes.

The sensor network paradigm is motivated by applications such as guiding rescue efforts in disaster areas, monitoring poorly accessible or dangerous environments, collecting military intelligence, tracking wild-life, or protecting equipment and personnel in unfriendly terrains. In such environments, it is usually impractical to build fixed infrastructures of powerful and expensive nodes. Instead, the sensor networks philosophy advocates the use of myriads of inexpensive nodes strewn arbitrarily in the environment and left largely unattended.

The primary function of sensor networks is the collection and delivery of sensory data. Power is identified as one of the most expensive resources. Due to the difficulty of battery recharging of thousands of devices in the remote or hostile environment, maximizing battery lifetime by conserving power is a matter of great importance.

In this paper, we develop a distributed framework that improves power conservation by application-layer sensor data caching and asynchronous update multicast. The goal of the framework is to reduce the total power expended on the primary network function; namely, data collection and delivery.

The importance of optimizing communication cost is also supported by measured data from recent prototypes of sensor network devices, which show that the main power sink in the network is, indeed, wireless communication. For example, the Berkeley motes [15] consume 1 μJ for transmitting and 0.5 μJ for receiving a single bit, while the CPU can execute 208 cycles (roughly 100 instructions) with 0.8 μJ. Assuming full load, CPU power consumption is about 10mW, compared to 50mW for the radio. The high power cost of communication makes it a prime candidate for optimization.

The remainder of this paper is organized as follows. Section 2 presents the service model and the formulation of the power minimization problem. Section 3 presents the details of the data placement middleware and its API. Section 4 presents an evaluation using experimental as well as simulation results. Section 5 reviews the related work. The paper concludes with section 6.

2. Service Model

Consider a dense *ad hoc* wireless sensor network with multiple observers, spread over a large monitored area. At any given time, the observers' attention is directed to a relatively limited number of key locales in the network, where important events or activities are taking place. We call them *focus locales*. For example, in a disaster area scenario, rescue team members may be interested in monitoring survivors. The locations of found survivors therefore represent the focus locales of this application. The total number of sensor nodes is assumed to be much larger than the number of focus locales at any given time.

Sensor nodes at each focus locale elect a local representative for communication with the rest of the world. Distributed leader election algorithms may be

borrowed for this purpose from previous literature and are not the goal of this paper. Our service adopts a publish-subscribe model, as shown in Figure 1. In this model, each representative publishes sensory data about its focus locale to observers who subscribe to a corresponding multicast group to receive such data. The size of the published update stream originating at a given locale is time-varying, depending on the volatility of the environment and the type of sensors involved. An environment, which changes frequently, will generate more update traffic than a quiescent environment. Similarly, sound sensors (microphones) will generate more traffic than temperature sensors.

Contrary to previous multicast frameworks for sensor networks, update traffic is multicast from focus locales to receivers in an *asynchronous* manner. Data caches are created at the nodes of the multicast tree. A lazy algorithm is used for propagating data updates among neighboring caches along the tree in the direction of the receivers. These receivers may be wireless hand-held devices or laptops, for example, in the possession of rescue team members operating in a disaster area. We assume that receivers do not move, or move slowly compared to communication delays in the network.
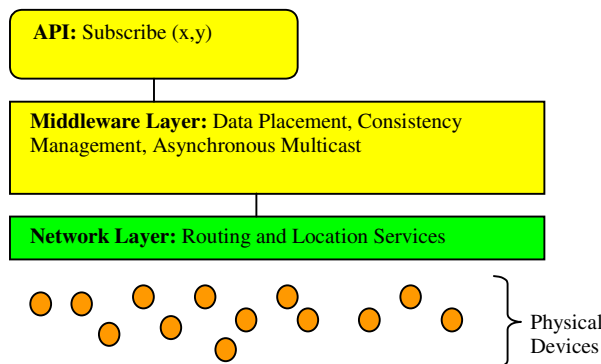

Fig. 1: Middleware Architecture

In general, data updates can be either *accumulative* or *non-accumulative*. An example of accumulative updates is recorded sound. To receive a continuous recording, *all* (or *most*) sound samples should be communicated. An example of non-accumulative updates is thermal measurements. If the application is interested in the current temperature only, past temperature updates need not be reported. Most real-time sensor outputs, with the general exception of multimedia data, are non-accumulative in that current measurements subsume stale measurements. Hence, our scheme is restricted to non-accumulative updates. This decision is also motivated by the fact that current sensor network technology is too slow to handle multimedia traffic in a cost-efficient way.

While in this paper we do not consider streaming multimedia, an argument in favor of addressing such traffic in sensor networks is that more powerful devices may become available in the foreseeable future at an affordable price. We argue, however, that advances in sensor network technology are most likely occur in *two* directions: developing more powerful devices of the same form factor, and developing smaller devices of the same processing and communication capacity. Research reported in this paper is more relevant to the latter direction.

In our model, observers who join the asynchronous multicast tree specify a period at which the requested data should be reported. Flurries of changes in the environment need not be individually reported if they occur at time-scales smaller than this period. Different observers may specify different period requirements for the same measurement. For example, an observer who is close to the measured activity may request a higher reporting rate than a distant observer.

Our middleware achieves four main functions; (i) it determines the number of data caches for each focus locale, (ii) it chooses the best location for each cache such that communication energy is minimized, (iii) it maintains each cache consistent with its data source at the corresponding focus locale, and (iv) it feeds data to observers from the most suitable cache instead of the original sources.

A key difference between this problem and the problem of caching in an Internet context is that in the latter case, the topology of the network restricts the choice of cache locations. In contrast, we assume a sensor network that is dense enough such that a data cache can be placed at any arbitrary physical location in the monitored region, offering new degrees of freedom to the data placement algorithm. Another key difference is that the number of Internet proxy caches is typically much smaller than the number of different web sites. Hence, such caches are centralized powerful machines, which gather and retain content from a large number of distributed sources. In contrast, in this paper, we consider a middleware caching service, which runs on *every* sensor node. Since the number of sensor nodes is larger than the number of focus locales, the storage requirements of this service on any single node are very small.

We assume that sensor nodes know their location. Algorithms for estimating geographic or logical coordinates have been explored at length in the sensor network research [5][6]. These efforts address the problem of location awareness using algorithms that do not require high cost devices such as GPS on every node. Classical ad hoc wireless routing protocols like AODV [8], and DSDV [9] may be used along each unicast edge of our data dissemination tree. These protocols, however, are not location-aware which may

affect performance. Several more recent adaptations such as Location-aware routing (LAR) [7] and geographical forwarding [4] make use of the location information. These routing algorithms would be a natural choice for the network layer underneath our service. We now formulate our data placement problem mathematically.

2.1.   Problem formulation

Consider a sensor network that is monitoring a set of focus locales at which events of interest occur. Given a locale *(X,Y)* in a sensor network, let BS= {BS_1, BS_2,....., BS_M} be a set of *M* observers that request data from that locale with rates R_{req}={R_1, R_2,...., R_M}. Let sensor updates at *(X,Y)* occur at an *average* rate R_{update}. A tree of copies is created for the sensor as shown in Figure 2.
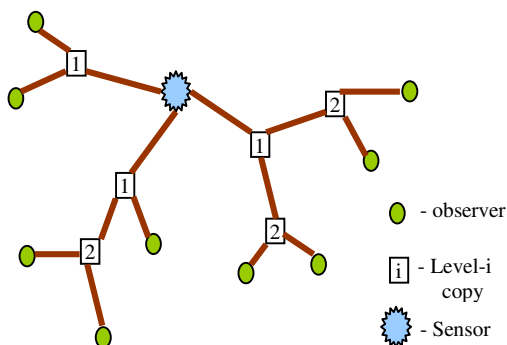

Fig 2: Creating a hierarchy of copies

We define the cost of message transfer between two nodes in the tree as the power expended on a packet's transfer on the shortest route multiplied by the packet rate. Consider the case of placing a single data copy to minimize cost as defined above. Let the data copy be placed at a distance n_i hops from the ith observer and at a distance n_{sens} hops from the sensor node serving the data. In a densely populated network, the hop counts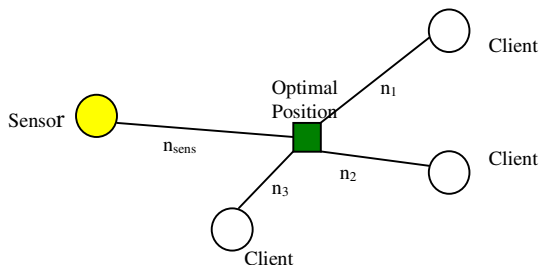 will be large. The cost of sending a single packet is proportional to the hop count. Hence, the net cost of serving all observers is:

$$T = n_{sens} \cdot R_{update} + \sum_{1 \leq i \leq M} n_i \cdot R_i \qquad (1)$$

To place the copy at the optimal location, *T* has to be minimized. Figure 3 shows the situation with three observers. We can reduce this problem to the following geometric optimization. Given *N* points, where point *i* is at location (X_i, Y_i), find a point *(x,y)* such that $D = \sum_{1 \leq i \leq N} (d_i \cdot w_i)$ is minimum, where, d_i is the distance of the ith point from *(x,y)*, and w_i is the weight of the edge from the ith point to *(x,y)*. This is illustrated in Figure 4. A heuristic solution to this problem is to place *(x,y)* at the center of gravity of the *N* input points in question, i.e.:

$$x = \sum_{1 \leq i \leq N} x_i w_i / \sum_{1 \leq i \leq N} w_i \qquad (2)$$
$$y = \sum_{1 \leq i \leq N} y_i w_i / \sum_{1 \leq i \leq N} w_i \qquad (3)$$

Hence, in a minimum-cost tree with multiple copies (i.e., multiple internal vertices), each copy *(x,y)* should be at the center of gravity of those vertices to which it is connected. The objective of our algorithm is to find such a tree.

In the following, we compare our formulation to other popular variants of content placement problems described in prior literature. If the number of copies in the tree is known in advance, a popular variation of the problem is expressed as a Minimum *K*-median problem, stated as follows. Given n points (possible copy locations), select *K* of them to host data copies, and feed each observer from a copy such that total communication cost *D* is minimized, where:

$$D = \sum_{1 \leq j \leq K} \sum_{1 \leq i \leq N} c_{ij} \cdot y_{ij} \qquad (4)$$

c_{ij} is the cost of the edge from *i* to *j* and y_{ij} is 1 if the jth copy serves the ith observer, and 0 otherwise. Many Internet-based content placement algorithms adopt this model. In this case, the possible locations of the caches are fixed. Hence, c_{ij} is fixed for the given network topology. The problem is NP-hard, but heuristic solutions are possible, e.g., [10] and [11]. If the cache locations are specified, a minimum spanning tree can be constructed to disseminate information from senders to receivers at the lowest cost.
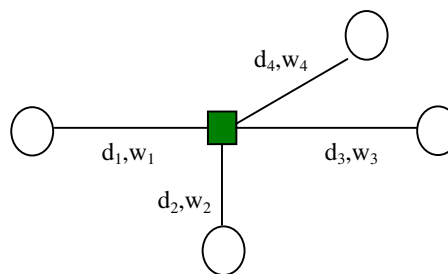

Fig. 3: Scenario for M = 4. The three base-stations are served by the same copy.

Fig. 4: Corresponding geometric problem for N=4.

Our model differs in that copy locations are not known *a priori*. In a dense sensor network, the number of nodes n approaches infinity. Copies can essentially be placed anywhere in the Euclidean plane without restrictions. In this case, the problem is that of constructing a minimum-cost weighted Steiner tree, which connects the sensor node to the observers.

The Steiner tree formulation differs from the K-median and spanning tree problems in that it allows one to create new nodes in the tree as opposed to having to choose from a pre-specified set of possible node locations. This difference separates our paper from similar work in web caching and content distribution literature.

Note that R_{update} in our algorithm is not a fixed sampling rate, but rather refers to the average rate of change of the environment. Hence, it may vary dynamically with environmental conditions. For example, it may decrease when the environment is quiet. An advantage of such dynamic adaptation is that no energy is wasted when no updates occur. A disadvantage is that an application is unable to tell when it has missed an update (e.g., due to message loss), since it does not expect updates to arrive at particular time intervals. This problem can be solved in several ways.

First, we may let R_{update} be a fixed sampling rate. The formulation of our algorithm remains the same. In this case, if a sample does not arrive in time, the application can tell. Alternatively, the origin sensor may number the updates. If a gap occurs in the received update numbers, the application is aware that a previous update was lost. The occasional loss may be acceptable since we assume that only the latest update is relevant at any given time. A potential problem with the latter approach is that in the absence of subsequent environmental changes, an important update may be lost, unbeknown to the application, indefinitely. One solution is to enforce an upper bound, *B*, on the update period. Hence, when the environment is quiet a message is expected at least once every *B* seconds. Otherwise, the application is aware of a problem. In the rest of the paper, we shall not address the issue update loss any further.

3. Data Placement
Upon perturbation, distributed physical systems such as weights interconnected by strings settle into an equilibrium position, which represents a minimum energy state. Our data placement algorithm is inspired by such systems. Assuming environmental conditions don't change, each step of the algorithm reduces a measure of total energy until a minimum energy tree is found. More specifically, we use a distributed greedy heuristic that iteratively places each node at the center

of gravity of its neighbors. Note that, while in a physical system, energy has a direct meaning, in our system energy is an abstract mathematical quantity. We call the depth of the copy in the distribution tree rooted at the origin sensor, the *copy level*. The original data at the sensor is referred to as the level-*0* copy. A heuristic is used to add or remove copies in the tree. The algorithm is described in more detail next.

3.1 The Algorithm
Each node on the multicast tree rooted at the sensor maintains a location pointer to its parent as well as a location pointer to each of its children. One can think of these pointers as an application-layer routing table. For each child, the node maintains the *maximum propagation rate*, which is the maximum of all requested update rates of all observers served by that child. A node never forwards updates to a child at a rate higher than the child's maximum propagation rate. This way, flurries of environmental updates that exceed some receivers' requested rates are not propagated unnecessarily to those receivers.

3.1.1 Joining the Multicast Tree
An observer, *k*, joins a multicast tree by sending a join() message to the location of the origin sensor, i.e., to the level-*0* copy. The message indicates the location of the observer and its desired update rate R_k. The origin sensor forwards the message along the multicast tree in the direction of the new observer as follows. Each level-*i* copy (starting with the origin sensor), upon receipt of the join message, determines if the new observer is closer to any of its children than to itself. If so, it forwards the join message to the corresponding child, i.e., to a level-*(i+1)* copy. If the maximum propagation rate for that child is lower than R_k it is changed to R_k. This recursive forwarding terminates when a node is found with no children that are closer to the observer. We call this copy the *nearest neighbor*. The nearest neighbor adds the observer to the set of its children. The maximum propagation rate for the observer is initialized to its requested update rate. Figure 5 illustrates the message exchange in the join process.

3.1.2 Copy Creation and Migration
For the purposes of creation of new cache copies, nodes are differentiated into fixed and migratory. The origin sensor and observers are fixed nodes. Other nodes are migratory nodes that can move to better locations of fork off new copies.

When a newly joined observer is connected to its nearest neighbor N, node N computes the center of gravity of itself and all its neighbors. It then computes the savings, if any, resulting from creating a new copy

at that center of gravity. If the savings from creating the copy exceed a threshold, the option of creating this copy is deemed *viable*. Before we proceed further, let us look more closely at how the copy may be created.
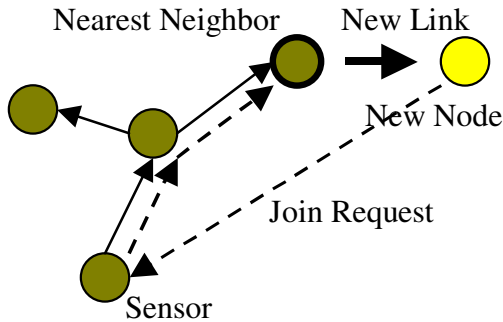

**Fig. 5:** Joining the Multicast Tree

If N (the nearest neighbor) is the origin sensor, the new copy can only be created *downstream* from it. The copy would be fed from N and in turn feed N's children as shown in Figure 6-a. Otherwise, if N is not the origin sensor, the new copy can in principle be created either downstream or upstream from N. An *upstream* copy would be fed from N's parent and would feed both N and N's children as shown in Figure 6-b. A downstream copy would be created as described above (Figure 6-a). Observe that, if N is not a fixed copy, a third option is also possible. Namely, it is possible to simply move N to a new position. This is called copy *migration*. In copy migration, when a newly joined observer is connected to a migratory nearest neighbor N, the node computes the center of gravity of all its neighbors (including the new observer), and evaluates the savings that would arise if it moves to the computed position. If the difference is larger than a fixed threshold the option of migration is deemed *viable*. This is illustrated in Figure 6-c.

A viable option with the maximum savings among three data placement options described above is executed. It is easy to show that no new copies are created unless there are three nodes in the system, and that at most one copy is created for every newly joined member. Hence, the algorithm creates at most *m-2* copies where m is the total number of observers.

3.1.3 Leaving the Multicast Tree

An observer, k, leaves the multicast tree by sending a leave() message to its parent N. The parent stops forwarding messages to the departed observer. If k had the highest maximum forwarding rate among N's children, N resets its own maximum forwarding rate to that of the next-highest rate child. If N is a migratory node, it computes the center of gravity of all remaining
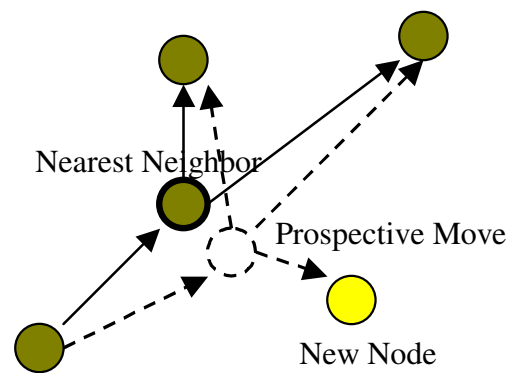
neighbors, computes the savings that result from moving to that center, and moves there if the savings exceed a threshold. If there is only one child left for the migratory node, the node is deleted and its parent takes over its child.
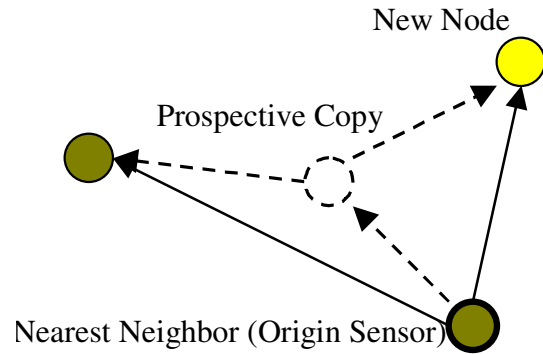
(a) Nearest neighbor creates downstream copy
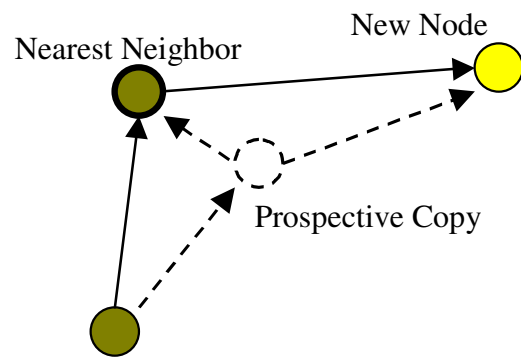
(b) Nearest neighbor creates upstream copy

(c) Nearest neighbor moves

Fig. 6: Copy Creation and Migration Rules

3.2 Sampling R_{update}

To perform center of gravity computations, nodes must know not only the requested observer rates, but also the environmental sensor update rate, R_{update}. There are two simple approaches towards the measurement of that rate. One approach is to measure the number of updates over the last n seconds. A disadvantage of this approach is that it has a fixed time horizon after which it forgets the past. It may be more advantages to adapt the horizon to the current rate of updates itself, such that system agility is increased when activity is high. An approach for calculating R_{update}, which has the aforementioned adaptive property, is to take the inverse of the average of the last k inter-arrival times. More complicated methods like predictive modeling could have been used but this would be limited by the computation and storage resource constraints of the nodes. For our simulation purposes we have used a simple model where R_{update} is calculated as the inverse of the average of the last five inter-arrival times. The number five is selected as the sample size to reflect that we expect any five consecutive updates to be strongly correlated, though a larger or smaller number could be chosen according to how volatile we expect the environment to be.

4. Evaluation

Our current service implementation utilized Berkeley motes [15] as the underlying distributed platform. These are tiny computing devices, which run a microthreaded operating system called TinyOS [16]. Each node has up to three sensors. It runs on an 8-bit 4 MHz micro-controller and has 128KB of program memory and 4KB of data memory. However, the number of motes available to us at present is insufficient for large-scale experiments. Hence, in the first set of experiments, we use the motes only to derive communication and power consumption characteristics that are then fed to a simulator. Accordingly, we also implemented our data placement middleware in the ns-2 simulator [20]. Our goal in simulating the data placement algorithm is to test whether it actually conserves power as expected given the power and communication characteristics of the motes. Our simulation model is a network of ($200 \leq N \leq 500$) nodes in a 200m x 200m grid each having a radio range of 20m. The packet sizes are kept as 30 bytes, as used by the Berkeley motes running TinyOS. The number of base-stations is roughly 5% the number of nodes in the network. Since we have to model a location-aware network, we assume that each node knows it own location. We implemented a wrapper, which works on top of the simulated routing protocol in ns-2, and translates the destination location of packets into actual destination node addresses. A focus

locale in the network is generated at random for simulation purposes, and the base-stations send periodic queries to the hot spot. The request rates are generated at random with a specific average throughout the experiment. The energy consumption is measured in terms of Joules per node per flow. Each value is taken as the average over three simulations.

At peak load the Berkeley motes consume about 60mW of power [15]. Of this 60mW about 10mW is consumed by the microcontroller unit. However, the MCU is not loaded about 50% of the time, and it has an idle mode in which it consumes only 40% of the normal power. Another very important fact to note is that more than 90% of the CPU utilization is due to bit, byte and packet level processing. Thus reduction in the number of packets in the network leads to lower CPU utilization and hence even greater power savings. However for simulation purposes, we have no accurate way to model CPU utilization in ns-2, so we measure only the radio power consumption, and the energy spent in communication. Since we discount savings in CPU power consumption, the actual power savings of the algorithm may be higher than shown in this section. The importance of optimizing communication cost is also supported by measured data from recent prototypes of sensor network devices, which show that the main power sink in the network is, indeed, wireless communication. Energy consumption for communication in our simulation follows the Berkeley motes [15] which consume 1 μJ for transmitting and 0.5 μJ for receiving a single bit. We also use the two-ray ground model ($1/r^4$) as the radio propagation model and an omni-directional antenna having unity gain in the simulation.

Geographic forwarding (GF) [29] is found as a routing protocol appropriate for the sensor networks. It is therefore used in the simulation. GF makes a greedy decision to forward a packet to a neighbor if it has the shortest geographic distance to the destination among all neighbors and it is closer to the destination than the forwarding node. To illustrate the appropriateness of GF, we compare its performance to that of AODV whn they are used in conjunction with data placement. AODV is a reactive routing algorithm developed for ad hoc wireless networks. It computes and caches routes on-demand. As shown in Fig. 7, the power consumption measured for GF is lower than that for AODV. The primary reason is that AODV does not leverage geographical information, thereby consuming more energy on route discovery. Hence for our simulations, we use GF as a routing algorithm.

4.1 Simulation Results

We compare the performance of the data placement middleware against four baselines; (i) a simple unicast-based query-response model, (ii) update multicast

(synchronous push model) (iii) directed diffusion, and (iv) update flooding.
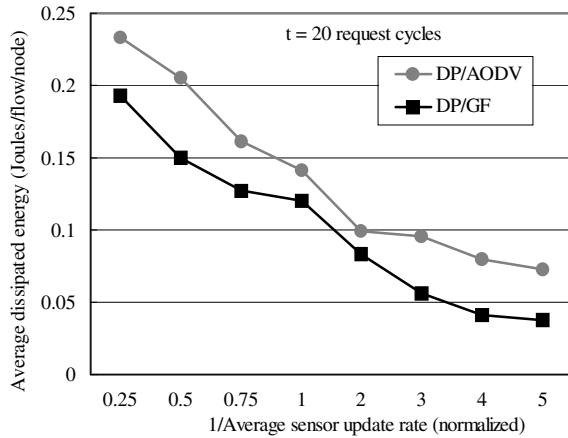

Fig 7: Average dissipated energy of data placement over AODV and GF

The first experiment (Figure 8) compares the energy consumption of the four aforementioned baselines for different node densities. For this experiment, the request rate is set, on average, to about two times the average update rate of the environment. Hence, regular (as opposed to asynchronous) multicast should be an optimal policy. We vary the number of nodes in the network over a grid of 200m x 200m. As we can see from Figure 8, flooding performs very badly. Traces reveal that power is wasted on both excessive communication and collisions caused by the update messages flooding through the network. The query-response scenario performs much better than flooding, but not as good as data placement. Directed diffusion performs almost as good as data placement. As expected, regular multicast performs best. The slight difference between regular multicast and data placement is due to the somewhat higher overhead of our scheme. As we show shortly, this overhead is offset by considerable savings when the average update rate increases beyond the request rate.

Unlike regular multicast, a main feature of the data placement algorithm is that it is adaptive and sensitive to changes in average sensor update rates. When the sensor update rates are high, more replicas are refreshed at a rate determined by the request rates and when the update rate is low, the copies are refreshed only when an update actually occurs. Next we compare the performance of the four baselines to that of our adaptive heuristic as the average update rate is changed. These communication models are compared over a network of 400 nodes in a 200m x 200m grid each having a radio range of 25m. The sensor update rate is normalized against the average request rate. The

sensor update period is varied between 1.5 s and 30 s. The average request rate is one per six seconds.
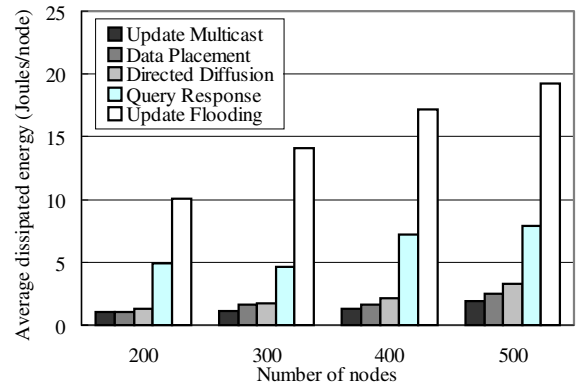

Fig 8: Average dissipated energy

In Figure 9, we show the average energy consumption in the steady state after all observers have joined the tree.
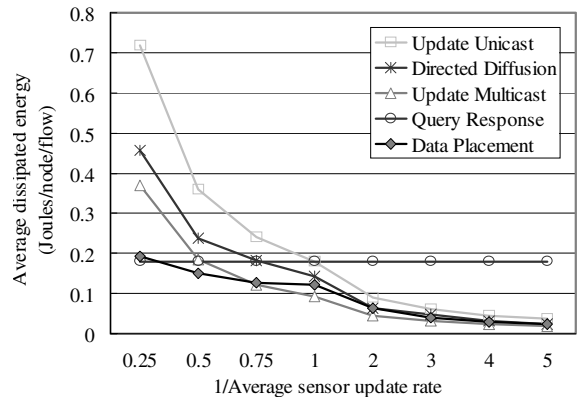

Fig 9: Steady-state dissipated energy plotted against sensor update rate.

As can be seen from the figure, when the Sensor Update Rate is high (i.e., 1/ Sensor Update Rate <1), the data placement algorithm performs better than the simple query response model, the directed diffusion, and update unicast because it does not send unnecessary updates. When the network is more quiescent, it achieves results, which are quite close to the simple multicast strategy. The difference between the two is the power overhead of adaptation. Thus, our data placement middleware adapts to the volatility of the environment resulting in a performance that is better than update multicast when the update rate is low, but does not suffer a performance degradation in when the sensor update rate is high.
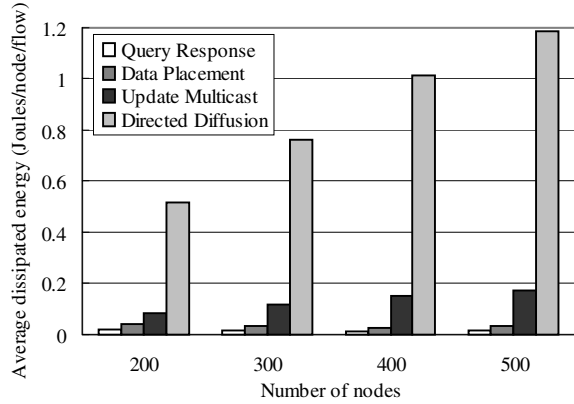
Fig 10: Energy consumption in transient phase

The next experiment is carried out in order to measure energy consumed when a new observer joins the tree. In the data placement, the new observer sends a join message to the origin sensor and the nearest node is found to connect the observer. Figure 10 shows the average energy consumption per join during the transient phase of tree construction. It is evaluated under the same environmental conditions as in the preceding experiment. The transient phase in the case of directed diffusion includes both initial flooding and multi-path sending before a single path is chosen by the gradient. The data placement heuristic consumes less energy than the update multicast and the energy dissipation does not depend upon the number of nodes in the network.

In our data placement algorithm, when a new data copy is placed, neighbors of the new copy are no longer in the center of gravity of their own neighboring nodes. Hence, as a rule, when the number or locations of neighbors of some data copy change, the copy recalculates its position to the new center of gravity. If the difference between the cost at the new position and the cost at the old one is larger than a given threshold, the old copy migrates to the new position. This migration, in turn, may cause other surrounding nodes to migrate. The effect percolates through the tree branch until no more nodes need to be moved. Figure 11 shows a transient and steady state energy consumption graph against various threshold values in the data placement heuristic. A smaller threshold results in more aggressive tree optimization. It entails more overhead for tree construction, but results in lower power consumption at steady state. Conversely, an extremely large threshold makes the tree essentially more static. Transient overhead is reduced, but steady state power consumption increases due to a less optimized tree. Observe that the difference between power consumption at the two extremes is not that pronounced. This is because network density is not infinite. Hence, a newly computed copy location is

likely to fall in a void between sensor nodes. As an approximation, the copy is mapped to the closest node from that void. Even if the threshold is zero, it is often the case that the new (optimized) position of a copy is close enough to its current position so that the copy is mapped to the same node as before. No move is therefore taken making the performance more similar to that of a static tree.
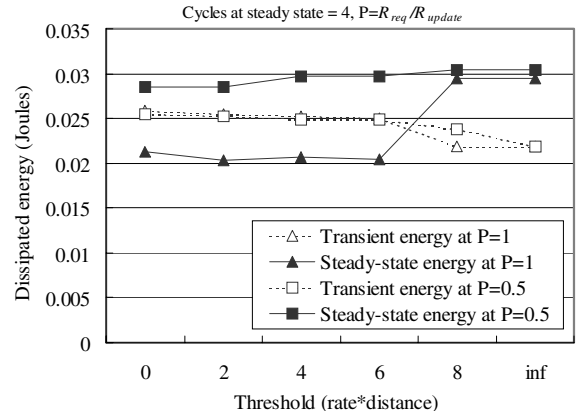

Fig 11: Energy consumption in adjusting data placement

Next, we test system performance when the average update rate is a non-stationary process, which changes drastically over short periods of time. We use the average update traffic pattern shown in Figure 12. The sensor switches between high and low average update rates with a period T. This average rate is used to determine actual update inter-arrival times, drawn from a Poisson distribution. This load model is used to simulate quiescent periods in the environment interlaced with volatile periods. At high update times, the update rate is made to be a Poisson distribution with mean of 0.5 times the maximum request rate, and at low update times it is made to be a Poisson distribution with mean of 3 times the request rate.
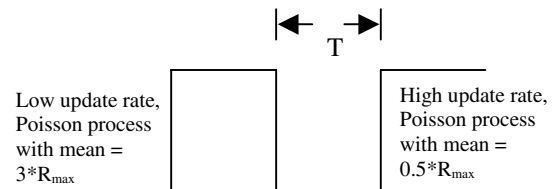

Fig 12: Update pattern

Figure 13 shows the performance of the data placement algorithm for different values of T. The network size was fixed at 300 nodes over a 200m x 200m grid. When T is sufficiently larger than R_{max} (R_{max} is the maximum request rate), data placement performs better than both query-response and update multicast, because it adapts to suit both the high and low update rates. As the average update rate is calculated as the inverse of the average of the last five inter-arrival times, when T becomes less than $5R_{max}$, this sampling becomes increasingly inaccurate. Since the average update rate keeps changing rapidly, there are more failures and hence, the communication savings decrease as might be expected.
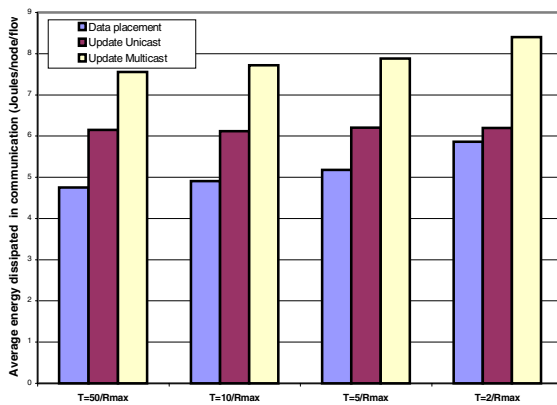

Fig 14: Lifetime of nodes in a sensor network using data placement

4.2 Effect of Sampling of R_{update}

As mentioned in section 3.3, there are several ways for accurately measuring R_{update}. We argue that inaccuracy in measuring/predicting R_{update} has very little impact on the overall performance of Data Placement. Figure 15 evaluates the sensitivity of power savings to the choice of the averaging interval for the sensor reported rate, R_{update}. Averaging intervals 5 and 10 are compared. The simulated traffic is the same as in Figure 12 and the experimental setup is the same as the experiment of Figure 13. The figure shows that power savings are insensitive to the averaging interval.

Fig 13: Average dissipated energy for the three models for different values of T

Another important concern to address is how the energy savings translate into an increase in the lifetime of the network. Figure 14 shows the fraction of nodes that are alive versus network run-time. We simulate a network of 400 nodes, keeping other parameters the same as earlier. Observers and the origin sensor are set to have infinite energy. A point to be noted here is that not all the nodes consume energy equally as some nodes are more active than others. When a certain number of nodes die, the network becomes partitioned, and parts of the network may become unreachable. At this point we consider the network unusable. Figure 14 shows the time it takes the network to get partitioned under various communication schemes. (The end of the plot indicates when the network gets partitioned.) It is shown that the system lifetime with the data placement heuristic is longer than with other baselines. As shown in Figure 9 and Figure 10, the data placement heuristic results in fair energy savings in both the steady state and the transient state. Hence, it increases system lifetime.
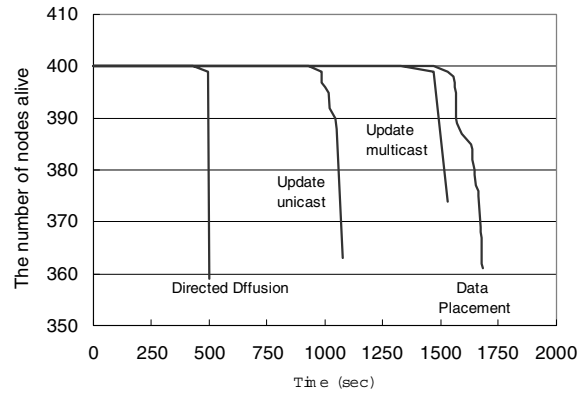

Fig 15: Average Dissipated Energy using different sample sizes for R_{update}

Thus, our simulation results show that our data placement middleware gives us considerable energy savings irrespective of the amount of load or the dynamic nature of the network. A point to note is that the energy savings are not uniform over the whole network. They depend on the location of the base-stations and the locations from where data is requested, just as the communication without data placement

would have expended energy non-uniformly. In general, we expect the results to improve with network size. This is because our main savings come from the use of adaptively constructed multicast trees. Since the size of a well-balanced tree is logarithmic in the number of recipients, power savings compared to unicast should increase exponentially with tree size.

4.3 Prototypical Testbed Implementation

To conclude our results, we constructed an experimental prototypical mini-testbed from a 5 by 5 grid of motes with one light sensor at a corner, and three base-stations (Figure 16). We ensured that a node can hear its immediately adjacent and diagonal neighbors. The three base-stations request light data from the sensor node, as shown in Figure 16. The base-station at the bottom right corner is connected to the serial port of a PC, where an application package reads the packets being received at the base-station.
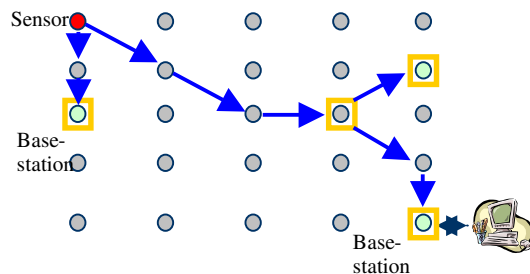

Fig 16: Sensor Network Testbed

The underlying routing protocol used is geographic forwarding, in which a node forwards a packet to the neighbor that is closest to its destination. The interface to the routing protocol accepts TinyOS commands from the data placement middleware to send messages to a given node. In addition, it signals an event when the message transmission is complete. The data placement layer handles this event.

The base-station (0) communicates with the PC using the serial port. Thus this base-station has both a RS232 communication channel as well as RF communication. In order to distinguish between the two channels, a special address (0x7e) is assigned to the serial port interface. Thus a device receiving a packet for this destination forwards the packet to the local UART instead of the radio.

The light sensor acquires 10-bit values for light intensities. The sensor data is acquired by polling the light sensor. We define a resolution range of 5 bits, i.e., an update occurs only if any of the 5 most significant bits change. We calculate the value of R_{update} by sampling the last 5 inter-update times.

Our implementation of data placement has about 250 lines of code (C statements), and the complete

package including TinyOS and geographic forwarding takes about 14.5KB of program memory. Thus memory-wise, the middleware is not heavyweight in terms of memory.
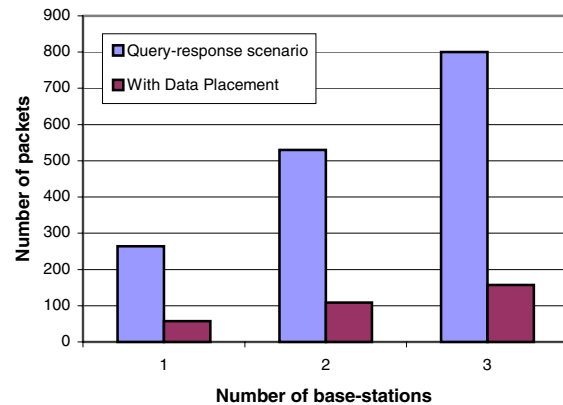

Fig 17: Number of packets using the two approaches

As has been discussed earlier, communication is the main sink for energy. The motes have an idle mode in which the mote draws 5μA compared to 5mA in the peak model. To estimate actual energy consumption on the motes, we measure the amount of communication in terms of the number of packets in the network. Figure 17 shows the number of packets in the network over a period of 200 seconds in a single trial run using data placement and without data placement. Here the sensor gets updated approximately once every 12 seconds. The base-stations request data once every 5 seconds. For the cases on one and two base-stations, all possible combinations of the three base-stations are considered, and the mean is taken.

Though the savings in the number of packets are an order of magnitude, this does not translate into proportional energy savings, since some power is expended on listening (even when nothing is being received). Even so, we have shown that we can get significant reduction in the number of packets in the network.

6. Related Work

Wireless sensor networks are a relatively new area of research. Traditional networking paradigms are not directly applicable to this scenario. However, there has been a lot of work lately on developing new paradigms and services for sensor networks, taking into account the unique features of these networks. New protocols are being developed for routing, MAC, data dissemination and location services. Several hardware platforms as well as specialized operating systems have also been developed. Since there are a number of parallel efforts, several different paradigms and protocols are bound to come up. Our data placement

algorithm makes minimal assumptions about the underlying layers and the other supporting services.

One of the essential properties of a sensor network is that all the nodes are location aware. Since the nodes cannot afford to have heavyweight GPS, they have to use some location service, that estimates the location of the individual nodes using some GPS equipped nodes as beacons. Nagpal [5] and Bulusu, Heidemann et al. [6][12] proposed location estimation using beacons which does not require GPS at every node.

In a network with thousands of nodes, it would be wasteful to assign unique id's to each node. Also, queries would be addressed by location, and that would necessitate a location directory service, if unique id's were used. This would consume more resources. Papers by Heidemann, Silva et al. [3] and Imielinski, Goel [13] proposed addressing by geographical location and attributes. Our data placement algorithm shall work both for fixed addresses and addressing by location. For the case in which fixed addresses are used, a location directory service is used to lookup mapping from node-id's to locations.

To conserve energy from communication, Xu, Heidemann, and Estrin [4] have proposed a geographical adaptive fidelity (GAF) algorithm in which equivalence classes of nodes are formed from a routing perspective. Their method of energy conservation is to put nodes to sleep whenever possible. Our data placement algorithm can co-exist with GAF, by putting the constraint that nodes that are holding copies of the data cannot go to sleep until they handoff their data to another node.

The formation of a tree of copies along which the update is propagated is similar to the formation of a multicast tree, where all the nodes are members of the multicast group. In that sense, our work is related to multicast protocols. However, our approach differs from traditional multicast routing in several respects. First, updates are propagated asynchronously in a lazy manner in accordance with consistency constraints. Second, the depth of our tree is determined by the update and the request rates, and it adapts itself to minimize the communication. Finally, our work is an overlay multicast algorithm that works on top of the network layer, rather than traditional multicast routing [23][24] that takes place at the network layer. In wired networks, End-to-end multicast and Scribe based on Pastry, are included into overlay multicast designs.

Data placement is also similar to some of the ideas used in the placement of web server replicas [1]. In these schemes, replicas of web server content are placed online based on predicted demand. Data placement furthers this idea by using the property of location-awareness of the sensor nodes. Another approach used in [21] is to let the documents themselves decide their replication strategy. However,

in our case we make use of the homogeneity of the data, and the fact that the sensors do not differentiate between types of data, to provide for a unified replication strategy. Another approach has been geographical push-caching [22] in which the server decides when and where to cache the files. This technique would not work for a sensor network because a sensor node cannot keep track of all the base-stations it is serving. In our approach, the sensor node only needs to keep track of a small number of level-1 copies.

ShopParent algorithm [30] is the latest work of publish/subscribe tree construction in the wireless adhoc network. This greedy algorithm builds the tree in a distributed fashion and uses a spanning tree to find a better outcome. Its model uses every node in the network for tree construction, which makes searching for the nearest node easier. However, the model is not available in the sensor network with a large number of nodes. The ShopParent is also different from ours in that it does not use a Steiner tree for multicast and does not use location information.

In recent years, research on Content Distribution Networks has focused on replication and placement of content to improve performance over a large scale distributed systems. Most of the work focuses on internet-type topologies and scale [26][25]. Chord [26] uses a variant of consistent hashing to map a key to a node. Its scalability lies in the fact that the amount of state that needs to be maintained by each node scales logarithmically with the number of Chord Nodes. In [27], the concept of Content Addressable Networks (CAN) is used for providing hashing like functionality to retrieve data from replicas in the network. CAN routing uses a co-ordinate routing table, and the network is visualized as a d-dimensional space. It extends functionalities of DNS by providing a flexible naming scheme. However replication in CAN is more hotspot driven. Another system, OceanStore [25], provides an infrastructure designed to span the globe and provide access to persistent information. The primary difference between these systems and data placement and replication in a sensor network is that the data placement algorithm presented in this paper leverages the location information available to the nodes to reduce power consumption. In contrast, work on CDN falls into two categories. It either (i) ignores physical topology altogether, focusing on peer-to-peer protocols defined in a logical overlay space, or (ii) optimizes the weight of the tree assuming a heterogeneous network of known topology with point-to-point links of different bandwidth. This optimization is not applicable to wireless sensor networks, where the physical network topology is unknown, yet it is the physical (i.e., geographic not overlay) tree that needs to be optimized for power consumption. Geographic

information and energy consumption have been not considered in aforementioned papers.

Finally, the problem addressed in this paper is somewhat reminiscent of data placement in distributed shared memory systems [20]. As in shared memory systems, data in sensor networks can be thought of as a set of objects manipulated by *read()* and *write()* operations. The *write()* operations are performed by sensors. The *read()* operations are performed by users. In shared memory systems, it is desired to maximize the average performance of memory accesses given some desired data consistency semantics. In sensor networks, the problem is to minimize average power consumption per data access subject to data consistency constraints. Both problems reduce to finding algorithms for appropriate dynamic placement of data objects in the network such that communication is minimized. Sensor networks, however, due to their fine granularity, large scale, and direct interaction with the physical environment, exhibit significantly different data access patterns, consistency constraints, and communication cost models than do distributed shared-memory systems. Hence a new set of algorithms is called for to achieve power minimization.

7. Conclusion

In this paper, we have presented data placement as a means of reducing energy consumption and, hence, increasing the lifetime of a sensor network. We present an algorithm, which places copies of the requested data and updates them so as to minimize the communication overhead and power consumption of data transfer.

Our algorithm is completely distributed and requires very little local processing. The amount of bookkeeping involved is small, which fits in nicely with the constraint of limited memory resources. Also it makes minimal assumptions about the underlying MAC and routing layers, although pro-active routing algorithms like DSDV are not a good choice for these types of networks.

In conclusion, data placement is a new approach for energy conservation in wireless sensor networks. To our knowledge, very little previous work has been done to apply data placement to a location-aware network.

Further work such as accommodating quickly moving observers and accounting for node failures needs to be done to introduce guarantees into the data placement model. In the big picture, data placement may act as a service that aids in providing power saving and QoS guarantees to applications running on these sensor networks. This is analogous to how web-caching and content distribution help in providing better performance and guarantees over the Internet.

References
1. Lili Qiu, Venkata Padmanaban, Geoffrey M Voelker, On the Placement of Web Server Replicas, Proc. IEEE INFOCOMM 2001.
2. Chalermek Intanagonwiwat, Ramesh Govindan and Deborah Estrin, Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks, In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCom 2000), August 2000, Boston, Massachusetts.
3. John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In Proceedings of the Symposium on Operating Systems Principles (SOSP 2001), Lake Louise, Banff, Canada, ACM. October 2001.
4. Ya Xu, John Heidemann, and Deborah Estrin, Geography-informed Energy Conservation for Ad Hoc Routing, Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2001), Rome, Italy, July 16-21, 2001.
5. N. Bulusu, J. Heidemann and D. Estrin, GPS-less Low Cost Outdoor Localization For Very Small Devices, IEEE Personal Communications, Special Issue on "Smart Spaces and Environments", Vol. 7, No. 5, pp. 28-34, October 2000.
6. Radhika Nagpal, Organizing a Global Coordinate System from Local Information on an Amorphous Computer, MIT AI Memo 1666, August 1999
7. Young-Bae Ko and Nitin H. Vaidya, "Location-Aided Routing(LAR) in Mobile Ad Hoc Networks," In Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1998), ACM, Dallas, TX, October 1998.
8. C. E. Perkins and E. M. Royer, "Ad-hoc On Demand Distance Vector Routing." 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99), New Orleans, Louisiana, February 1999.
9. Charles E. Perkins and Pravin Bhagwat, Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers, in *SIGCOMM Symposium on Communications Architectures and Protocols*, (London, UK), pp. 212-225, Sept. 1994.
10. M. Charikar, S. Guha, E. Tardos and D.B. Shmoys, A constant factor approximation algorithm for the k-median problem. Proceedings of the 31st Annual ACM symposium on Theory of Computing.

11. M. Charikar and S.Guha, Improved Combinatorial Algorithms for the Facility Location and K-median Problems. In Proc. Of the 40th Annual IEEE Conference on Foundations of Computer Science, 1999.

12. N. Bulusu, J. Heidemann and D. Estrin, Adaptive Beacon Placement, Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, Arizona, April 2001.

13. Tomasz Imielinski and Samir Goel, "DataSpace - querying and monitoring deeply networked collections in physical space," IEEE Personal Communications Magazine, Special Issue on Networking the Physical World, October 2000.

14. Philippe Bonnet, J. E. Gehrke, and Praveen Seshadri. "Querying the Physical World," IEEE Personal Communications, Vol. 7, No. 5, October 2000, pages 10-15. Special Issue on Smart Spaces and Environments.

15. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, " System architecture directions for network sensors," ASPLOS 2000.

16. Development platform for self-organizing wireless sensor networks, Proc. SPIE, Unattended Ground Sensor Technologies and Applications, Vol. 3713, p. 257-268.

17. A Compendium of NP optimization problems http://www.nada.kth.se/viggo/problrmlist/compendium.html

18. Self-organizing distributed sensor networks, Proc. SPIE, Unattended Ground Sensor Technologies and Applications Vol. 3713, p. 229-237.

19. The ns-2 simulator. http://www.isi.edu/nsnam.

20. Distributed Operating systems. Andrew S. Tanenbaum Prentice Hall.

21. Guillaume Pierre, Maarten van Steen, andAndrew S. Tanenbaum, *Self-replicating Web document*s, Technical Report IR-486, Vrije Universiteit, Amsterdam, February 2001,

22. The Case for Geographical Push Caching. *Proceedings of the Fifth Annual Workshop on Hot Operating Systems*, Orcas Island, WA, May 1995

23. E. M. Royer and C. E. Perkins, Multicast operation of the ad-hoc on-demand distance vector routing protocol, in Proc. of ACM/IEEE Intl. Conference on Mobile Computing and Networking (MOBICOM), Aug. 1999

24. A Survey of Multicast Technologies (2000), Vincent Roca, Luís Costa, Rolland Vida, Anca Dracinschi, Serge Fdida September 2000.

25. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gumadi, S. Rhea, H. Weatherspoon, W. Weimer, C.Wells, B. Zhao. Oceanstore: An Architecture for Global-scale Persistent Storage. In the *Proceedings of ASPLOS 2000*, Cambridge, Mmassachusetts, Nov. 2000.

26. I. Stoica, R. Morris, D. Karger, f. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-Peer lookup Service for Internet Applications. In *Proceedings of ACM Sigcomm 2001*, San Diego, CA, Aug. 2001.

27. S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM Sigcomm 2001,* San Diego, CA, Aug. 2001.

28. A. Woo, and D. Culler. A Transmission Control Scheme for Media Access in Sensor Networks, Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2001), Rome, Italy, July 16-21, 2001.

29. B. Karp and H. Kung, Greedy Perimeter Stateless Routing, in Proc. of the Sixth Annual ACM/IEEE Intl. Conference on Mobile Computing and Networking (MOBICOM), Boston, 2000.

30. Yongqiang Huang, Hector Garcia-Molina. Publish/ Subscribe Tree Construction in Wireless Ad-Hoc Networks, 4th International Conference on Mobile Data Management, January, Melbourne, Australia, 2003.