

Data Placement for Energy Conservation in Wireless Sensor Networks

Sagnik Bhattacharya

Department of Computer Science

University of Virginia

sagnik@cs.virginia.edu

Tarek Abdelzaher

Department of Computer Science

University of Virginia

zaher@cs.virginia.edu

Abstract

In recent years, wireless sensor networks have emerged as a new fast-growing application domain for distributed computing. Evolution of wireless sensor networks leads to new research challenges in both the distributed systems and the embedded systems arenas because of unusual application requirements, resource constraints, scale, and functionality. Energy conservation is of prime importance in a sensor network. The amount of battery power consumed is directly proportional to the amount of communication. Power-aware data communication protocols and services are therefore needed to reduce energy consumption and increase network lifetime.

In this paper, we present a distributed sensor-network middleware service whose purpose is power conservation. The service sits on top of the network's routing layer and performs data placement and caching as a strategy to conserve battery power. We show that since the dominant traffic in a sensor network is that of data retrieval, caching data at key locations, and offering those locations as a dimension to manipulate, can significantly reduce the total number of packet transmissions in the network. Our and simulation results show that our service subsequently reduces network energy consumption and decreases client-perceived network response time while maintaining the desired data consistency semantics.

Keywords: *Sensor networks, data placement, power-aware computing, embedded systems*

1. Introduction

Advances in processor, memory and radio technology have made it possible to build small and cheap sensor nodes that are capable of short-range wireless communication and a limited amount of computation and storage. Their availability shall enable the deployment of networks of thousands of nodes for a wide range of applications to monitor poorly accessible or dangerous environments such as disaster areas, neighborhoods of volcanic activities, hostile territories (e.g., behind enemy lines), and active nuclear fields. Unfriendly or dangerous terrains make it impossible 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. Such a highly distributed network shall have a high degree of built-in fault-tolerance, so that the failure of few nodes causes only a marginal decrease in the quality of data sensed.

The vision of sensor networks presents new and unique challenges arising from the highly constrained resources of individual sensor-equipped nodes, and the large scale of the overall network. Power is identified as the most expensive resource in a sensor network. In most cases, such networks are meant for one-time use, i.e., once the battery dies, the node dies too. Hence, maximizing network lifetime by conserving power is a matter of great importance.

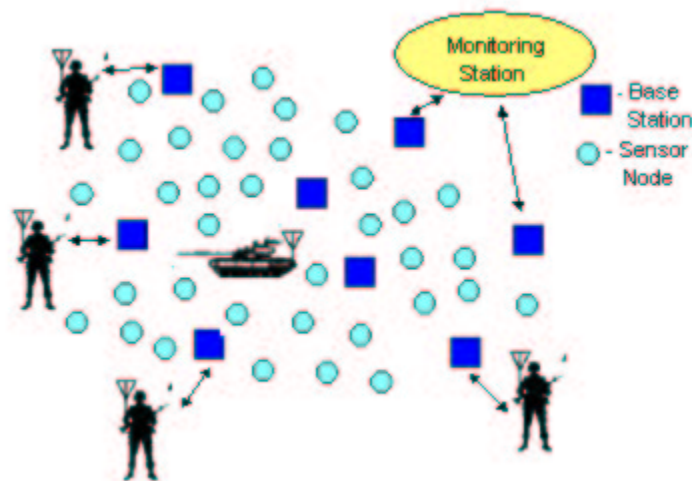


Fig. 1: A typical sensor network scenario

The main power sink in a sensor network is communication. Hence algorithms are needed for reducing the amount of communication in the network. In this paper, we adopt a data replication scheme as a primary mechanism for reducing the data retrieval traffic. We analyze the problem of replica placement that minimizes energy consumption. Replica placement involves finding the number of replicas, their locations, and the manner in which each replica should be updated to satisfy all consistency constraints. Our data placement strategy though not optimal, provides a good approximation. Our simulation results show that our strategy provides significant savings in battery power.

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

Our distributed middleware runs on both sensor nodes and base-stations above the network routing and location services. Base-stations generate queries that are typically addressed by location, eg., “What is the temperature at (x,y) ?”. Our middleware interjects these queries and maintains a table which has mappings from every query location (x,y) to a corresponding location (w,z) where a data replica is maintained by our service. The middleware achieves three main functions; (i) it chooses the best location (w,z) for each replica such that energy is minimized, (ii) it maintains each replica consistent with its data source, and (iii) it redirects the queries to the replicas instead of the original sources.

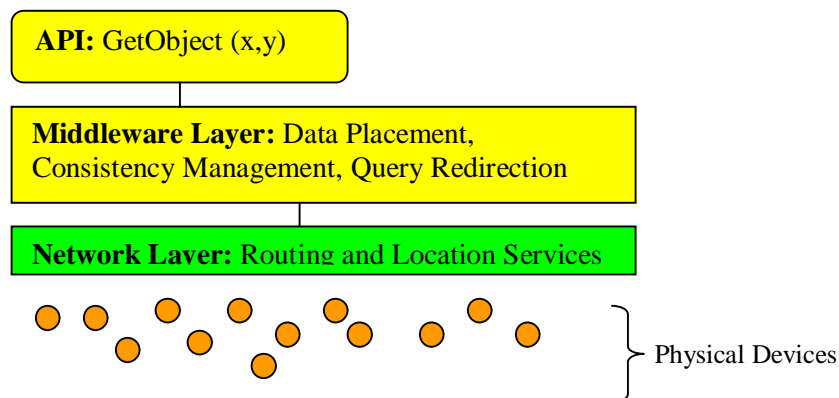


Fig. 2: Middleware Architecture

It is assumed 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]. In particular, many research efforts address the problem of location awareness using algorithms that do not require high cost devices such as GPS on every node. For example, in [5], a few nodes with built-in GPS act as beacons to disseminate location information. At deployment-time all nodes run a location detection algorithm to estimate their locations relative to the fixed beacons. Another approach proposed by Nagpal [6] uses local information and local communication to determine global coordinates of the nodes.

An explicit goal of our scheme is its independence from routing policies. Classical ad hoc wireless routing protocols like AODV[8], DSDV[9] may be used. These protocols, however, are not location-aware which may affect performance. Several more recent adaptations such as Location-aware routing (LAR) [7] make use of the location information. Another approach is geographical forwarding [4], in which packets are forwarded in the general direction of the destination. We do not make assumptions about the routing algorithm used except that it will use a reasonably direct route to the destination most of the time.

We assume that the application requires weak data consistency. In particular, we make the following two assumptions. First, we assume that changes in the environment that are below some noise threshold, d , need not be reported. Hence, as long as the environment is quiescent, no data updates need to occur. An update $\text{write}(x)$ occurs only when the absolute change in the value of variable x exceeds the threshold d . We call the resulting average update rate of x , $R_{\text{update}}(x)$. Second, we assume that each client i has a period, R_i at which data is requested. Flurries of changes in the environment need not be individually reported if they occur at time-scales smaller than this period. In general, different observers may have different periods.

Subject to these consistency constraints, the prime requirements of the data placement middleware are energy conservation, reduction of query response time, reduction of communication overhead, and scalability. We now formulate our data placement problem mathematically.

2.1. Problem formulation:

Given a network of N nodes, let $BS = \{BS_1, BS_2, \dots, BS_M\}$ be a set of M base-stations that request data from a certain location (X, Y) with rates R_1, R_2, \dots, R_M . The sensor data at (X, Y) changes at a rate R_{update} . We want to create a hierarchical structure of copies such that the net traffic is minimized. A tree of copies is created such that updates are propagated along the tree, and the base-stations get served from nodes of the tree. The main problem is to form the tree such that communication is minimized. *Fig. 3* illustrates the problem. In this figure, the solid edges represent the update paths and the red edges represent the dashed edges represent the request paths. Whenever, an update occurs at the sensor node, it gets propagated along the tree, and all the copies residing at the nodes get updated, and the base-stations get the data from these copies according to their request rates.

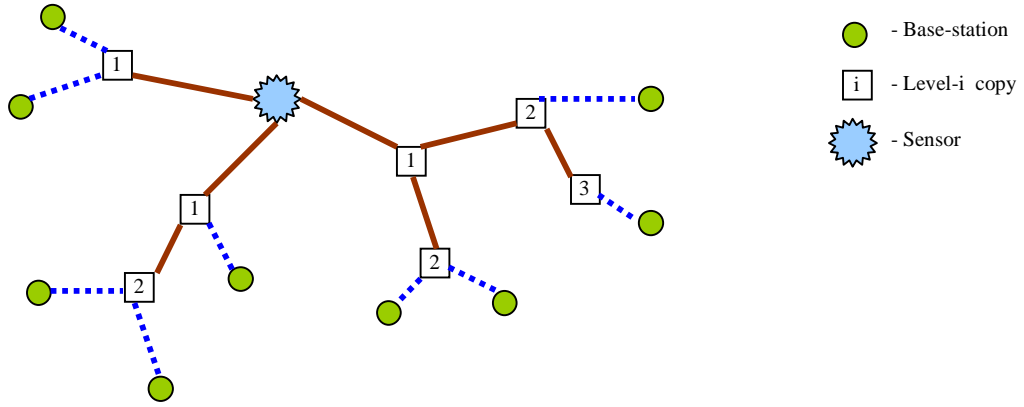


Fig 3: The problem of creating a hierarchy of copies

To illustrate the main idea, consider first a one level hierarchy. Consider a data copy placed at a distance of n_i hops from the i^{th} base-station and at a distance of n_{sens} from the sensor node serving the data. Then the net cost in serving the base-stations from the copy is:

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

To place the copy at the optimal location, T has to be minimized. *Fig. 4* shows the situation for $M=4$. We can reduce this problem to the following geometric optimization problem: Given N points, 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 i^{th} point from the point (x, y) and w_i is the edge weight of

the edge from the i^{th} point to (x,y) . This is illustrated in *Fig. 5* for $N=4$ for one level of copies. For K copies at the first level, the cost function is,

$$D = \sum_{1 \leq j \leq K} \sum_{1 \leq i \leq N} d_{ij} \cdot w_{ij} \cdot y_{ij} \quad \dots\dots(2)$$

where, d_{ij} is the distance of the i^{th} point from the j^{th} copy, w_{ij} is the weight of the edge from i to j and y_{ij} is 1 if the j^{th} copy serves the i^{th} point, 0 otherwise.

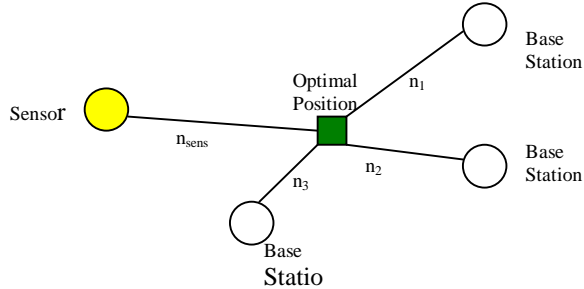


Fig. 4: Scenario for $M = 4$. The three base-stations are getting served by the same

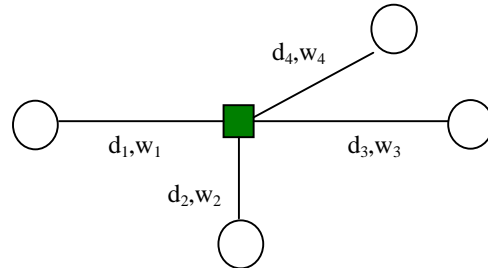


Fig. 5: Corresponding geometric problem for $N=4$.

This is very similar to the Minimum K -median problem, which can be stated as follows: Given n points, we must select K of these to be centers (facilities), and then assign each input point j to the selected center that is closest to it. If a location j is assigned to a center i , we incur a cost $d_j \cdot w_{ij}$. The goal is to select the K centers so as to minimize the sum of the assignment costs.

A hierarchical form of the above problem can now be constructed. The locations of the base stations in the single level problem formulation are replaced with the locations of the 2^{nd} -level copies. It is desired to iteratively find the location of each copy such that communication is minimized. There has been a number of approximation algorithms developed for this NP-hard problem in metric space [10] [11]. The problem complexity makes it infeasible to find the optimal location for our copies. Also, the sensor nodes are not powerful enough to use any of the existing ρ -approximation algorithms [10]. Hence, our algorithm does not attempt to provide a solution within a factor ρ of the optimal. When a level i copy needs to create level- $(i+1)$ copies, we use a simple distributed heuristic to do so. The total cost incurred is the sum over all cost of all the levels. The main advantage of our scheme is that it does not require centralized computation. Thus

using a simple heuristic at each level to spawn off the next level copies, we can get a hierarchical structure of copies in an incremental manner, even though the placement of copies at each level might not be optimal.

2.2. A note on Data Aggregation

In a sensor network, a single sensor value may not be representative of the actual environmental value. For example, a single temperature sensor value might be erroneous due to hardware malfunction. In other cases, single attributes like temperature or light reading might not be of interest to the application at the base-station. Rather, an aggregated value of different sensor types might be of interest. In these cases, in-network aggregation of sensor values takes place using some type of group communication protocol. These communications take place over a very small region of the network (typically within one or two radio hops). The amount of overhead incurred in this communication is negligible compared to the cost of communicating the aggregated values to the base-stations.

Our data placement algorithm is fully scalable to work with these aggregated data, and it can also work with single sensor values, if the application so requires. In general it is oblivious to the kind of data of which it is making copies, which is controlled by the application. A direct benefit of this approach is that it does not place any constraint on the type of data aggregation protocol used. So, for our purposes we assume the presence of a data aggregation protocol that combines the raw individual sensor data to obtain information that is of interest to the applications.

3. Data Placement

For our algorithm to work, the individual base-stations need to know their request rates. A good way of doing this would be to compute the inverse of the inter-request times, or it could be explicitly specified by the application. Finding the rate at which the sensed data changes is a trickier job. We use an approximation that calculates the rate to be equal to the inverse of the mean of last five inter-update times.

3.1 Data Structures

Each data placement middleware at each base-stations maintain a redirect table, which contains mapping from query locations to copy locations. It gets updated each time any data is received. Each time a query is generated, the redirect table is looked up to check if a valid mapping from the query location to a copy location exists. It needs to be noted here that the locations are not node id's, but Cartesian coordinates. At the sensor nodes the middleware maintains a copy table in which it keeps track of all the copies of its generated data. At each node the middleware maintains a cache where each cache entry consists of a data section and the location from which the data originated, and a pointer to a copy table (possibly empty). Fig. 6 shows the schematic for the data placement middleware.

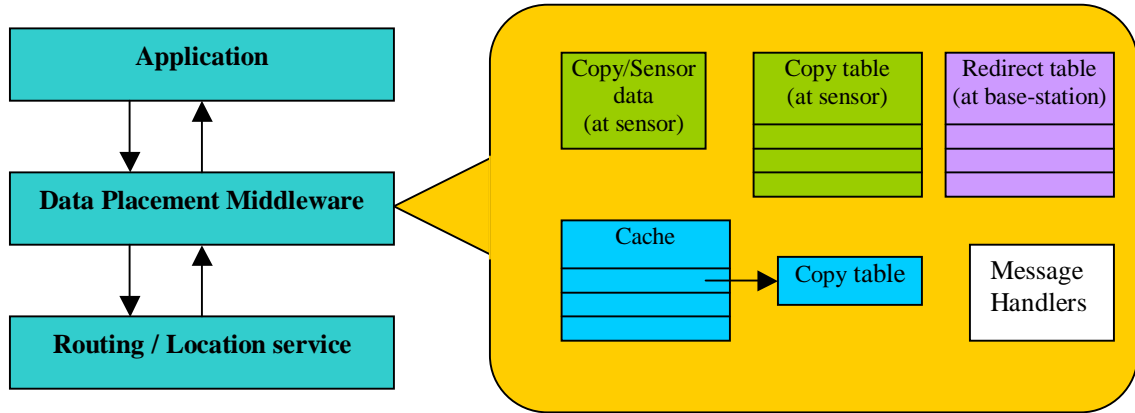


Fig 6: Schematic for the data placement middleware.

3.2 The Algorithm

When a base station, at location BS_I , makes a query for certain data at a location, L , the query gets forwarded to the sensor node that can best serve its request. The rate of request of the base-station (R_I) gets forwarded with the query. The sensor node receives the query and checks if $R_I > R_{update}$. If not it simply sends the data to BS_I . Otherwise it calculates the position for a copy (called the primary copy) of the data, C , by the following formula:

$$C = (R_I \cdot BS_I + R_{update} \cdot L) / (R_I + R_{update}) \quad \dots\dots\dots(3)$$

It also attaches a weight to the copy (*wght*) which is equal to $(R_I + R_{update})$. It then sends the copy to the node, N_I , which is nearest to the location C , along with a forwarding pointer to BS_I . The sensor maintains a record of the location of this copy and its weight in its copy table. When N_I receives the copy, it puts it in its cache, thus creating a primary copy, and sends a data packet to BS_I . When BS_I receives a data packet, it adds an entry to its redirect table. If an entry for L already exists but maps to a different copy location, it gets overwritten. Next time the base-station wants to query location L , it first looks up in its redirect table and if it finds an entry for L , the query is forwarded to the copy location instead.

Now if another base-station, at location BS_2 , sends a query to location L , with a request rate R_2 , the sensor node first determines if it should create another primary copy for the new base-station or serve it from the existing copies. It simply checks in its copy table to check if there is any copy that is nearer to BS_2 than itself. If it doesn't find any such primary copy, it creates another primary copy using the process described earlier. Suppose it determines that the primary copy at location C is closest to BS_2 . To accommodate the new request, it moves the copy to a new location, C' , which is nearer to BS_2 , using the formula:

$$C' = (wght \cdot C + R_2 \cdot BS_2) / (wght + R_2) \quad \dots\dots\dots(4)$$

where, *wght* is the weight of the primary copy. It also increments the weight of the primary copy by R_2 . The sensor node now sends a move request to the node having the current primary copy, N_I , asking it to forward the copy to the new location and serve the base-station BS_2 . It also updates the location of this copy in its copy table. When node N_I receives the move request, it sends a copy of the data from its cache to the node, N_2 , which is closest to the location C' . It then invalidates its own copy and inserts a forwarding pointer to N_2 . When the base-station at location BS_I redirects it query to N_I , it gets redirected to N_2 . When N_2 receives the copy, it puts it in its cache, and sends a data packet to BS_2 . Now the primary copy resides at N_2 . When the base-station at BS_2 receives a data packet, it updates its redirect table. This process goes on for any number of base-stations requesting data from that location.

We have used a heuristic that places the copy at a location which places the copy at a location which divides the line joining the current position of the copy and the

requesting base-station (with rate R) in the ratio $R:wght$. The reason behind using this using this simple Cartesian heuristic is that it is simple and can be computed in a distributed and incremental fashion. This is a greedy heuristic because at any step, we always have a location for the copy, irrespective of however many base-stations get served from the copy. As can be seen from *Fig. 7*, this heuristic performs quite close to the optimal for $K=4$, where K is the number of copies produced. We calculate the optimal using a exhaustive state-space search. The optimal placement is an NP-hard problem in itself[18], so our algorithm performs quite well given the constraints and also the fact that it is calculated in a distributed manner and no global topology knowledge is required. For most cases, our heuristic does not spawn off more than four copies from a single copy. The cost function used here is the one described in (2). Thus our heuristic for a single level performs quite well. Using this heuristic in a cascaded manner leads to the formation of a tree of copies.

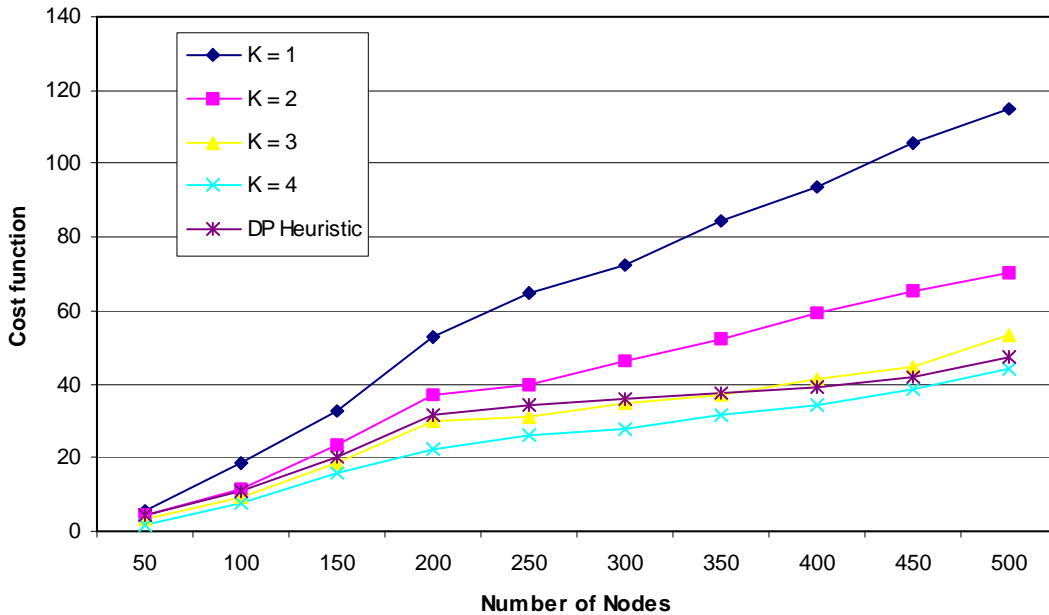


Fig 7: Cost of optimal placement for various values of K and cost of our data placement heuristic.

When the environment is generally idle, i.e., the update rate of the sensor is much lower than the request rates, it becomes profitable to make more copies of the data. So, another level of copies is created. Suppose base-stations BS_3 , BS_4 and BS_5 , having request

rates R_3 , R_4 and R_5 , are being served from the primary copy at N_3 . If the sensor update rate, R_{update} is less than $R_3/2$ then another copy (called a level-2 copy) is created for BS_3 by N_3 in the same way as the primary copy was created using formula (3). (In general a level- $(i+1)$ copy is created by the level- i copy if the request rate R_k is greater than $(i+1) * R_{update}$). The location of this level-2 copy is maintained in the copy table associated with this copy. Similarly, requests from BS_4 and BS_5 are handled the same way if R_4 and R_5 are individually greater than $2R_{update}$; by creating a new level-2 copy for them or serving them from an existing level-2 copy by moving it. However, if R_4 is less than $2R_{update}$, then it is served from the primary copy itself. This process continues in an iterative manner, so that n levels of copies are created, where $n = \lfloor R_{max} / R_{update} \rfloor$ (R_{max} is the maximum request among the base-stations). In general, a base-station with request rate R_i is served by a i^{th} level copy where $i = \lfloor R_i / R_{update} \rfloor$. Thus, a tree of copies is formed originating at the sensor. In some sense, the depth of the tree, n , is determined by how volatile the environment is.

Now when the sensor at location L detects a change in sensed data, it sends update messages to all the primary copies, whose locations are maintained in its copy table. These update messages also contain the new update rate, R_{update}^{new} and the fresh data. When the nodes with the primary copies receive the update messages, they update the corresponding entries from their respective caches. If there are any child copies (i.e., level-2 copies), then the update messages are forwarded to them as well, which then forwards it to its children and so on. When a level- i copy receives an update message with the new update rate, R_{update}^{new} , it updates itself if $i > R_{update}^{new} / R_{max}$, where R_{max} is the highest request rate in the subtree of that copy. Otherwise, it invalidates the copy and all the copies in its subtree. When a base-station sends a redirect query to any of these nodes with invalid copies, they get back a failure notice. Upon receiving a failure notice, a base-station removes the corresponding entry from its redirect table and proceeds to query the sensor at the original location directly and the whole process repeats until the base-station is finally served from a copy from the correct level.

One thing to note is that the primary copies are basically the level-1 copies. If the sensor update rate is greater than the request rate for any base-station, then no copies would be created and all base-stations would get served by the sensor itself. So, we have

a situation where in a very volatile environment prevents creation of copies and an idle environment leads to the spawning of a lot of copies. In other words, the copies consolidate in volatile environments and replicates in an idle environment.

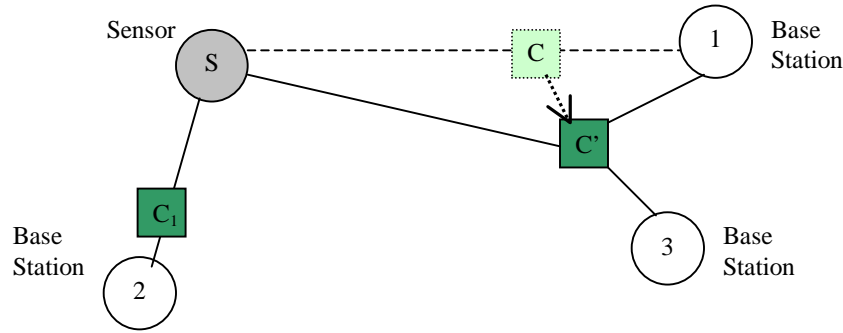


Fig. 8: A typical scenario while using data placement.

Fig. 8 illustrates the working of this algorithm for the case when the rates of requests of the base-stations are greater than the update rate of the sensor R_{update} but less than $2 R_{update}$. Base-station 1 requests data from the sensor, S . Sensor node, S , creates a copy C and places it using the formula (3). Base-station 1 now gets served from this copy. Base-station 2 then makes the same request to S . S then decides to create another copy C_1 , because copy C is further away from 2 than S . When base-station 3 makes the same request to S , S finds that copy C is closer to 3 than itself. So it sends a move request to the node holding C and creates a new copy, C' , using formula (4). Base-station 3 then gets served by C' . When base-station 1 makes a redirect query for C , it gets forwarded to C' , and upon getting data from C' , base-station 1 updates its redirect table. When sensor data at S changes, it just has to send invalidation packets to C' and C_1 .

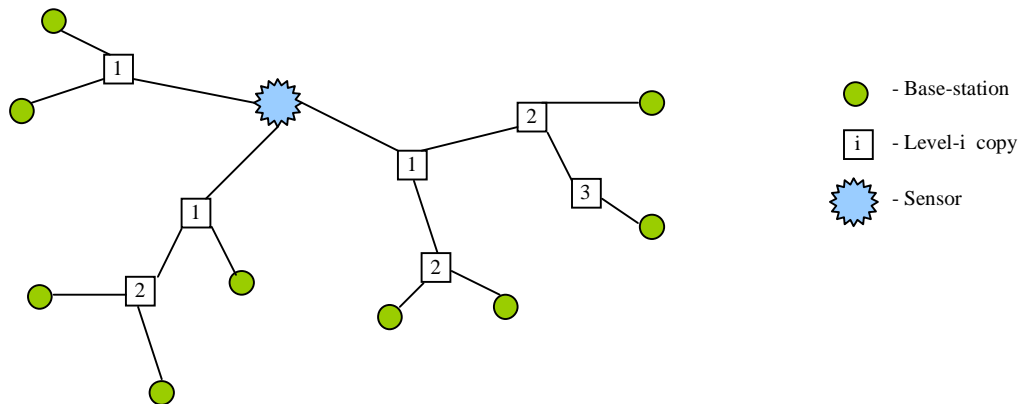


Fig 9: Levels of copies created using data placement.

Fig. 9 shows a more general situation. In this situation three base-stations are served from level-1 copies, five base-stations from level-2 copies and one from a level-3 copy. This implies that the base-station served from the level-3 copy has the highest request rate and it is at least three times the update rate of the sensor.

This algorithm is very scalable because any time a new base-station requests the same data, it gets redirected to an appropriate copy or a new copy is created such that the communication is minimized. One point to note is that using this heuristic, no copy shall spawn off more than eight copies. This limits the amount of state that needs to be maintained. Response time is reduced because the copies are much closer to the base-stations than the original data. Having the data closer to the base-station and preventing flooding of the network whenever the data becomes invalid reduce communication overhead. Consistency of the data served to the base-stations remains the same because the copies are invalidated as soon as the data changes. Since the copies are well spaced out around the sensor node, these invalidation messages have little probability of colliding with each other. Also, this data placement is independent of the routing scheme used.

Fig. 10 shows the important message handlers that are used at a node A having the level- i copy. The original sensor node is considered to be a level-0 copy. Fig. 11 shows the additional message handlers present at the base-stations.

```

Query from B: if( copy not present)
    send Failure_Notice to B
    if(query_rate > update_rate * (i+1))
        if( there exists a level-(i+1) copy nearer to B than A)
            /* serve from existing copy*/
            send Move_Request to appropriate sub-copy using formula (4)
        else
            create a level-(i+1) copy using (3)
    else
        send data to B
Move_request: get move destination C from message
    send Put_in_cache message to C
    invalidate its copy and maintain a forwarding pointer deleted it after timeout
Put_in_cache: put copy in cache
    end data to base-station B (the one which prompted the Put_in_cache message)
Update: if( new_update_rate > (max_request_rate in subtree) * i)
    invalidate copy
    else
        update copy in cache
    send update message to all its subcopies

```

Fig 10: Message handlers at the data placement layer

<i>Data:</i>	<i>Update redirect table</i> <i>Send data to application</i>
<i>Receive query:</i> <i>(from application)</i>	<i>if(entry exists in redirect table)</i> <i>send query to the redirect location</i> <i>else</i> <i>send query to original location</i>
<i>Failure_notice:</i>	<i>remove entry from redirect table</i> <i>send query to original location</i>
<i>On_timeout:</i>	<i>remove entry from redirect table</i> <i>send query to original location</i>

Fig 11: Additional message handlers at the base-stations

3.3 Data Placement API

Between the data placement middleware and the application:

```
bool SubmitRequest ( type , location , rate );
interrupt DATA ( type , location , value);
```

The method *SubmitRequest* sends the application request to the middleware and it specifies the type and the location of the data it requires. It may also specify the rate at which it requires the data. When a certain data is retrieved by the data placement algorithm, it sends a software interrupt to the application layer, which then invokes its receive function to process the data.

Between the data placement middleware and the routing layer:

```
void recv ( Packet * pkt );
bool send ( Packet * pkt , destination);
```

This is self-explanatory; the *recv* and *send* functions handle the receiving and sending of packets from and to the routing layer respectively.

Between the data placement middleware and the location service:

```
location getLocation ( );
```

This method returns the location of the node.

4. Target Platform

We plan to implement our data placement middleware on the Berkeley motes[16], which runs a microthreaded operating system TinyOS[16]. Each node has up to three sensors embedded on it. It runs on a 8-bit 4 MHz micro-controller and has 8KB of

program memory and 512 bytes of data memory. TinyOS, itself takes about 180 bytes of data memory, and around 4KB of program memory. Future versions of the motes (due January) shall have about 10 times the memory resources. However, experiments on the test bed suffer from some drawbacks. Firstly, the number of motes available to us is only about 20, so we shall not be able to conduct experiments on a large scale. Also, the energy consumption of the motes cannot be measured as it runs from standard 1.5 V AA batteries. These constraints motivate our simulations, where we can measure the energy consumption at nodes, as well test for significantly larger scales. However, we intend to validate the working of our middleware and measure the number of packet transmissions as part of our experiment.

5. Evaluation

We 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 as expected in a realistic sensor network. Our simulation model is a network of N ($100 < N < 500$) 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. Since we have to model a location-aware network, we assume that each node knows its own location. We did not want to have any specialized routing protocol as well as maintain non-unique (location dependent) id's, so we implemented a wrapper, which works on top of any routing protocol and translates the destination location of packets into actual destination node addresses. Certain hot spots in the network are generated at random, and the base-stations send periodic queries to these hot spots. The rate of update of data in these hot spots is varied according to the experiment. The energy consumption is measured in terms of Joules per node per flow. Each value is taken as the average over five simulations.

First we looked at the performance of our middleware over different routing algorithms. *Fig. 12* shows the performance of data placement and a simple query response scenario for two different routing protocols: AODV and DSDV. The request rates are about five times the update rate for this experiment. We vary the number of nodes in the network over a grid of 200m x 200m. The graph proves that DSDV is not a

good choice for this type of network model. The primary reason is that DSDV is a proactive routing protocol, and periodically updates routing tables. This requires a substantial amount of memory to maintain the routing tables, as well as overhead to compute routes that are never used. On the other hand, reactive algorithms like AODV compute and cache routes on-demand. So the overhead of this type of algorithm is low for a network with a limited number of flows. Hence for further simulations, we use AODV as our routing algorithm.

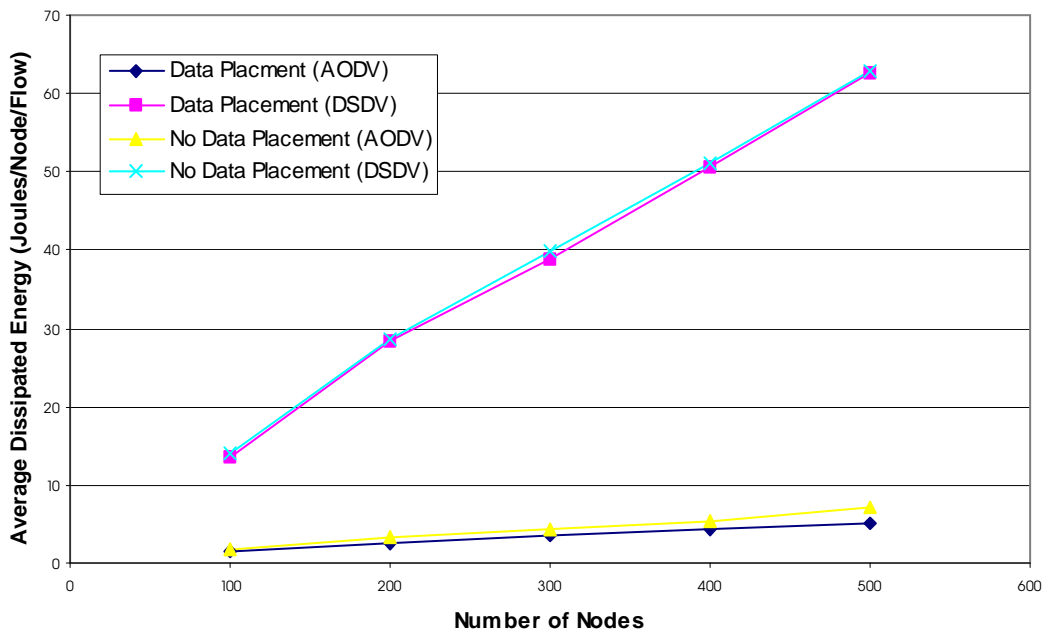


Fig 12. Average dissipated energy using AODV and DSDV

Next, we consider the performance of the data placement middleware against two baselines. First, a simple query-response model with no data placement and second, caching at the base-stations. In the second case, when an update occurs, the update message is flooded through the network. For this experiment, the request rates are about five times the update rate for this experiment. We vary the number of nodes in the network over a grid of 200m x 200m. As we can see from *Fig. 13*, the second case performs very badly due to the collisions caused by the update messages flooding through the networks. The data placement middleware reduces energy conservation by

about 35% for a network of 100 nodes and more than 50% for a network of 500 nodes, as compared to a simple query-response scenario.

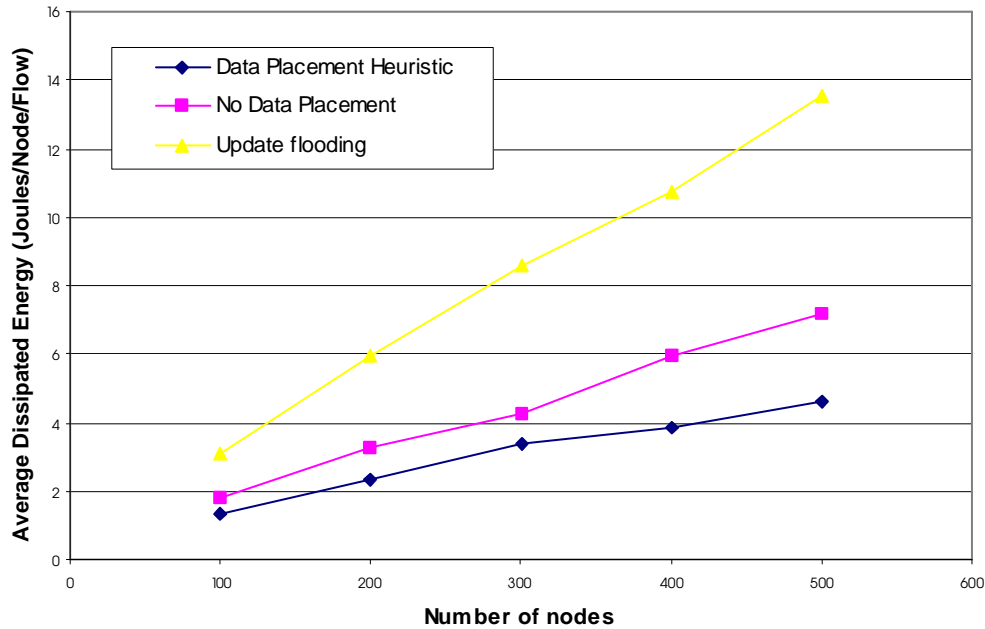


Fig 13. Average dissipated energy using three approaches

A main feature of the data placement algorithm is that it is adaptive and sensitive to changes in sensor update rates. When the sensor update rates are low, more replicas are created and when the rate is high, the copies are consolidated. In general a tree of copies is built starting at the sensor, and the depth of the tree is inversely proportional to the sensor update rate. Now if the update rate were lower than the request rates, it would obviously be beneficial to have the base-stations keep copies with themselves, and then to send the update messages to only these base-stations. These would be infeasible in a real sensor network, because a sensor node would then have to keep track of all the base-stations it is serving, which may be beyond its memory capacity. This method is sometimes referred to as *omniscient multicast*. But, this provides an optimal baseline for the case when the update rate is lower than the request rates. However if the update rate is more than the request rates, then this model would be wasteful because the base-stations would be receiving update messages more often than it would be requesting the data. In this situation, it is more beneficial to have no copies, and revert to a simple query response model. Our data placement algorithm provides a smooth transition between

these two models as the update rate changes. *Fig. 14* shows the performance of these three models over a network of 400 nodes in a 200m x 200m grid each having a radio range of 20m. The sensor update rate is normalized against the average request rate.

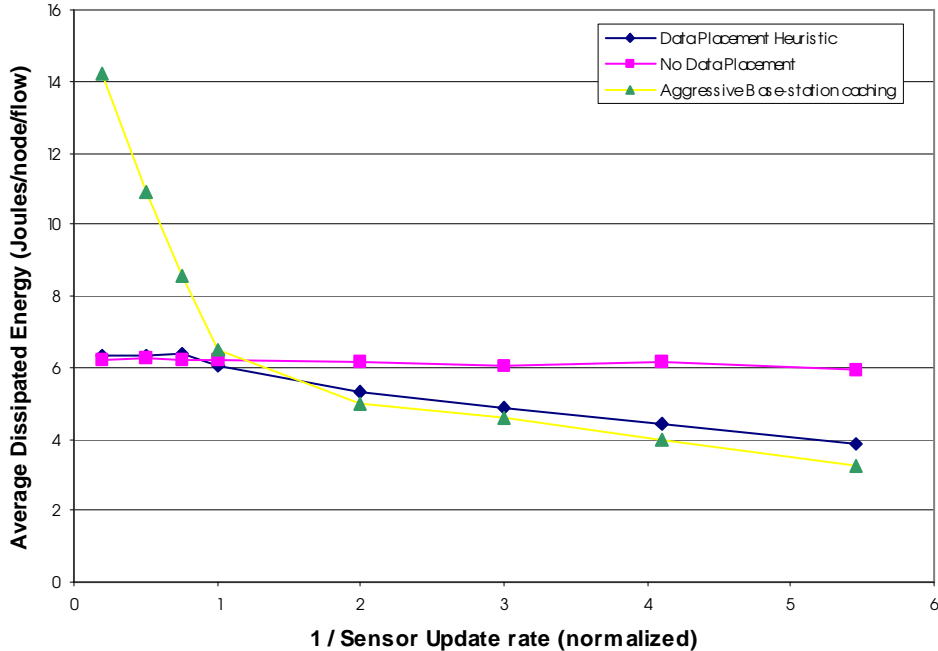


Fig 14. Average dissipated energy for the three models plotted against sensor update rate.

As can be seen from the figure, when the Sensor Update Rate is high (i.e., $1/\text{Sensor Update Rate} < 1$), then the data placement algorithm reverts back to the simple query response model, but when the network is more idle, it achieves results which are quite close to the optimal. As can be seen from the figure, our data placement middleware adapts the amount of replication to be done according to the rate of updates and hence we get a performance that is almost as good as *omniscient multicast* (aggressive base-station caching) when the update rate is low, but it does not suffer from the degradation in performance when the sensor update rate is high.

Till now we have tested for cases when the sensor update rates are almost constant. Now we test the performance when the update rate changes drastically over short periods of time. We use the update traffic pattern shown in *Fig 15*. The sensor switches between high and low update rates with a period T . At high update times, the update rate is made

to be a Poisson distribution with mean of 0.5 times the maximum request, and at low update times the update rate is made to be a Poisson distribution with mean of 3 times the request rate.

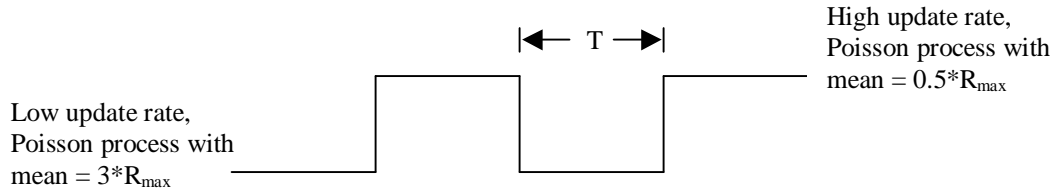


Fig 15: Update pattern for experiment.

Fig. 16 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 no caching and omniscient multicast, because it adapts to both suit both the high and low update rates. As the 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, and since the update rate keeps changing rapidly, there are more failures and hence, more communication. However, this scenario does not seem very likely to occur in a sensor network.

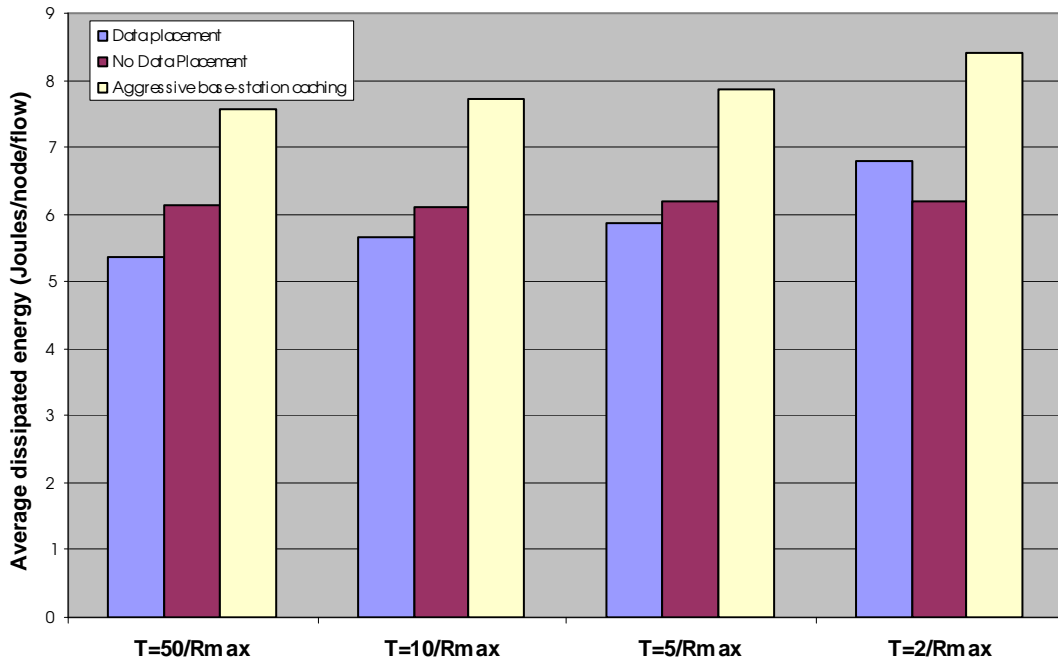


Fig 16: Average dissipated energy for the three models for different values of T

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.

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 in developing new paradigms and services for sensor networks, taking into account the unique features of these networks. Distributed sensor networks are a step towards the overall vision of ubiquitous computing. 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 does not make any 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], have done some work in location estimation using beacons. The accuracy of the location algorithm has a direct impact on the accuracy of data gathered, as also the amount of benefit of data placement. A point to note is that, while in some cases inaccuracies in location shall cause the copy locations to be further away from optimal, the same inaccuracy could provide better results than expected. This is a challenging area, because of the arbitrary structures of the network, and the fact that the computational constraints of the nodes make complicated algorithms difficult.

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. [13] and Imielinski, Goel [14] proposed addressing by geographical location and attributes. However most routing protocols cannot support these types of addressing schemes. 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.

Routing protocols for traditional ad-hoc wireless networks are primarily of two types: proactive (eg., DSDV) and reactive(eg., DSR, AODV). For the purposes of sensor networks the reactive approach seems more appropriate as discussed earlier, but they all have the basic assumption of fixed addressing. Geographical forwarding is a promising candidate, but fails in sparse regions of the network. Xu, Heidemann, 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.

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.

However our approach differs from traditional multicast in the sense that it works on top of the network layer, whereas multicast routing [24][25] takes place at the network layer. Application-layer multicast derives support from the network layer, which leads to assumptions about the network layer itself. Our approach is free from such assumptions. Finally, the depth of our tree is determined by the update and the request rates, and it adapts itself to minimize the communication. As our simulation results show, a fixed tree structure gives very poor performance when update rate becomes very high.

Data placement is 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 [22] 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 [23] 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.

The problem is somewhat reminiscent of data placement in distributed shared memory systems[21]. 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.

Several hardware models of sensor nodes have been developed, primary among them being the Rockwell WINS nodes [17] and the Berkeley motes [16]. The Berkeley motes (still at a development stage) currently run on a 4 MHz microcontroller and have 8 KB of program memory and 512 bytes of data memory. It runs on a micro-threaded operating system called TinyOS, which takes approximately 200 bytes of data memory. It has a short-range radio and a packet size of around 30 bytes. This goes to show that services for sensor networks have to be very lightweight.

Keeping all these related work in consideration, we attempted to make our data placement framework as general as possible, so that as new technology evolves, they can be rapidly integrated with the data placement service.

On a theoretical side, our problem is reducible to the Minimum K -median problem which has been proven to be NP-hard [18], and at best a ρ -approximation algorithm is possible in polynomial time, and that too with involved centralized computation. So it is impossible to guarantee optimality using a simple distributed heuristic. Hence we focus more on trying to attain a good improvement over certain baselines.

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 so as to minimize the communication overhead of data transfer. Our data placement algorithm adapts the number and location of the copies according to the frequency of updates in the network. It creates copies more aggressively when the updates are infrequent, and the number of copies is reduced as the environment becomes more dynamic and the update rates increase.

Our algorithm achieves all the goals it set out to achieve. It reduces communication and energy consumption. It reduces query response time by placing copies closer to the base-stations. The algorithm is completely distributed and requires very little local processing and the amount of bookkeeping involved is small, which fits in nicely with the constraint of limited resources. Also it does not make any assumptions about the

underlying MAC or routing layers, though as shown in section 4.2 pro-active routing algorithms like DSDV are not a good choice for these types of networks.

Data placement is a new and unique approach for energy conservation in wireless sensor networks. This makes use of the location-aware nature of the nodes and leverages this benefit to overcome other constraints. To our knowledge, no previous work has been done to apply data placement to a location-aware network.

However, as of now our data placement algorithm is only best effort. Further work needs to be done to introduce guarantees into the data placement model. In the big picture, data placement would act as a service that would aid in providing guarantees to applications running on these sensor networks. This is analogous to how web caching helps in providing guarantees over the internet.

7. References

1. [1] Lili Qiu, Venkata Padmanaban, Geoffrey M Voelker, On the Placement of Web Server Replicas, Proc. IEEE INFOCOMM 2001.
2. [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. [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. [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. [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. [6] Radhika Nagpal, Organizing a Global Coordinate System from Local Information on an Amorphous Computer, MIT AI Memo 1666, August 1999

7. [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. [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. [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. [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. [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. [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. [13] 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.
14. [14] 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.
15. [15] 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.
16. [16] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, " System architecture directions for network sensors," ASPLOS 2000.
17. [17] Development platform for self-organizing wireless sensor networks, Proc. SPIE, Unattended Ground Sensor Technologies and Applications, Vol. 3713, p. 257-268.

18. [18] A Compendium of NP optimization problems
<http://www.nada.kth.se/viggo/problrmlist/compendium.html>
19. [19] Self-organizing distributed sensor networks, Proc. SPIE, Unattended Ground Sensor Technologies and Applications Vol. 3713, p. 229-237.
20. [20] The ns-2 simulator. <http://www.isi.edu/nsnam>.
21. [21] Distributed Operating systems. Andrew S. Tanenbaum Prentice Hall.
22. [22] Guillaume Pierre, Maarten van Steen, and Andrew S. Tanenbaum, *Self-replicating Web documents*, Technical Report IR-486, Vrije Universiteit, Amsterdam, February 2001,
23. [23] The Case for Geographical Push Caching. *Proceedings of the Fifth Annual Workshop on Hot Operating Systems*, Orcas Island, WA, May 1995
24. [24] 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
25. [25] A Survey of Multicast Technologies (2000) Vincent Roca, Luís Costa, Rolland Vida, Anca Dracinschi, Serge Fdida September 2000.