A Mobility Management Framework for Optimizing the Trajectory of a Mobile Base-station

Madhu Mudigonda[1], Trisul Kanipakam[1], Adam Dutko[1], Manohar Bathula[1], Nigamanth Sridhar[1], Srinivasan Seetharaman[2], Jason O. Hallstrom[3]

[1] Electrical & Computer Engineering, Cleveland State University
[2] Deutsche Telekom Labs
[3] School of Computing, Clemson University

Abstract. We describe a software framework for prescribing the trajectory path of a mobile sink in a wireless sensor network under an extensible set of optimization criteria. The framework relies on an integrated mobility manager that continuously advises the sink using application-specific network statistics. We focus on a reference implementation for TinyOS. Through extensive physical experimentation, we show that the mobility manager significantly improves network performance under a range of optimization scenarios.

1 Introduction

Wireless sensor networks afford the promise of ultra-dense instrumentation of the natural and built environment for purposes of observation and control. For these networks to become integrated as part of a permanent planetary monitoring fabric, network longevity obstacles must be overcome. Several methods have been proposed to extend the lifetime of multi-hop networks, including the use of multiple sinks, improved sensor distribution, energy-balanced clustering, data mules, and other strategies. In this paper, we focus on the use of *mobile base-stations*, which periodically alter the routing structure to avoid the possibility of static bottlenecks. While introducing a mobile sink is not always possible, a number of existing applications inherently rely on mobile sinks.

Sink mobility introduces a number of questions: When should a sink move? Where should it move? How is its location shared with the routing network? How are multi-hop routes maintained in the presence of mobility? Our objective is to design a generalized framework that provides answers to these questions. To our knowledge, we are the first to consider an intelligent mobile sink that requires no a priori scheduling. Instead, a distributed mobility manager collects salient network statistics and uses these statistics to drive mobility decisions. While others have considered the problem of sink mobility in sensor networks, all but a few [7, 9] have relied exclusively on simulations. In contrast, we have developed an integrated hardware/software testbed to conduct experiments. The testbed includes a programmable mobile element and a network of 30 TelosB nodes.

We report three contributions: *(Sects. 2 and 4)* A generalized framework and reference architecture to support mobility decisions in sensor networks. *(Sect. 3)* A collection of decision metrics to support sink mobility, including (*i*) residual node energy, (*ii*) regional network congestion, and (*iii*) average relay distance. We also describe how new metrics are readily integrated into the framework. *(Sect. 5)* A prototype implementation of the framework for TinyOS.

Problem Definition. We consider a sensor network consisting of an arbitrary number of nodes, deployed arbitrarily. Each node participates in a spanning tree routing layer rooted at a base-station (sink) and communicates sampled data over this layer. Hence, the load on each device comprises (*i*) sampling sensors, (*ii*) transmitting sampled data toward the sink, (*iii*) relaying received data toward the sink, and (*iv*) updating the routing path when needed (e.g., , due to node death). For nodes closest to the base-station, (*iii*) quickly becomes the dominating factor; a small set of nodes must relay *all* sampled network data, creating a lifetime bottleneck.

While there is substantial prior work focused on introducing sink mobility to mitigate this bottleneck (detailed in Sect. 7), questions concerning the trajectory of the sink have not been completely addressed. Examples of simplifying assumptions adopted in prior work include time invariance of battery and radio performance, uniform radio propagation range, and independence of MAC delays on energy consumption [7, 11, 21]. In contrast, our work makes only two basic assumptions: The target network must provide time synchronization, and the geographic extent of the network must be known to the mobile sink.

While the most important mobility goal is ensuring uniform energy consumption across the network, other optimization metrics are also possible. A base-station might, for instance, attempt to locate itself to reduce network congestion or to overhear an interesting data stream. Hence, we define the problem as follows: **the design of a generalized framework for prescribing a sink mobility path under an extensible set of optimization criteria, in a manner responsive to real-time network conditions.**

2 System Architecture

The architecture is illustrated in Fig. 1; the corresponding component interfaces, expressed in nesC, are shown in Fig. 2.

Metric Generator. An implementation of the MetricGenerator interface runs on each node, monitoring its runtime behavior and recording a set of statistics material to the relevant optimization goal. For example, ResidualEnergyMonitor, discussed later, is a specialization of MetricGenerator that computes a real-time estimate of available device energy. This value (obtained through a call to getCurrentValue()) is then transmitted over the routing layer to the sink to support its mobility decision. In many cases, the metric data can be piggybacked on standard application data packets. In Sect. 3, we describe three different metrics for which we have built generators. The descriptions there roughly correspond to implementations of getCurrentValue().

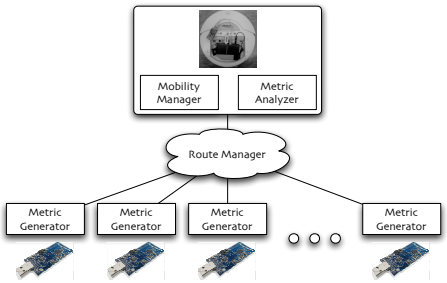


Fig. 1. Framework Architecture

```
interface MetricGenerator {
  command int getCurrentValue();
}
interface MetricAnalyzer {
  command int compare(int m1, int m2);
}
interface RouteManager {
  command void formRoute();
  command void cancelRoute();
  command int getParent();
  command int getDistance();
}
interface MobilityManager {
  command bool isMoveNecessary();
  command int getNewRegion();
  command void moveToRegion(int target);
}
```

Fig. 2. Framework Interfaces

Metric Analyzer. The sink must be able to impose an ordering on this data for purposes of comparison. In the case of residual energy, the sink should move to the region where the average residual energy is *highest*, whereas in the case of network congestion, the sink should move to the region where congestion is *lowest*. To support relative valuation, an implementation of the MetricAnalyzer interface is used on the mobile device. The interface provides a compare() operation that returns 1 if m1 is *better* than m2, -1 if m2 is better than m1, and 0 if the values are indistinguishable — where the definition of *better* depends on the optimization goal. The mobility manager relies on MetricAnalyzer to determine when the local-area average of collected metric values dictates a move, and to perform pairwise area comparisons when determining where to move.

Route Manager. When the sink moves from one position to another, the routing topology must be updated to ensure node-to-sink connectivity. This process is supported by an implementation of the RouteManager interface, used both at the sensing end-points and the sink. When a move begins, the sink invokes cancelRoute() to initiate route cancelation. When the move is complete, formRoute() is invoked to reestablish the routing tree. Each end-point records its position using two state variables, the node's parent and distance from the root. RouteManager is intentionally general. Several strategies have been proposed to dynamically modify the structure of a routing tree [2, 4, 11, 12, 18]. Any of these may be used to implement the RouteManager interface.

Mobility Manager. An implementation of MobilityManager is used by the sink to signal the need for movement and identify the new target location. The sink periodically polls for a movement signal using isMoveNecessary(). If a true response is received, getNewRegion() is used to compute the new sink location. This decision is based on a pairwise comparison of the metric values associated with all movement alternatives (using MetricGenerator and MetricAnalyzer). Finally, moveToRegion() directs the mobile platform to its new location.

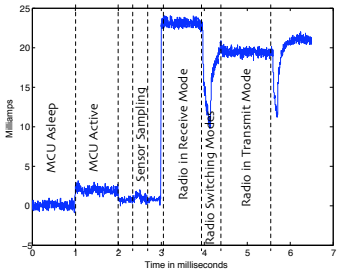


Fig. 3. TelosB Current Draw

Term	Notation	Current
Radio in transmit mode	I_{Tx}	17.4675 mA
Radio in receive mode	I_{Rx}	21.0675 mA
MCU active	$I_{McuActive}$	1.9325 mA
MCU sleep	$I_{McuSleep}$	0.0437 mA
Sensor use	I_{Sensor}	0.0061 mA

Fig. 4. Tmote Sky Current Draw

3 Decision Metrics

Residual Energy. The first decision point we consider relates directly to network longevity — the residual energy available to each device. We adopt the *Credit-Point* (CREP) system [23] for estimating residual energy: $E_{max} = V_b \times I_b \times 3600$ *Joules*. Here V_b and I_b correspond to battery voltage and capacity, respectively. Borrowing from [23], for a pair of AA batteries, the capacity is 2.2 A-Hr, with an effective voltage of 3V; we can compute the maximum energy as $E_{max} = 3 \times 2.2 \times 3600 = 23760$ J. Based on this initial energy budget, the CREP system deducts "energy points" from E_{max} for each action on the device.

The CREP model assumes a constant battery voltage, which is invalid for most, if not all, battery chemistries. To lift this assumption, we periodically sample the battery voltage using `VoltageC`, provided by TinyOS. The results are used to compute the energy consumed during a given period based on (1), substituting the actual battery voltage for V_b. To deduct "energy points", the monitor records the actions performed by a node during each activity period, along with the duration of each activity, focusing only on the most energy-intensive activities. For example, the monitor records the time the microcontroller was active, the time the radio spent in transmit and receive modes, the particular sensors activated, etc. Energy points are assigned empirically by measuring the current draw during each activity/configuration (a priori). A sample current plot is shown in Fig. 3; mean values are used as energy points, as summarized in Table 4. Using these values, energy points are deducted for each activity performed. No additional hardware is required to support this metric.

Network Congestion. The second metric is designed to minimize network congestion, and thus avoid message loss and energy depletion through retransmissions. To compute the level of network congestion in a given region, we aggregate multiple measures. Specifically, our congestion monitor measures the average packet reception rate (PRR), the average received signal strength (RSSI), and the average link quality indicator (LQI) along all incoming links at each node. In many cases, PRR can be measured directly. In particular, if messages are transmitted at a specific period in a given application, receiving nodes can measure the packet reception rate directly. Alternatively, in [16], the authors study the correlation

between RSSI and PRR. They conclude that when RSSI values are higher than the radio sensitivity threshold (about −90dBm for the CC2420 radio), they are strongly correlated with the packet reception rate.

It is important to note that radio-level metrics, like RSSI and LQI, provide information beyond what can be gleaned from the packet reception rate alone. RSSI, for instance, is useful in measuring the noise floor within the vicinity of a node. A high noise floor increases the likelihood of congestion(-like scenarios). Bluetooth devices, when operated near a CC2420 radio, can cause the noise floor to rise as high as −25dBm. Providing access to this information enables the mobile sink to avoid such regions during intermittent periods of interference.

Average Distance to Sink. The final metric is designed to reduce the average number of transmission hops. If, for instance, the base-station is located far from the principal data source (which may vary over time), the sink should relocate closer to the source to reduce the workload on intermediate nodes. The implementation approach is straightforward. It assumes that each data packet is tagged with the distance-to-root value maintained by the publishing device. This information is then used by each node to compute a moving average over the distances of the nodes contained within its respective routing subtree.

4 Managing Mobility

Discovery Phase. We assume that the mobile base-station has no a priori knowledge of the network deployment. Instead, the sink uses the assumed network extent information to divide the deployment area into a regular grid of a desired granularity. It then proceeds to tour the region, using a beacon-based discovery process to define the membership of each grid cell. If a node is "contained" in multiple cells, it is assigned to the cell that offers the best link quality. This discovery phase may be repeated if network characteristics are fundamentally changed — if, for instance, nodes are inserted or removed.

Mobility Management Phase. Throughout the life of the network, the mobility manager is responsible for signaling the need for sink movement and determining the new target location. But the travel trajectory is not without constraints.

How far can the sink go? The maximum travel distance in a single move is limited by the duty cycle of the application. More specifically, the best performance will be achieved if the time required to complete a move is less than the application reporting period. Otherwise, the movement of the base-station will interfere with data collection (since the routing tree will be in transition). More important, stationary nodes are required to remain active during the transition to ensure route reconstruction. Hence, long moves can degrade system performance and limit the lifetime of the network.

To limit the reach of the sink, the mobility manager estimates the travel time associated with each path candidate based on the speed of the mobile device. The manager culls candidate paths that require travel time significantly beyond

procedure $MobilityManager$
  While collecting metrics:
    if ($alarmExpires$) **then call** $MetricAnalyzer$
end $MobilityManager$
procedure $MetricAnalyzer$
  Calculate average metric for $R_{current}$
  if $\texttt{compare}(avg\_metric(R_{current}), threshold) < 0$ **then call** $Move$
end $MetricAnalyzer$
procedure $Move$
  $R_{new} := (r \in \mathcal{R} : (\forall r' \in \mathcal{R}_{reach} : \texttt{compare}(avg\_metric(r), avg\_metric(r')) \geq 0)$
  Calculate route from $R_{current}$ to R_{new}
  Broadcast *cancel_route* message
  Perform move
  Advertise current location to initiate route reconstruction
end $Move$

Fig. 5. Mobility Management Algorithm

that of the application reporting period. This includes subtracting, from the allowable time, the time spent ranking the target candidates based on collected metric data, as well as the estimated time to reconstruct the routing tree.

Where should the sink go? We now consider the details of the mobility algorithm summarized in Fig. 5. While the sink is stationary, the mobility manager receives a continuous stream of metric data from across the network. At a fixed period, an *analysis alarm* is signaled to activate the metric analyzer. To synchronize sink movement with the application sleep cycle, the alarm period is a multiple of the application sleep period. The analyzer in turn compares the average of the decision metrics from its current region ($R_{current}$) to its movement threshold and determines whether to move.

If a move is required, the sink identifies the best target location (R_{new}) by comparing the metric averages of all regions within its single-step reach (R_{reach}). This comparison is realized using an implementation of compare() appropriate to the desired optimization goal. It next computes the route to the new location and broadcasts a *cancel_route* message to alert the network that it has become unrooted. The stationary nodes wait in an active state during the move and wait for the base-station's location advertisement to begin route reconstruction.

The metric analyzer may require history data to achieve optimal performance. If the analyzer simply selects the best target location within its reach during each step, it could become trapped within a set of local maxima/minima. A more sophisticated analyzer can overcome this problem. Given that the analyzer has access to network-wide metric data, it can compute the optimal target location and factor this information into the movement decision in each step. Hence, the analyzer can select, in each step, the region within its reach that brings the sink closer to the network-wide optimum.

(a) Testbed Layout (b) Testbed Realization (c) Mobility Platform

Fig. 6. Testbed Infrastructure

5 Evaluation and Results

We developed a prototype implementation for TinyOS 2.x and conducted extensive experimental studies using a physical testbed. Given that our primary optimization goal was network lifetime, we compared the lifetime benefits achieved using our sink manager to other mobility management strategies.

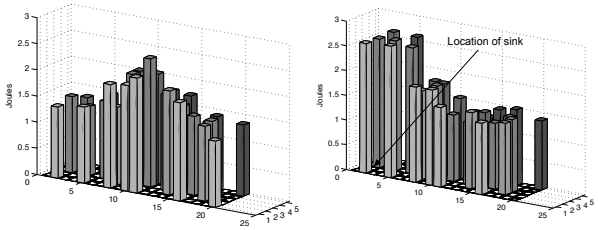
Testbed Infrastructure. Our studies were conducted using a testbed of 30 ceiling-mounted *Tmote Sky* nodes placed in a pseudo-random fashion throughout a 1500 sq. ft. (60' x 25') area, as shown in Figs. 6(a) and 6(b). To ensure multi-hop connectivity, radio power was reduced, and each node was required to select a parent outside its grid cell. The motes were powered using standard consumer batteries; each experiment used a fresh set. The base-station uses a Tmote Sky linked to a *Linksys NSLU2* device. The NSLU2 is in turn connected to an *iRobot Create*^{TM}, shown in Fig. 6(c). As shown in Fig. 6(a), the testbed is partitioned into 5 regions. The sink's reach is limited to one region transition per step.

5.1 Energy Consumption and Lifetime

We characterize the network lifetime increase enabled by a sink under four mobility strategies: (*a*) no movement, (*b*) a fixed arbitrary path, (*c*) a fixed path computed using a linear programming formulation, and (*d*) a dynamic path guided by our mobility manager. The experiments were conducted using an application with a sleep period of 10 minutes. Each time a node wakes, it samples its local sensors and transmits the sampled data to its parent. It then waits to receive and relay each data point from its subtree before returning to sleep.

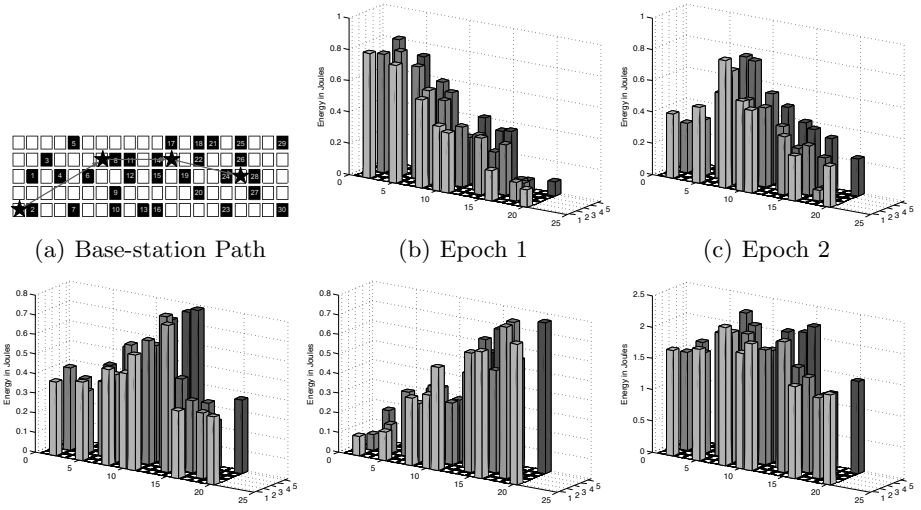
Static Base-Station. We ran two experiments with a static sink. The two were identical, except for the sink location. In the first run, the base-station was placed near the center of the network. In the second, it was placed near one end of the network. Each run was executed for a duration of ten hours. We measured the residual energy of each node before and after each run. Figs. 7(a) and 7(b), summarize the energy consumed by each node in the network, in Joules.

As shown in Fig. 7(a), nodes near the center of the network consumed considerably more energy than nodes near the edges of the network. The situation in the second experiment is similar, except that the peak is shifted to the network's



(a) Sink (near the middle) (b) Sink (near one end)

Fig. 7. Energy Usage (static base-station)



(a) Base-station Path (b) Epoch 1 (c) Epoch 2

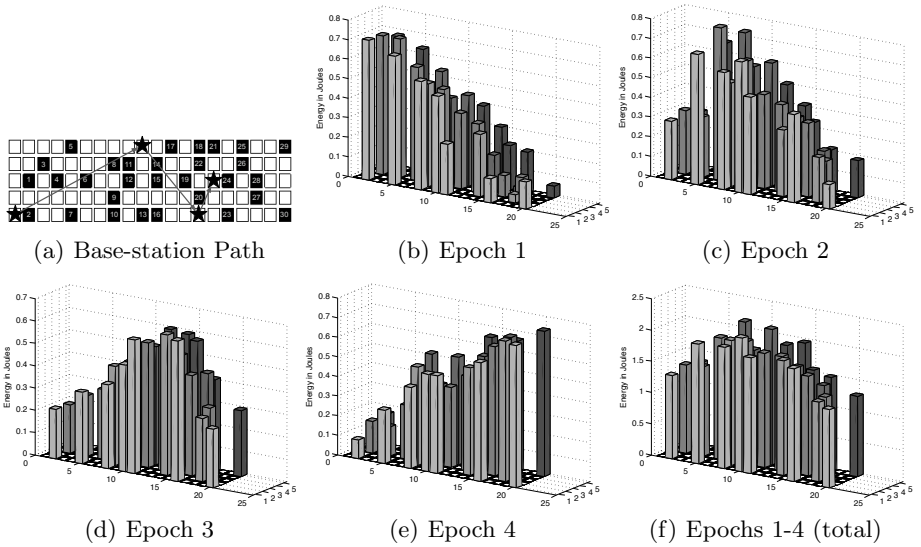
(d) Epoch 3 (e) Epoch 4 (f) Epochs 1-4 (total)

Fig. 8. Energy Usage (mobile, arbitrary path)

edge, as shown in Fig. 7(b). The results are not surprising: The activity period of nodes closer to the base-station is longer since these nodes must forward all of the messages from their respective subtrees. Since the base-station is static, nodes near the base-station consume more energy over the run.

Mobile Base-Station on Arbitrary Path. We next consider a mobile base-station that traverses a pre-determined path spanning each of the regions within the deployment area. The device remains stationary at four points for an epoch of 2.5 hours each. The results are summarized in Fig. 8.

As the sink moves from region to region, the energy consumption trend is clearly visible in Figs. 8(b)–8(e). The combined total consumption is shown in Fig. 8(f). Although the consumption pattern is not completely uniform, it is improved over the static case. Comparing Figs. 8(f) and 7(a), the average difference between the maximum and minimum consumption across nodes is



(a) Base-station Path (b) Epoch 1 (c) Epoch 2

(d) Epoch 3 (e) Epoch 4 (f) Epochs 1-4 (total)

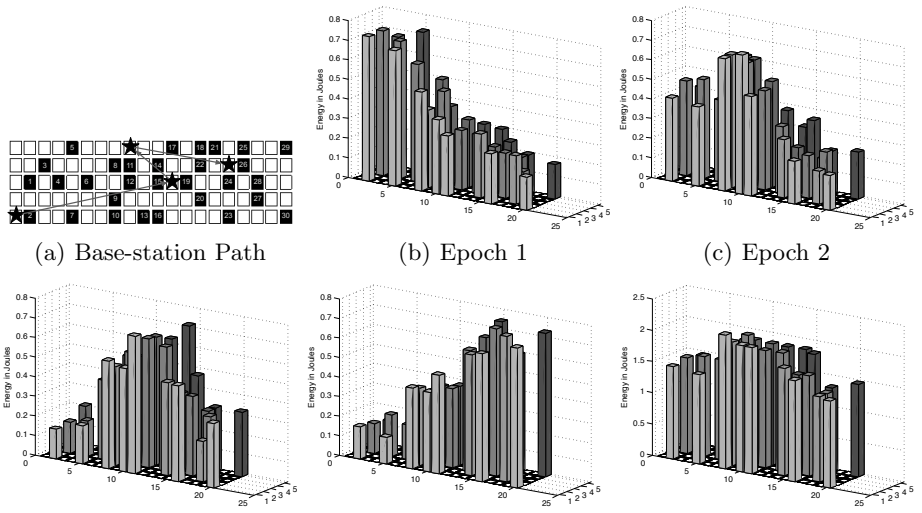
Fig. 9. Energy Usage (mobile, linear programming)

approximately $1.5J$ in the static sink case, and approximately $1J$ in the mobile case. While the result is far from optimal, it demonstrates the lifetime extension opportunities afforded by a mobile sink. The limiting factor is that the movement path ignores the residual energy available across the network. It offers no way to adapt to consumption conditions.

Mobile Sink on Pre-Computed Path using LP. Prior work in route management relies on linear programming (LP) to derive an optimal path for the mobile sink [3]. An important characteristic of the LP approach is an underlying assumption that radio behavior is time-invariant. This enables pre-calculation of the complete base-station route and corresponding sojourn times based on the transmission range of each sensor and the sink, respectively. We applied a standard LP formulation to our setup to obtain a pre-calculated sink path and compared the performance of the LP approach to our algorithm. Here we describe the important elements of the LP formulation, beginning with two possible objective functions:

1. *maximize* $\{\ min_{Node_i}\ \{\textit{residual energy at node } i\}\ \}$
2. *maximize* $\{\ \sum_{Site_k}\ \{\textit{sojourn time at site } k\}\ \}$

We adopt the first objective function because the depletion rate of each mote is insignificant compared to its initial energy. Hence, the second objective function would be too time-consuming to experiment with. By contrast, for the first objective function, the residual energy of each mote can be computed as the difference between the initial energy and the energy consumed during each sojourn period of the mobile sink. Thus, given the energy consumed at each node while the sink is at a location k, it is easy to compute the optimal sink path.



(a) Base-station Path (b) Epoch 1 (c) Epoch 2

(d) Epoch 3 (e) Epoch 4 (f) Epochs 1-4 (total)

Fig. 10. Energy Usage (mobile, mobility manager)

However, computing the energy consumed at each sensor node is non-trivial. The residual energy available at a given device depends on its data relay activity. Simultaneously solving for the optimal inter-node routes and the sink traversal path is generally not possible in a linear program. Hence, we partition the program into two components and run them serially. The behavior of each node is assumed to be time-invariant; we pre-compute the optimal multi-hop routes from each node to every potential sink location. For each sink site k, determining the multi-hop route that yields the least energy consumption involves solving a supplementary LP with an objective of *minimize* { *maximum* {*energy consumed at each node i when sink is at site k*} }. On solving the supplementary LP and passing the output to the primary residual energy maximization LP, we obtain the desired route, which is optimal when radio behavior is time-invariant. Note that the ordering of the target locations is only influenced by the reach constraint imposed on the sink in each step. Thus, using the observed energy depletion rate and the reach information in our testbed, we compute the LP-based sink route.

Figure 9 shows the results across four epochs, each of 2.5 hours in duration, when the sink uses this computed path. Figures 9(b)–9(e) show the energy consumed in each epoch; Fig. 9(f) shows the total energy consumed.

Mobile Base-Station on Dynamic Path. Finally, we study the performance of the mobility management framework. In this set of experiments, the base-station is deployed at an arbitrary initial location without a pre-defined traversal path. The mobility management framework is used to collect residual energy data and to inform the sink's trajectory to maximize residual energy across the network. Every hour, the sink determines whether the regional resid-

Sink Location	Mean (J)	Std Deviation (J)
Static (Middle)	1.62	0.4393
Static (End)	1.75	0.5981
Mobile (Arbitrary)	1.56	0.3270
Mobile (LP-computed)	1.49	0.2330
Mobile (Dynamic)	*1.43*	*0.1329*

Table 1. Standard Deviation of Power Consumption

ual energy has fallen below the required threshold. If so, the mobile sink initiates a move. It first computes the average amount of residual energy available in each region within its reach. The sink then determines the new target location and directs the robot to move to the corresponding coordinate location.

Again, the experiment was executed for ten hours. Note that it does not matter where the sink is placed since the mobility manager uses network measurements to guide its path. The experiment resulted in a total of three moves, dividing the run into four epochs. The results are summarized in Fig. 10. It is important to emphasize the low degree of variability across the network.

Comparing Mobility Strategies. To quantify the uniformity of energy consumption across mobility strategies, we compare standard deviations in Table 1. When the static sink is placed at the center of the network, the standard deviation in residual energy is low. Perhaps surprisingly, the opposite is true when the sink is placed at one end of the testbed. The explanation is straightforward: The nodes in the testbed form their routing paths based on network connectivity. When the static sink is placed at one end of the network, the nodes that are at the other end are at a distance of five hops from the sink. However, when the sink is at the center, the nodes that are furthest away are only three hops; the resulting difference in load is smaller.

The standard deviation in energy consumption is smallest in the case of the dynamically computed path. This shows that our mobility decision engine is effective in achieving the intended goal — ensuring that all nodes in a sensor network deplete their energy reserves at approximately equal rates. Further, the mean energy consumption is also reduced when the sink is moved along the path computed by our mobility manager.

Effective Lifetime. The goal of engineering an intelligent sink is to extend effective network lifetime. Figure 11 shows a comparison of expected node lifetimes using various mobility management strategies. These are estimates of how long each device will stay active based on battery capacity and average consumption rates; network dynamics will obviously play a role in determining the actual lifetimes. The arbitrary path strategy does a poor job of ensuring uniformity. As expected, the LP-computed path reduces variability, as does our dynamic strategy. Notice, however, the expected lifetime of the first node to die in each of these cases; there is a significant difference. With a pre-defined path, the first node dies in 42 hours, whereas with the LP-computed path, the first node dies at 63 hours. Using our mobility manager, the first node dies after 68 hours.

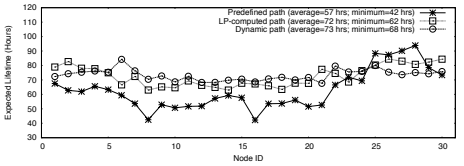


Fig. 11. Lifetime Extensions across Strategies

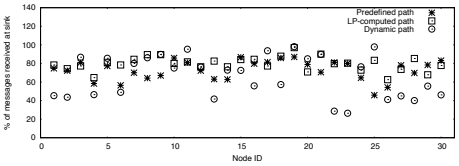


Fig. 12. Network Throughput

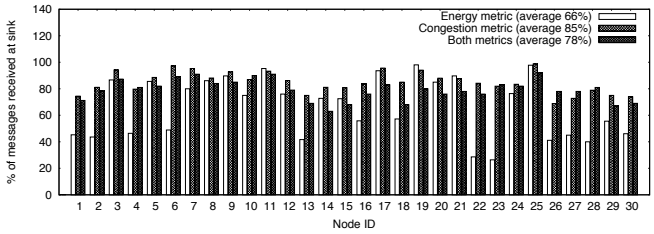


Fig. 13. Network Throughput (lifetime, congestion)

Mobility Overhead. There is a cost associated with each transition. When the sink moves from one location to another, routes from the stationary nodes to the sink must be recomputed. In our case, we use a naïve strategy, canceling all routes and reforming the tree. Over the experiments we ran, we observed that the energy cost for the stationary nodes is about 0.1 J for each sink transition. Further, our protocol introduces an extra cost for sending residual energy data (4 bytes per message). The overhead is justified by the significant benefit that changes in the network topology can offer.

Throughput. Finally, we consider the throughput of the network as a validation measure to ensure that the application is functioning properly. While our initial experiments exhibited poor yield, we were able to improve the yield using a supplementary radio layer focused on reducing link quality variation among nodes [6]. Figure 12 shows that most nodes deliver 60% to 80% of their messages.

5.2 Network Congestion

We also studied the impact of the network congestion decision metric. The experimental setup (length, application) was the same, except that the mobility manager used NetworkCongestionMonitor as the metric generator. In Fig. 13, we compare the average throughput from each node when using the network congestion metric as compared to the residual energy metric. When the sink makes mobility decisions based on network congestion, the average throughput increases from 66% to 85%. Note that we did not consider network longevity, focusing exclusively on improving network throughput.

We next repeated the experiments, applying both metrics. The primary metric was network congestion; residual energy was used as a secondary metric in

the event of ties. In this case, the average throughput was approximately 79%, and the lifetime of the first node to die was 59 hours. In the future, we expect to further investigate simultaneous optimization strategies using multiple metrics.

6 Discussion

Extending to Other Metrics. The prototype implementation of the mobility management framework demonstrates the utility of our approach in managing a mobile sink. Using this implementation, we have conducted significant evaluation studies to verify that the approach achieves uniform energy consumption across the network. The primary extension point for using the implementation in settings other than a testbed environment is the use of other decision metric components. Although we have not presented complete evaluation results here (primarily for lack of space), we have experimented with this possibility using the network congestion monitor and the message distance monitor. The benefit afforded by these metrics is easy to see: If the sink is located in a congested area, the throughput of the network is bound to suffer; and if a majority of messages must travel a long distance, throughput is again going to be affected.

Our mobility management framework can be viewed as a harness for developing specific strategies for controlling a mobile sink. For example, in [9], the authors use a mobile sink to collect data from a sensor network with the aim of reducing the number of multi-hop data transmissions. The sink can travel at varying speeds depending upon how much data it can buffer. This system can be implemented using our framework by designing a metric analyzer component sensitive to buffer size, which determines both a target speed and location for the sink. One of the directions for our future development is to develop a generic testbed that can be available for testing such mobility protocols.

Sophistication in Metric Analysis. A potential problem with performing simple pair-wise analysis in the metric analyzer is network partitioning. Consider a scenario in which the sink is in a given region of the network, and a small number of nodes bridge its current reach with the rest of the network. If these nodes die, the sink may deplete the regions in its current reach and declare the lifetime of the sensor network to be over. With a more careful analysis, the sink can detect that the throughput has dropped in an unpredictable manner. This recognition can trigger another discovery traversal to rebuild the sink's view of the available nodes.

As another example, we observe that there is a correlation between the in-degree of the sink and network congestion. The larger the number of adjacent (gateway) nodes, the higher the level of congestion and associated retransmissions near the sink. Thus, it may be in the best interest of both the sink and the network to limit the number of gateways it associates with. However, if the number of gateways is too small, they will become overloaded by the network traffic being routed to the sink. Hence, it may be valuable to limit the *betweenness* of the gateway nodes (i.e., the number of multi-hop routes passing through each node) [10]. Our framework is capable of accounting for such constraints.

Algorithm	Algorithm execution	Metric	Platform mobility	Metric generation
Joint Mobility and Routing [11]	Offline	Load distribution	Low	Simulated Model
Greedy Maximum Residual Energy (GMRE) [3]	Online	Residual energy	Variable	Simulated Model
Adaptive Sink Mobility [19]	Online	Data events	Variable	Observed
Deterministic/Random Walk Models [5]	None	N/A	Pre-defined	N/A
Our Dynamic Mobility Management Framework	*Online*	*Parameterizable: Residual energy, network congestion, distance to sink*	*Variable depending on network conditions*	*Observed*

Table 2. Comparison of Related Schemes for Managing Mobile Sinks

7 Related Work

The literature is rich with work focused on extending the lifetime of sensor networks [13, 14, 17, 24]. Here we present an overview of related research and identify the novelty of our contributions (summary in Table 2).

Mathematical Sink Trajectory Models. Luo et al. [11] show, using a mathematical model, that when nodes are distributed according to a Poisson distribution within a circle, the (near-) optimal mobility strategy is for the sink to travel along the periphery of the circle. They argue that using such a path will improve lifetime by approximately 500% over using a static sink. They also propose an algorithm for routing to the sink. As the sink moves along the network boundary, message flows from the network "follow" the sink.

Wang et al. [21] present an LP formulation for determining an optimal sink trajectory parameterized by *sojourn time*. Their model makes several simplifying assumptions, limiting its use. Basagni et al. [3] improve upon the LP model presented in [21] by lifting some of these assumptions. They also present a new path planning scheme, *Greedy Maximum Residual Energy*: rather than pre-computing the sojourn times at different nodes in the network, the mobile sink greedily selects a neighboring node as its new location based on residual energy.

Efficient Message Routing Protocols. Baruah et al. [2] present an approach to maintaining node-to-sink routes within a network. They do not address the problem of determining an optimal mobility pattern, instead focusing on how to maintain usable data routes. Urgaonkar et al. [18] present an approach that allows static nodes to learn a sink's mobility pattern. Some nodes (*"moles"*) statistically characterize the sink's (random) movements using a probability distribution function; the result is used to inform message forwarding decisions.

Luo et al. [12] extend MintRoute [22] by adding steps to account for (*i*) link breakage, (*ii*) topological changes, and (*iii*) packet loss. They test their routing protocol using TOSSIM and illustrate the energy savings associated with using a mobile, time-synchronized sink. Chakrabarti et al. [4] use a pre-defined path to bring the mobile sink close to each node in the network to minimize long-range radio transmissions, thus reducing transmission energy.

Data Ferrying/Relaying. Jea et al. [8] present a load-balanced data collection algorithm that uses multiple mobile elements (*"mules"*) to collect data. The authors provide evidence supporting the use of large storage buffers on both the static nodes and mules to ensure data integrity. Shah et al. [15] provide insight into buffer requirements for nodes and mules within a sensor network and the effects of various buffer sizes on transmission success rates.

Somasundara et al. [1] and Kansal et al. [9] prove that mobile-sink-based sensor networks transmit fewer packets of data when compared to their static counterparts. They propose a model for calculating a data-complete trajectory for a mobile sink. Wang et al. [20] discuss two alternatives to using a mobile sink for sensor network lifetime extension in cases where utilizing a mobile sink might be infeasible. The first method uses extra static nodes near the sink. These extra nodes act like *"sleeper"* nodes within the network, and sleep for long durations, but once other nodes around the gateway begin to fail, they wake up to continue servicing the network. The second method uses additional resource-rich nodes near the sink, which together provide a rudimentary load balancing service by distributing the workload of the other gateway nodes.

8 Conclusion

We presented the design of an extensible, generalized framework for managing the trajectory of a mobile sink within a static sensor network. The framework supports objective-specific mobility decision metrics, based on which the path of a mobile sink can be computed dynamically. The framework does not require any a priori information about the sensor network, except for the extent of the area it covers. The framework periodically calculates the *quality* of each region within the network using dynamic measurements, based on an optimization-specific notion of *quality* (e.g., maximum residual node energy). This global portrait is used to drive sink mobility decisions without any assumptions of constancy or uniformity in radio reach or power consumption.

We also presented a reference implementation of the framework for TinyOS, with an emphasis on decision metrics aimed at ensuring uniform energy consumption across devices. The goal was to maximize network longevity. We evaluated this implementation using experiments on a testbed of TelosB motes. The dynamic base-station path computed by our mobility manager achieved a lower mean and standard deviation in energy consumption (1.43J and 0.13J) compared to an arbitrary path (1.56J and 0.33J), and a path computed using a typical LP formulation (1.49J and 0.23J). This means that the rate of consumption was more uniform, and consequently, the effective lifetime of the network was longer. Although the energy savings look modest, the lifetime extension was substantial.

Acknowledgments. This work was supported in part by the National Science Foundation (CNS-0746632, CNS-0745846). The authors gratefully acknowledge the NSF for its support.

References

1. A. Somasundara et al. Controllably mobile infrastructure for low energy embedded networks. *IEEE Trans. on Mobile Computing*, 05(8):958–973, 2006.
2. P. Baruah et al. Learning-enforced time domain routing to mobile sinks in wireless sensor fields. In *LCN '04*, pages 525–532, Washington, DC, USA, 2004. IEEE.
3. S. Basagni et al. Controlled sink mobility for prolonging wireless sensor networks lifetime. *Wireless Networks*, 14(6):831–858, December 2008.
4. Chakrabarti et al. Communication power optimization in a sensor network with a path-constrained mobile observer. *ACM TOSN*, 2(3):297–324, 2006.
5. I. Chatzigiannakis et al. Sink mobility protocols for data collection in wireless sensor networks. In *MobiWac '06*, pages 52–59, New York, NY, USA, 2006. ACM.
6. A. R. Dalton et al. Reducing the impact of link quality variation in embedded wireless networks. *Int. J. of Ad Hoc & Sensor Networks (AHSWN)*.
7. E. Ekici, Y. Gu, and D. Bozdag. Mobility-based communication in wireless sensor networks. *IEEE Communications Magazine*, 44(6):56–62, jul 2006.
8. D. Jea et al. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *DCOSS'05*, pages 244–257, June 2005. Springer.
9. A. Kansal et al. Intelligent fluid infrastructure for embedded networks. In *MobiSys '04*, pages 111–124, New York, NY, USA, 2004. ACM.
10. L.C. Freeman et al. Centrality in Valued Graphs: A Measure of Betweeness Based on Network Flow. *Social Networks*, 13(141):141–154, 1991.
11. J. Luo and J.-P. Hubaux. Joint mobility and routing for lifetime elongation in wireless sensor networks. In *INFOCOM'05*, pages 1735–1746, New York, 2005.
12. J. Luo et al. Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In *DCOSS '06*, pages 480–497, Berlin/Heidelberg, 2006. Springer.
13. I. Papadimitriou and L. Georgiadis. Maximum lifetime routing to mobile sink in wireless sensor networks. In *Proc. IEEE SoftCOM*, New York, September 2005.
14. S. Jain et al. Exploiting mobility for energy efficient data collection in wireless sensor networks. *Mob. Netw. Appl.*, 11(3):327–339, 2006.
15. R. Shah et al. Data mules: modeling a three-tier architecture for sparse sensor networks. In *WSNA'03*, pages 30–41, New York, 11 May 2003. IEEE Press.
16. K. Srinivasan and P. Levis. RSSI is under appreciated. In *EmNets'06*, Boston, MA, May'06.
17. L. Tong, Q. Zhao, and S. Adireddy. Sensor networks with mobile agents. In *MILCOM 2003*, pages 688–693, New York, October 2003. IEEE.
18. R. Urgaonkar and B. Krishnamachari. FLOW: An efficient forwarding scheme to mobile sink in wireless sensor networks. In *SECON '04*, Washington, DC, 2004.
19. Z. Vincze et al. Adaptive sink mobility in event-driven multi-hop wireless sensor networks. In *InterSense '06*, page 13, New York, NY, USA, 2006. ACM.
20. W. Wang et al. Using mobile relays to prolong the lifetime of wireless sensor networks. In *MobiCom '05*, pages 270–283, New York, NY, USA, 2005. ACM.
21. Z. M. Wang et al. Exploiting sink mobility for maximizing sensor networks lifetime. In *HICSS '05*, page 287.1, Washington, DC, USA, 2005. IEEE Computer Society.
22. A. Woo et al. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03*, pages 14–27, New York, NY, USA, 2003. ACM.
23. O. Younis and S. Fahmy. An experimental study of routing and data aggregation in sensor networks. In *LOCAN '05*, Washington, DC, USA, November 2005.
24. W. Zhao et al. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc '04*, pages 187–198, New York, NY, USA, 2004. ACM.