Learning-Enforced Time Domain Routing to Mobile Sinks in Wireless Sensor Fields

Pritam Baruah[§] Rahul Urgaonkar[†]
Bhaskar Krishnamachari[†§]
Departments of Computer Science[§] and Electrical Engineering-Systems[†]
University of Southern California
Los Angeles, CA 90089
baruah, urgaonka, bkrishna@usc.edu

Abstract

We propose a learning-based approach to efficiently and reliably route data to a mobile sink in a wireless sensor field. Specifically, we consider a mobile sink that does not know when to query or does not need to query. Furthermore, the sink moves in a certain pattern within the sensor field. Such a sink passively listens for incoming data that distant source sensors unilaterally push towards it. Unlike traditional routing mechanisms, our technique takes the time-domain explicitly into account, with each node involved making the decision "at this time what is the best way to forward the packet to the sink?". In the presented scheme, ***moles*** *(nodes in the vicinity of the sink) learn its movement pattern over time and statistically characterize it as a probability distribution function. Having obtained this information at the moles, our scheme uses reinforcement learning to locate the sink efficiently at any point of time.*

1. Introduction

The classical view of a wireless sensor network involves one or more stationary base-stations/sinks gathering data from a network of sensors. These sinks are usually multiple hops away from the sources of data and this gives rise to one of the fundamental operations in the network - routing of data to the sink. Routing schemes for wireless sensor networks need to consider four important factors - energy cost, robustness, throughput and delay. Most current solutions try to strike a balance between all the four factors but they are designed mainly for stationary sinks. We consider the case where a sink moves within the sensor field. We assume that there is an inherent pattern to the sink's movement. This pattern might change in time but it is assumed that every new pattern is sustained for a significant period of time. This pattern could be characterized by temporary local variability but the approach presented accounts for them. An application-oriented assumption is that the sink does not know when to query and so it does not issue queries. Instead, it listens passively for data pushed by source sensors towards it. Such situations are representative of a class of real-life situations that are characterized by non-requirement of disseminating queries and uninformed mobile sinks. The example below illustrates such situations.

Consider a forest patrol that makes periodic patrols along certain paths in the forest. These patrols aim at preventing poaching and monitoring endangered wildlife. However their efficacy is limited because they cannot cover the entire area assigned to them. This problem can be solved to an extent by deploying a wireless sensor network that senses unusual events in the area (incoming poachers or endangered animals). Information about these events needs to get routed to the moving patrol. The naive way of doing it is to route to a stationary base-station at the periphery of the network and then directly transmit to the sink. However this is not always desirable because of energy and delay constraints. Another method is to flood the network but this can result in unacceptable wastage of energy.

Instead, a better approach is to route data to the patrol (mobile sink) while it is *on the move*. Note that the patrol does not know when to query. Periodic query might not be feasible due to the energy cost as well as the time lag incurred in receiving information. The main reason for energy inefficiency of periodic querying is the fact that unusual events are not likely to happen very often. In such situations, it is desirable for the mobile sink to passively listen for incoming data from the sources. This is equivalent to source sensors *pushing* data towards the sink. In this case, routing can exploit the fact that patrols move in a certain pattern that can be characterized by some spatio-temporal relation. Over time, the network can learn this pattern and then by adding a temporal dimension to the routing decision, data can be pushed towards the moving patrol with a degree of confidence about its energy efficiency and delivery guarantees.

The approach proposed in this paper, which we call Hybrid Learning-Enforced Time Domain Routing (HLETDR), makes use of the underlying pattern in the sink's movement to discover *good* data delivery paths. We define *moles* to be the nodes that lie in the vicinity of the path that the sink takes. It is assumed that a mole can somehow detect the presence of the sink near itself. Every mole characterizes the sink's presence in its vicinity as a probability distribution over the tour duration of the sink. The objective of HLETDR is to route data towards the sink in an energy efficient and robust way. Here, each node that either generates the data or receives it from a neighboring node, makes a decision depending on the current local time as to which node to forward the data to. We say that these decisions are made in time-domain. Ultimately data reaches a mole which subsequently delivers it to the sink.

Upon receiving data, a mole *reinforces* the route taken depending on a goodness value associated with it. This value is calculated by the mole using probabilistic local information about when the mobile sink is expected to make an appearance in its vicinity. The reinforcement propagates to the source and in the process, sets up gradients that enable a temporal dimension to the routing decision problem. Over time this distributed algorithm is expected to converge to an efficient route/set-of-routes in terms of energy cost and delay.

The remainder of this paper is organized as follows. The related work is presented in section 2. We then present our approach in section 3 which describes the scheme and presents extensive simulations results followed by discussion. And we conclude the paper in section 4.

2. Related Work

The problem of routing to mobile sink been addressed in some recent papers (see [1], [2], [3]). The authors of [1] propose an asynchronous data dissemination protocol SEAD for mobile sinks in a sensor network. SEAD constructs and maintains a data dissemination tree from source nodes to multiple mobile sinks. A sink that wants to join the tree selects one of its neighboring sensor nodes and sends a join query to the source of the tree. The scenario that we are considering is different from this because here, the sink is not explicitly querying for the data. Secondly, none of these schemes uses learning-based techniques to try and learn the movement pattern of the sink.

The authors of [2] propose a three-tier architecture composed of a top tier of WAN connected devices, a middle tier of mobile transport agents (MULES) and a bottom tier made of fixed wireless sensor nodes. The MULES perform a random walk on the sensor field, collecting data from the nodes and delivering it to the sink. The underlying assumption here is that the MULES cover the entire region in a finite time duration, so that data can be delivered to the sink in bounded time delay.

TTDD (Two-Tier Data Dissemination) proposed in [3], aims to provide scalable and efficient data delivery to mobile sinks. Upon detecting a stimulus, each source sensor proactively builds a data dissemination grid and sets up forwarding information at the sensors closest to grid points, called dissemination nodes. A query from a sink traverses two tiers to reach a source, the first being the local grid square of the sink's current location and the second being the dissemination nodes at grid points. TTDD promises to be more energy efficient than several data dissemination protocols such as Directed Diffusion [4], GRAB [6] etc. However, it does not make any assumptions on the mobility pattern of the sink. For the specific scenario that we consider in this paper, we show that our learning-based approach yields much better energy efficiency.

A Reinforcement-based approach to data gathering was proposed in Directed Diffusion [4], a data-centric dissemination scheme primarily meant for stationary sinks. It uses attribute based naming to match data to sensor nodes. It is specially designed to enable in-network data processing. It uses a reinforcement scheme to locally select neighbors from which to receive high-rate data originating somewhere downstream. Initially, the sink floods the query to the entire network. The attributes in the query identify destination sensors. These source sensors on obtaining data, flood the network with low-rate data. This data will reach the sink from multiple neighbors. The sink in turn reinforces the neighbor it wants to receive data from. This reinforcement is recursively applied right down to the source sensors. From then on, high-rate data disseminates along this reinforced path. This adds tremendous robustness to data dissemination because of allowance for negative reinforcement and local repair. The reinforcement decision can be based on observed latency or some form of discretionary priority. Note that Directed Diffusion uses reinforcement to select a preferable path from a set whereas HLETDR uses reinforcement to locate the sink as well as to arrive at an efficient path.

Our reinforcement-based scheme is closest in spirit to [5] which proposes a reinforcement learning-based efficient querying mechanism LEQS for sensor networks. In this scheme, the object being queried for is assumed to have an underlying probabilistic spatial description that is learned over time. It is shown that this in-network learning helps reduce energy expenditure significantly over time. In the scenario considered in our work, the sink has an underlying *spatio-temporal* description.

3. HLETDR: Hybrid Learning-Enforced Time Domain Routing

Consider a random deployment of sensor nodes in an area of interest. The sensor nodes are capable of sensing one or more phenomena and processing this information. The objective of each node is to send this information to a mobile sink over multiple hops. The sink might not be able to cover the entire region because of several limitations. We consider the specific case where there is an inherent pattern to the sink's movement through the sensor field with allowance for some variance. We assume that the sink does not explicitly query for the data collected by the sensor nodes. The problem in this set up can be described as follows:

Is it possible for source sensors to learn an efficient way to route data to the mobile sink in a distributed fashion ?

We show that answer to the above question is yes. More specifically, if there is a subset of nodes through whose vicinity the mobile sink passes periodically (moles), then for each of these moles, we can construct a function that represents the likelihood that the sink is in its vicinity in a given time interval. This information can then be leveraged to find efficient routes to the mobile sink using reinforcement leaning.

We now describe our approach in detail in the following sections.

3.1. Methodology

Every node in the field maintains weights which are used to calculate the forwarding probabilities to each of its neighbors. Every node can be a data source as well as a relay node. The goal is to forward data such that the path leads to a mole in whose vicinity the mobile sink in located. This path should be efficient and also dynamically transmuting in order to account for sink movement and random delays in the network.

In other words, the problem is essentially that of locating the moving sink efficiently even under indeterministic delays and high speed of the sink. This implies that routing decisions based on a global estimate of the sink's position at a given time are likely to fail. Hence, HLETDR makes all decisions locally based on local time. If a node has data that needs to be forwarded, then the probability of selecting a neighbor is determined by the likelihood of this neighbor being on a good path to the sink at that moment of time. Every node involved in successive propagation of data recursively follows this local decision making process at the time instant in which it has to forward data.

Since we assume that the sink has an underlying pattern to its motion, we define a *tour* as the smallest time duration after which the pattern repeats itself. Then the timeline as seen by a node can be divided into chunks (domains). If the tour duration is T time units and there are m time-domains, then each time-domain lasts for $\frac{T}{m}$ time units and the time-line is said to be "m-granular". Note that this granularity determines the state complexity of the routing table at any node. For example, if a node has N neighbors, then the complexity of the routing table is $O(mN)$. Intuitively, there would a trade-off between the accuracy of sink's location information and the time-line granularity.

This division of timeline into domains at the nodes serves three purposes:

1. Eliminates the need for explicit time synchronization: Each node now needs to make a forwarding decision in its local time-domain, which need not be same as others. The only loose requirement in terms of time synchronization is for each node to have a consistent notion of a tour.

2. Makes routing decisions robust to random delays (cumulative transmission delays, MAC contention etc.) in the network: As pointed out earlier, random delays in the network can make global routing decision erroneous. With local time-domains, if a packet gets delayed at a downstream node, then the upstream neighbor selects the best possible forwarding neighbor for the *current* time-domain. In this way time-domain routing can be considered as best effort in time.

3. Helps locate the sink as a function of time by adding a temporal dimension to the routing problem

Let S be the set of neighbors of node n. If $W_n(k, t_i)$ denotes the weight associated with a neighbor k at time-domain t_i, the forwarding probability $P_n(k, t_i)$ for that neighbor at t_i is calculated as follows:

$$P_n(k, t_i) = \frac{W_n(k, t_i)}{\sum_i W_n(i, t_i)} \forall i \in S \quad (1)$$

Initially, each node assigns equal weights to all its neighbors. Data from a source node performs a random walk based on the forwarding probabilities calculated as above. When data finally reaches a mole, the mole reinforces the path taken based on a *goodness value* which is calculated using local information of the sink's location available to the mole.

It is assumed that each mole maintains a mean and variance of the arrival time of the sink in its vicinity. This could be done by calculating simple running averages or by obtaining exponentially weighted running averages which allows us to give more importance to recent events. This becomes important in the scenarios where the mobility pattern of the sink changes to a new one. Because of severe space constraints, maintaining time-series data of sink arrivals at the moles in order to collect a distribution of arrival times is prohibitively expensive. This is the rationale behind maintaining only the mean and variance.

Given this setup, the goodness estimator at every mole assumes the arrival times within a tour to have a Gaussian distribution with a mean and variance as obtained above. When data arrives at a mole, it calculates the goodness value as follows. Suppose μ and σ are the mean and standard deviation associated with the arrival time of the sink at a mole. If the data arrives at the mole at some time t, then the goodness value G is calculated as:

$$G = \int_{t-\beta\sigma}^{t+\beta\sigma} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(\tau-\mu)^2}{2\sigma^2}} d\tau \quad (2)$$

where β is a tunable parameter.

The goodness value G calculated at a mole not only gives the probability of the sink being in its vicinity but also gives the notion of how far away the sink possibly is. Lower the probability, farther away the sink is and vice versa. The tunable parameter β can be selected based on the following criteria. If data arrives exactly at the mean time of arrival of the sink, then the goodness value for the path taken should be very high and yet, not 1 (to account for incidental non-arrival). To give a goodness of about 0.9545 to such a path, β can be taken to be 2. β determines the level of accuracy about the sink's position that is desired at the mole. For example, if the storage capacity at a mole is not a constraint, β can be set to a high value which would imply that data can be stored at the mole until the sink arrives in its vicinity.

Note that if a mole does not have enough storage capacity to hold the expected amount of traffic or the variance in the mobility pattern is very high, mole-to-mole data propagation is performed until a rendezvous point with a sink is reached (upon which the sink receives the data). It might also be needed if the sink is expected to take a very long time to reach that mole and the data has a time constraint attached.

Once calculated, the reinforcement parameter G propagates downstream to the source of the data along the same path taken from source to mole. Along the way, every intermediate node updates the weights for its upstream neighbor based on the reinforcement parameter.

The design of the weight update scheme is critical to the success of HLETDR. Intuitively, optimistic update methods are required because reinforcement is essentially a trial-and-error based search of the solution space. We considered four different update schemes and found that conservative update schemes might not be able to locate the sink at all and if they are able to locate it, the learning process is very long. Thus, in the update schemes that we propose, if the goodness value is high, then the weight of the upstream neighbor is increased aggressively. If it is low, then weights are decreased aggressively.

Specifically, the update process at each node can be described as:

$$\text{Update} = \begin{cases} \text{Perform Negative Reinforcement} & 0 < G < a \\ \text{No Reinforcement} & a \leq G \leq b \\ \text{Perform Positive Reinforcement} & b < G < 1 \end{cases} \tag{3}$$

The thresholds a and b can be selected based on empirical observations. In our simulations we choose a and b after empirically observing that at least one path was able to locate the sink with high probability if multiple flows are initiated in every time-domain. The “No Reinforcement” area has the effect of hysteresis; it is required because the update schemes are designed to be aggressive. Having this region avoids unnecessary crossovers to the opposite update type if G is close to the threshold.

Fig. 1 shows the basic mechanism of HLETDR update process at work at node R for time domain t_i. Data packets originating at Source S initially perform a random walk based on the forwarding probabilities obatained from the weights. When these finally reach a mole (X, Y, Z in fig. 1), goodness values are calculated using the learnt distribution and sent back along the reverse path as shown by the dashed curves. At every intermediate node (such as R in fig. 1), the weights for the neighbors (A, B, C in fig. 1) are modified based on the update scheme. Now we describe the update schemes considered.

3.2. Update Schemes

1. *Additive Increase Additive Decrease (AIAD)*:
   This is a conservative scheme where a constant value (equal to the initial weight W_0) is added to the current weight to get the updated weight under positive reinforcement and vice versa. Specifically, updated weights are given by:

$$W_{\text{new}} = \begin{cases} W_{\text{old}} + W_0 & \text{Positive Reinforcement} \\ W_{\text{old}} & \text{No Reinforcement} \\ W_{\text{old}} - W_0 & \text{Negative Reinforcement} \end{cases} \tag{4}$$

2. *Additive Increase Multiplicative Decrease (AIMD)*:
   This scheme conservatively increases weights on positive reinforcement and aggressively reduces on negative reinforcement. Updated weights are given by:

$$W_{\text{new}} = \begin{cases} W_{\text{old}} + W_0 & \text{Positive Reinforcement} \\ W_{\text{old}} & \text{No Reinforcement} \\ \frac{W_{\text{old}}}{2} & \text{Negative Reinforcement} \end{cases} \tag{5}$$

3. *Multiplicative Increase Multiplicative Decrease (MIMD)*:
   In this scheme, positive reinforcement causes the weight of the upstream neighbor to be doubled and negative reinforcement causes the weight to be halved. The key insight that led to this scheme is that the bias introduced by any reinforcement scheme should be such that the randomness in the network does not nullify the benefits of that bias. Updated weights are given by:

$$W_{\text{new}} = \begin{cases} 2W_{\text{old}} & \text{Positive Reinforcement} \\ W_{\text{old}} & \text{No Reinforcement} \\ \frac{W_{\text{old}}}{2} & \text{Negative Reinforcement} \end{cases} \tag{6}$$

4. *Distance Biased Multiplicative Update (DBMU)*:
   This is similar to MIMD except that the degree of reinforcement is not the same across all nodes. The nearer an intermediate node is to the mole, the larger is the degree. This is designed to be more aggressive than pure MIMD. Apart from the insight that led to MIMD, the other key insight that led to this scheme is that nodes closer to the

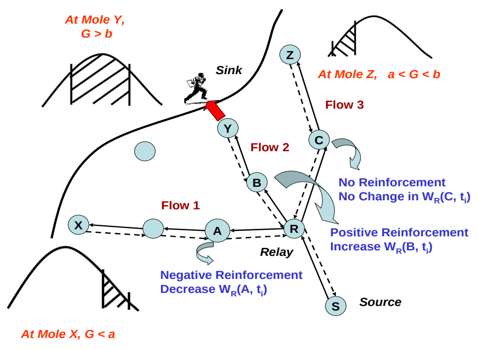

Figure 1. An illustration of the working of HLETDR showing the update process

mole are better indicators of the “goodness” of the path taken; nodes near the source are likely to be part of many paths irrespective of those paths being good or bad. So the consequence of the local decision-making process at every node in a path must be felt more strongly at the nodes near the mole than at the nodes near the source. If d is the distance (measured in terms of hop-count) of an intermediate node from the source, then the updated weight for the upstream neighbor is given by:

$$W_{\text{new}} = \begin{cases} \sqrt{(d+3)}W_{\text{old}} & \text{Positive Reinforcement} \\ W_{\text{old}} & \text{No Reinforcement} \\ \frac{W_{\text{old}}}{\sqrt{(d+3)}} & \text{Negative Reinforcement} \end{cases} \quad (7)$$

The term $\sqrt{(d+3)}$ makes sure that even in the worst case, MIMD is performed. At nodes one hop away from source, this becomes 2 and is even more for nodes farther away from the source.

Note that updates are performed only for the neighbor through which the reinforcement arrived; other neighbors are not affected by the nature of reinforcement performed. This is because a good reinforcement on one neighbor does not mean that other neighbors should be negatively reinforced and vice versa.

3.3. Simulation Experiments, Results and Discussion

Discrete event based simulations were performed to compare these update schemes. A packet level simulator was written in C++ that abstracts away the MAC layer to a pair of transmit and receive functions that have a random delay attached to account for MAC contentions. The link layer also introduces a transmission delay on every packet depending on its size and bandwidth of the link. The network layer uses these two functions to transmit and receive packets. The probabilistic forwarding process and the weight update process are the chief functions of this layer. The packet size is set to 1 Kb and bandwidth is taken as 1 Kbps. The simulation maintains local time-line at every node for use in the routing decision making process. All nodes were randomly placed in a 100m X 100m square region. An approximate trajectory was assigned to the sink along a diagonal of the square. All nodes whose communication and coverage range overlapped with the trajectory were designated as moles. The radio model chosen was the *connectivity within R* model where R was taken as 10m. Since the sink mobility model is outside the scope of this work, it is assumed that the moles have a distribution of arrival times at hand which is characterized by a mean and variance and is assumed to be Gaussian.

There are two main metrics of interest for comparing the update schemes. The first is the *convergence time* which is an indicator of the amount of time taken by the learning process to arrive at an approximate stable set of paths to the mobile sink for a given time-domain. In our simulations, we take this to be the number of sink tours after which the mean length of paths (in terms of hop-count) taken by data packets for a fixed time-domain stabilizes. We find the convergence for a time-domain is dependent on the update scheme used, the density of the field and the number of data flows initiated at the source within that time-domain.

The second metric is the mean cost of the set of paths after convergence. This is an indicator of the efficiency of the routing scheme and is found to be dependent on the update scheme and the node density.

In order to compare the performance of the update schemes in terms of these two parameters, the mean tour duration was taken as 100 sec and time-domain size was fixed at 20 sec. A randomly chosen source node near the bottom-right corner of the field detects an event at the start of its third time-domain. It therefore initiates data flows (taken to be 10 packets per tour) in that time-domain that carry out a random walk independently based on the forwarding probabilities calculated using the weights. Looping of these flows is avoided by having distinct flow ids and marking out nodes that have already been visited once. When a flow reaches a mole, the update scheme is used to modify the weights of the nodes involved in the path (this list is progressively built in the forward path).

3.3.1 Experiments on convergence times:

Fig. 2 shows the mean cost of path with tours for AIAD and AIMD update schemes in the setup described above. It can be seen that while AIMD performs better than AIAD in terms of cost, both the schemes do not show convergence even after 50 tours. While they do manage to locate the sink using an approximate set of paths, this set is not stable; hence oscillations in the mean cost.

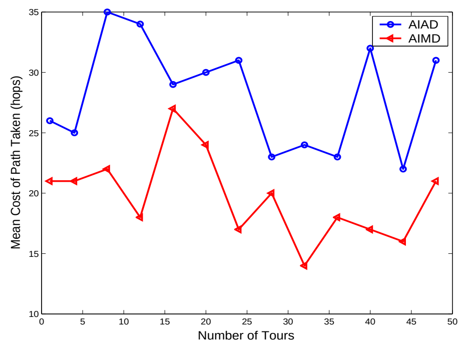

Figure 2. Mean cost of the path obtained using AIAD and AIMD updates over tours. Node density = 200.

Fig. 3 and Fig.4 indicate the performance of MIMD and DBMU for the same setup, albeit for 3 different node densities. They bring out the following facts:

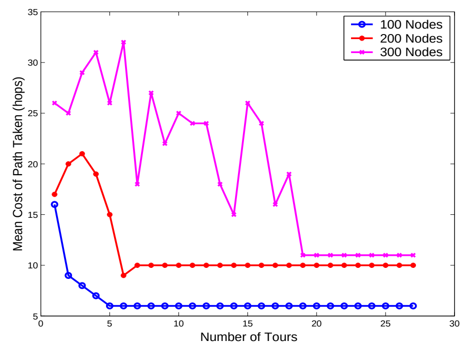

Figure 3. Mean cost of the path obtained using MIMD updates over tours.

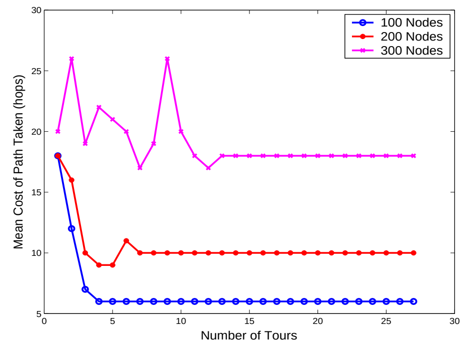

Figure 4. Mean cost of the path obtained using DBMU updates over tours.

1. The fact that convergence is taking place, even at different densities, as indicated by the horizontal section of the plots. After this point, all subsequent packets generated in that time-domain (irrespective of the tour number) are able to find their way to the moving sink efficiently using the same or similar paths.

2. Different densities show not only different convergence times (number of tours for convergence), but also different path lengths after convergence. This trend prevails over all update schemes used.

3. The convergence time for DBMU is the quickest and the convergence time also scales better with increasing density.

4. In the first few tours the mean path length experiences high oscillations, though there is a downward trend. This is because the search process eliminates bad routes from the solution space in spite of the effect of randomness in the network. As more packets are sent, the biases get stronger and this results in the downward trend.

Based on these observations, MIMD and DBMU were chosen as the update schemes in the remaining experiments.

3.3.2 Effect of Number of Packets sent per tour:

Fig. 5 brings out the impact of the number of simultaneous packets sent within the time-domain on the convergence time for MIMD and DBMU. It is observed that for a fixed density, if the number of packets sent within a time-domain increases then the convergence time decreases. This happens for both the schemes. This happens because when multiple packets are sent, a larger part of the network is searched within that time-domain. This implies that HLETDR performs very well for events that generate multiple packets for a sustained amount of time.

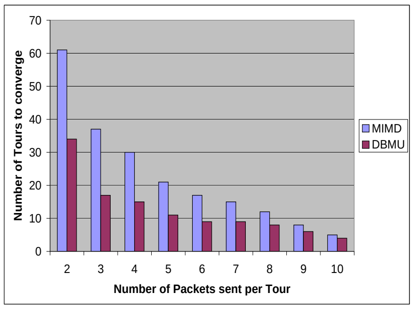

Figure 5. Comparison of the number of tours required to converge to a path for MIMD and DBMU for varying number of packets. Number of Nodes = 100.

3.3.3 Multiple Source Scenarios:

Table 1 captures the impact of multiple sources on convergence time when they all generate data at the same time. The value chosen is the highest time taken among all sources. Two scenarios are considered. First, where the sources are clustered spatially and second, where they are dispersed.

Table 1. Comparison of convergence times (in number of tours) for multiple sources (both clustered and dispersed)

Sources	1	2	3	4	5
Clustered	12	10	9	8	8
Dispersed	12	16	14	11	12

As Table 1 shows, when multiple sources are clustered and generate data at the same, the convergence time decreases. This is because nearby multiple sources increase the intensity of parallel search of the network; the routing table state at any point of time can be favorably used by peer and subsequent packets. If the sources are well dispersed, then the reinforcements act independent of each other. However, with increasing number of sources, regardless of their position in the field, the resultant reinforcements begin to complement each other and hence a decreasing trend sets in.

Fig. 6 helps visualize flows from three close-by sensors. All three sensors share a path to the sink and during path formation, reinforcements resulting from all the three sources complement each other. Individual flows join the path at some point and this enables the possibility of in-network processing and data aggregation.

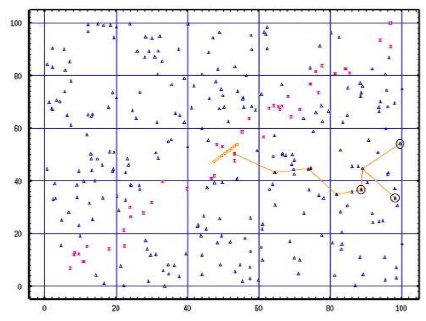

Figure 6. Path formation using DBMU under multiple nearby sources. Number of Nodes = 300.

3.3.4 Node Failure Scenarios:

HLETDR is inherently extremely robust to node failures. Fig. 7 shows that HLETDR is able to recover and find an alternate path very quickly under node failures. In the experiment, two nodes were failed after Tour 19 along the original converged path and the recovery process was observed for different node densities. The spike in the mean path lengths (shown by the vertical dashed line) indicates that an alternate high-cost path is being used; yet the continuing learning process enables fast convergence to a new set of stable paths within 2 tours. Table 2 compares the recovery performance of MIMD and DBMU under varying number of node failures.

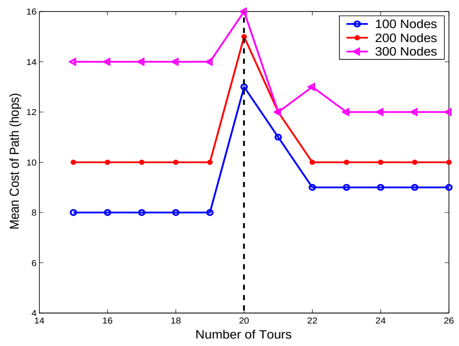

Figure 7. Recovery process under node failure for DBMU for different node densities. Packets sent per tour = 10.

It is seen that DBMU's recovery time is much faster than MIMD.

3.3.5 Comparison with Shortest Path:

Fig. 8 compares the path for MIMD and DBMU after convergence with the theoretically obtained shortest path using Dijkstra's algorithm for different node densities. It is seen that for low densities, the obtained paths are close to optimal. However, for higher densities, the obtained path length are higher. This can be explained by noting that if the communication range is fixed, the average number of neighbors per node increases linearly with the node density. Thus, the solution space for a random walk based method increases as well and can converge to a sub-optimal solution. However, it is more appropriate to scale down the communication radius appropriately with increasing node densities such that the average number of neighbors remains same. This is more so because in a realistic scenario, the neighborhood table space for a node would be bounded. Thus, we also plot the shortest path lengths in Fig. 8 for scaled down radius and find that they are close to MIMD and DBMU path-lengths.

Table 2. Comparison Recovery Time (in number of tours) for MIMD and DBMU under varying degree of node failures. Node density:200

Node Failures	1	2	3	4	5
MIMD	2	2	3	5	6
DMBU	1	2	2	3	3

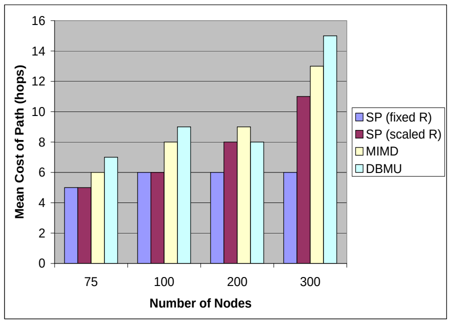

Figure 8. Comparison of the cost of the path obtained using MIMD and DBMU with the shortest paths with fixed and scaled down radius

3.3.6 Comparison with TTDD:

Fig. 9 shows the relative performance of HLETDR with respect to TTDD [3], a sink oriented data dissemination (SODD) scheme also described in [3] and flooding. In SODD, the sink floods the whole network with its location information to construct forwarding gradients towards it. Sources then explicitly route data along these gradients towards the sink. These costs were obtained by using the analytical expressions for communication overhead derived in [3]. It is assumed that all packets are unit length and the number of nodes is 200. Transmission of a packet consumes equal energy over all links in the network. For SODD, the sink advertises once every time-domain. The side of a TTDD grid-square is calculated to be approximately 14 meters in order to uniquely cover every part of the sink's trajectory given the number of time-domains.

In this setup, HLETDR performs almost three times better than TTDD and SODD. Even when a very few packet are sent, HLETDR performs better because TTDD involves a dissemination grid formation overhead as well as localized flooding by the mobile sink. It is to be noted, however, that TTDD and HLETDR have been designed with slightly different scenarios in mind. TTDD is designed to run in scenarios that require explicity querying, while HLETDR is not. HLETDR assumes that there is an underlying pattern in the sink's mobility, while TTDD does not.

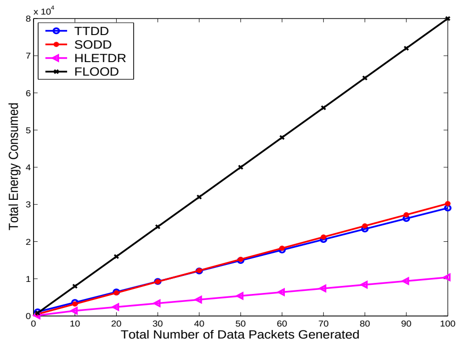

Figure 9. Comparison of energy efficiency of HLETDR, TTDD, SODD and flooding. Number of Nodes = 200.

4. Conclusions and Future Work

We have proposed a scalable and distributed learning-based approach to find efficient and robust routes to a mobile sink in a wireless sensor network. The scenarios that we were interested in involved sinks which move in a certain pattern. The key contribution of our work is identifying that learning-based mechanisms can be used in such scenarios to find routes that satisfy the requirements of energy-efficiency and robustness. The scheme that we have proposed uses a combination of statistical and reinforcement learning to come up with good data delivery paths to the mobile sink. This is a distributed scheme that does not require explicit time synchronization or node localization and is sensitive to in-network delays. It is highly energy-efficient and furthermore, it is robust to node failures because of its inherent exploratory nature.

As part of future work, we would like to come up with mechanisms that converge to paths that are closer to the theoretically optimal paths. We plan to study different statistical learning schemes that the moles can use to come up with probability distributions. The impact of different sink mobility models is another interesting extension. Finally, we would like to obtain analytical expressions for our protocol's performance.

References

[1] H.S. Kim, T.F. Abdelzaher, and W.H. Kwon,“Minimum-Energy Asynchronous Dissemination to Mobile Sinks in Wireless Sensor Networks,” *The First ACM Conference on Embedded Networked Sensor Systems (Sensys'03)*, Los Angeles, CA, November 2003.

[2] R.C. Shah, S. Roy, S. Jain, and W. Brunette, “Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks,” *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*, Anchorage, AL, May 2003.

[3] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, “A Two-Tier Data Dissemination Model for Large-scale Wireless Sensor Networks,” *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networks (MobiCOM 2002)*, Atlanta, GA, September 2002.

[4] C. Intanagonwiwat, R. Govindan, and D. Estrin, “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks”, *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCOM 2000)*, Boston, MA, August 2000.

[5] Bhaskar Krishnamachari, Congzhou Zhou, Baharak Shademan, “LEQS: Learning-based Efficient Querying for Sensor Networks”, *USC Computer Science Technical Report CS 03-795*, 2003.

[6] F. Ye, G. Zhong, S. Lu, L. Zhang, “A Robust Data Delivery Protocol for Large Scale Sensor Networks ”, *Proceedings of Second International Workshop on Information Processing in Sensor Networks (IPSN'03)*, Palo Alto, CA, April 2003.