Time Indexing in Sensor Networks

Rong Zheng[†]
Department of Computer Science
University of Houston
Houston, TX 77204
E-mail: *rzheng@cs.uh.edu*

Guanghui He, Indranil Gupta and Lui Sha
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
E-mail:{*ghe, indy, lrs*}*@cs.uiuc.edu*

***Abstract*— In this paper, we define the *time indexing* problem as the in-network storage and querying of sensor network data based solely on the time attribute. We argue qualitatively why existing storage schemes may be insufficient as solutions. We then present, analyze, and evaluate novel and lightweight solutions to both the storage and the querying sub-problems for time indexing. First, the time-indexed storage problem is formally defined, and two formulations are presented, seeking to optimize generic utility functions that are derived from concerns about energy, bandwidth usage, and storage balancing. We present and analyze decentralized protocols to solve these formulations, and prove the optimality of some of our solutions. Secondly, maintenance and use of simple overlays among rendezvous point nodes, in order to enable fault-tolerant and efficient time-indexed queries, are discussed. Finally, simulation results are presented to quantify performance characteristics of the protocols, and we find that our proposed scheme has low query overhead that scales with system size and density while exhibiting very good load balancing and fault tolerance properties.**

I. Introduction

Over their deployment period, wireless sensor networks (WSNs) can generate enormous amount of data. This data is then stored so that it can be queried by either a special set of destination nodes (e.g., querying users on computer hosts) or collections of individual sensors (e.g., for distributed signal processing). Existing solutions store a given data item either (1) externally at the edges of the sensor network, or (2) locally at the sensor node generating the data, or (3) in-network, i.e., distributedly at intermediate sensor nodes [16].

In-network storage is suited to data that is neither very old (and thus unlikely to be queried) nor current (which can only be present at the measuring sensor node itself). In-network storage allows efficient access to the data by both destination nodes and sensor nodes. Existing solutions in this category include data-centric storage (DCS) schemes, e.g., [16], [15], [9], [1]. These schemes treat each data as a tuple of attributes and store data or meta-data about entries that lie within given attribute ranges at small collections of sensor nodes.

Many WSN applications are concerned with the time aspect of sensor data, and query data items by the *time* attribute that specifies the (approximate) time at which the data item was generated. Examples include queries such as “what is the average of temperature readings last Tuesday between 10 am and 11 am?” (in an environmental observation and forecasting system), and “find the number of sensor nodes alive between 9 pm and 9.30 pm on Tuesday”. We call these as *time-indexed queries*. Unfortunately, existing DCS schemes are inadequate to handle time-indexed queries for several reasons. First, storing low-level time series data at distant sensor locations incurs significant communication overhead. Therefore, DCS schemes are best suited to scenarios where the nature of some archetypal high-level events is well-defined as opposed to low-level time series data. Second, in some DCS schemes, all data items within a given time interval range are hashed to (and stored at) a small set of sensor nodes, which tend to be geographically proximate. This has fault-tolerance problems since if that a particular region is affected by a failure or catastrophic event, all data for the interval in question will be lost. Lastly, this approach also creates hot-spots near a region when all other sensor nodes are in the process of inserting data entries for an interval that maps to a node in this region.

Our findings in this paper reveal that good fault-tolerance and load-balancing can be achieved for time indexed queries by using an in-network storage scheme that embeds a simple time-indexed structure through fully localized mechanisms. To enable time-indexed in-network storage of sensor data, we propose the use of *rendezvous points (RPs)* that are geographically dispersed. Each RP stores sensor readings collected from its neighborhood for scheduled periods of time. Non-RP nodes can be optionally put to low-power states to conserve energy and storage space. A sensor's role as an RP or non-RP node changes over time subject to a carefully designed schedules. We propose a generic utility function-based framework that formalizes the RP election problem as a non-linear integer programming problem to explicitly account for the trade-off between load balancing and energy/storage utilization. To solve the integer programming problem, we first present a fully localized algorithm to its fractional relaxation. Based on the optimal solution thus obtained, we devise a novel voting scheme to determine the set of RPs for each time interval. Our analysis shows the optimality and convergence properties of our proposed solutions.

To support efficient time-indexed queries, we describe how overlays of RPs can be constructed. Queries are forwarded along the overlay and sensor readings are gathered/aggregated on the reverse path to the query point. The number of query and response messages is $O(|V|\frac{+ln(D_{min})}{D_{min}})$, where $|V|$ is the number of sensors and $D \quad _n$ is the minimum node degree. Our simulation results show that the proposed query processing protocol outperforms a randomized scheme in terms of the number of messages transmitted per query and robustness in presence of node failures.

A comment is in order about clock skews. Clock skews across sensor nodes may affect the correctness of data returned

[†]The work was done when the author was at Dept. of Computer Science, UIUC.

by a time-indexed query. We assume that the intervals specified by time-indexed query are much larger compared to the clock skews across sensor nodes; this can be achieved by running existing synchronization algorithms, e.g., [8], and only infrequently. Hence, in this paper, we assume that the sensor nodes are synchronized.

The rest of the paper is organized as follows. In Section II, we give a succinct review of related work. In Section III, we formally define the time indexing problem in sensor networks and provide an overview of our proposed solution. In Section IV, we formulate a utility-based RP election problem. In Section V, a localized algorithm to approximate the optimal utility-based RP election is described. The protocol for forwarding query and retrieving time-indexed data is laid out in Section VI. In Section VII, we present simulation results to evaluate the performance of our proposed scheme. Finally, we conclude the paper in Section VIII with a list of future work.

II. Related Work

Existing solutions to data storage in sensor network mainly fall into three categories [16]: external storage (*ES*), local storage (*LS*) and data-centric storage (*DCS*).

In *ES*, data is stored at an external storage point for processing. In *LS*, information is stored locally. Queries for interested data need to be flooded in the network. For example, in the direct diffusion paradigm [10], a sink expresses its interest by sending out a query message to specify the event type, event rate and timespan. The query is flooded in the network and used to establish a gradient toward the sink at each node. Once the sources (of the interested events) are reached and the information flow is established from source to the sink, the sink can reinforce paths which have better quality. Replies from the sources can be aggregated at intermediate nodes. TAG [13] proposes an aggregation service for sensor networks that enables in-network aggregation and pruning of interested data that satisfy certain attributes based on pre-defined statistics such as, median, mean or histogram.

In *DCS*, data is stored by event type within the sensornet at designated nodes using data-centric naming. Queries for certain types of events are routed directly to the corresponding network node. GHT [15] hashes named data to a geographical location. Structured replication is provided in GHT to improve efficiency and load balancing. DIFS [9] is another instance of DHT in WSNs. It is built upon GHT and considers high-level events with multiple attributes. A multiple-rooted hierarchical index tree structure is constructed in DIFS to facilitate range query forwarding. In [1], a resilient DCS scheme (R-DCS) is proposed to replicate data and control information at strategic locations for better fault tolerance in sensor networks. Compared to LS, DCS has the advantage of using less number of messages during the query stage at the cost of additional overhead to store the data. Therefore, DCS is more appropriate to store high-level events generated infrequently.

III. Overview

A. Time Indexing in Sensor Networks

A sensor network is represented by an undirected graph $G(V, E)$, where V is the set of sensor nodes, and E is the set of edges. Let $_v$ be the closed neighborhood of , i.e., $_v$ includes as well as all its direct neighbors. Let $_v$ be its degree, i.e., $_v = |\ _v| -$. A node senses the environment and records the sensing results in the time series, $x(\ ,\ )$ (the sampling frequency is an application-specific parameter). We formally define the time indexing problem as follows:

Definition 1 (Time Indexing Problem): The *Time Indexing Problem* is concerned with efficient storage and retrieval of information based on time attributes. *Time-indexed storage* maps (and stores) time series data $\{x(\ ,\ )|\ \in V,\ \in (\ \Delta\ , (\ )\Delta\ )\}$ to a subset of nodes in the network, where $\in \mathbb{N}$ is a non-negative integer. A *time-indexed query*, $q(\ , \Delta\ )$ retrieves the individual or aggregated sensor readings during time interval $(\ \Delta\ , (\ )\Delta\ )$, i.e., $\{x(\ ,\ )|\ \in V,\ \in (\ \Delta\ , (\ )\Delta\ )\}$.

Clearly, one straightforward solution to the time indexing problem is to store information at sensing nodes locally and perform a network-wide broadcast for queries. The communication cost (in number of transmissions) in storing the information is zero, while the *per query* communication overheads for broadcasting the query and gathering time-indexed data are $O(|V|)$ and $O(|V| \cdot \overline{L})$ *per query*, where $|V|$ is the number of nodes in the network and $\overline{L}$ is the average path length from a sensor node to the query point. On the other hand, an external storage (ES) solution which stores data externally requires $O(|V| \cdot \overline{L})$ transmissions *per sample interval* and zero cost for query processing.

In contrast, we propose an in-network storage scheme that selects a subset of sensors, i.e., *rendezvous points* (RPs) to collect, compress and store sensor data from its neighborhood for pre-defined periods of time. Furthermore, as sensor data gathered at close-by sensors is in general highly redundant, it is sometimes sufficient to use these RPs to record sensing information and put the other nodes in low-power state to conserve energy and storage space. As the set of RPs change over time, a time-indexed storage structure is naturally embedded in the network and can be used to expedite query processing.

B. Outline of Proposed Solution

Our proposed solution to time indexing in sensor networks consists of two integral parts: 1) distributed time-indexed storage using RPs and 2) information retrieval based on time attributes.

The key design considerations are:

Low protocol overhead to maintain time-indexed structure. In particular, localized communication is desired for energy conservation reasons.

Low communication overhead to query and obtain interested data. The query processing overhead incurred should scale well with node density.

Energy and storage balancing. Both energy and storage are non-replenishable resources[1]. Sensor devices are usually battery-powered. Exhaustion of battery power degrades data gathering capability and sometimes may cause partitions in the network. Exhaustion of local

[1]Old data can be expelled from sensor storage but at the expense of degraded availability should such data be needed.

storage space forces a node to store data at remote locations and incurs extra communication overhead and energy cost.

In this paper, the distributed time-indexed storage is achieved by the use of rendezvous points (RP). For time interval (Δ , ( )Δ), RPs are selected who are responsible to sense the environment and collect (and perhaps aggregate) sensor readings from other non-RP nodes. The RPs form a dominating set of the network, i.e., all the other sensors are within one-hop wireless transmission radius to the closest RP nodes. It should be noted that our methodology can be readily extended to k-hop dominating sets.

The problem of finding dominating sets in a graph has been extensively studied in literature [6], [11] and applied in the context of ad hoc routing and connectivity management [7], [3], [17]. However, most of the work focuses on finding a *single* (connected) dominating set using least number of nodes. The key difference of our problem formulation is that we are interested in finding *multiple* of such sets. We propose a novel utility-based RP election mechanism that satisfies vertex coverage and load balancing requirements simultaneously so that the utilization of network resources can be maximized. It can be executed in a fully localized fashion at the initial phase of sensor deployment. In case of node failures, a local maintenance is first performed as a temporary remedy. The RP election mechanism can then be carried out infrequently during the lifetime of the network.

To retrieve sensing information based on time attribute, we construct an overlay that interconnects RPs for each time slot. Queries for time (Δ , ( )Δ) are forwarded on the overlay and the sensed information in each RP is aggregated along reverse paths to the query point.

An example of RP nodes and the overlay constructed for interval (Δ , ( )Δ) is given in Figure 1 using the algorithms presented in Section V and Section VI. From Figure 1, we can see the number of RP nodes (= 7) is significantly less than the total number of nodes (=) in a dense network. This greatly reduces the energy cost and communication overhead for storing and querying time-indexed data. Over time, multiple instances of such RP nodes and overlays are used to store time-indexed data.

IV. Utility-based RP Selection

In the next two sections, we investigate how to generate RPs for time-indexed storage. We first formulate the RP election problem to be a non-linear integer programming (NIP) problem and then present its fractional relaxation. We devise fully localized solutions to the NIP problem in Section V.

A. Notation

We divide the time axis into slots of length Δ . The slot size is chosen to be equivalent to the query interval length (discussions on how to handle variable query intervals can be found in Section VIII). Each node maintains a schedule of length T slots for RP election. T can be chosen arbitrarily but is typically on the same order of magnitude as the maximum node degree. The schedule is represented by ’s and ’s in each slot, where a ‘1’ slot represents that the node serves as an RP in this slot. Let $\overrightarrow{\ }_v$ represent the schedule vector of node , consisting of a series of ’s and ’s. $_v(\ )$ is the th element of $\overrightarrow{\ }_v$. Finally, $_v = |\overrightarrow{\ }_v|$ is the cardinality or the number of ‘1’ slots in node ’s schedule, i.e., the number of slots (out of T slots) a node serves as an RP. The schedule repeats every T slots.

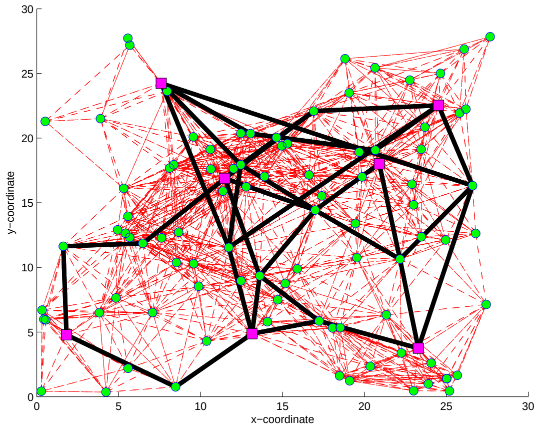


Fig. 1. RP nodes and the overlay constructed. RP nodes are represented by squares. Two sensor nodes are connected by a thin line if they are within wireless transmission range. Thick lines correspond to links on the overlay. $|V| = 100$, transmission range = 8, size = 28x28.

B. Problem Definition

To ensure that sensor data propagates at most 1-hop away in the storage phase, RPs at time slot form a dominating set of $G(V, E)$. Let $_{RP}$ and $_{non-RP}$ be the power consumption of RP and non-RP nodes (per Δ slot). The average power consumption is thus,

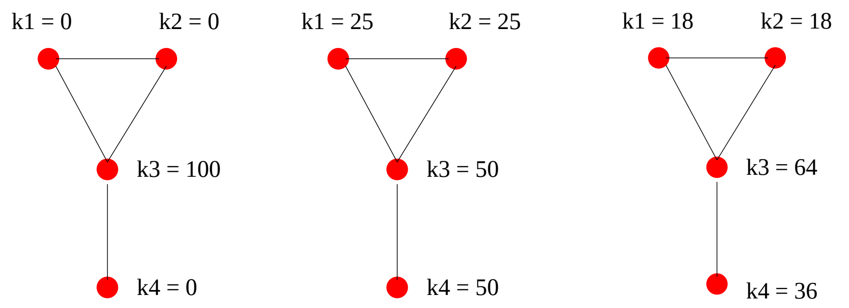
$$P_{avg} = \frac{1}{T} \sum_{v \in V} (k_v P_{RP} + (T - k_v) P_{non-RP}) \qquad (1)$$

Under the assumption that $_{RP}$ $_{non-RP}$, $_{vg}$ is minimized if $\sum_{v \in V}\ _v$ is minimized. By changing the order of summation, $_{vg}$ can be rewritten as $\frac{P_{RP} - P_{non-RP}}{T} \sum_{=0}^{\in T} \sum_{v \in V}\ _v(\ )\ \ |V|\ _{non-RP}$. This implies that to minimize the average power consumption, a minimum dominating set with the least number of RPs need to be selected for each slot.

For nodes equipped with infinite power sources, minimum dominating sets consume least amount of energy over time; or equivalently, it leaves out maximum amount of residual energy for other communication tasks. For energy-constrained networks, the above RP election scheme is not always desirable as nodes in the minimum dominating set are depleted of energy much faster than the rest of nodes. In this paper, we propose a generic framework that can account for the trade-off between energy/storage utilization and load balancing. Our approach is motived by work on the bandwidth sharing problem in wireline networks [14]. More specifically, we define the following utility-based RP election problem.

Problem 1: (Optimal Utility-based RP election problem (I $_{URP}$)) Given T,

$$\begin{array}{ll} max & \sum_{v \in V} U(k_v) \\ s.t. & \sum_{u \in N(v)} s_u(i) \geq 1 \quad i = 1, 2, ..., T \text{ and } \forall v \in V, \end{array} \qquad (2)$$



(a) $U(x) = -x$ (b) $U(x) = -(- \quad (x \quad T))^{\alpha}$ (c) $U(x) = \quad (T - x)$

Fig. 2. Optimal solution for NIP_{URP} with different utility functions, $T = 100$.

where $U(\cdot)$ is a concave utility function and the second inequality reflects the dominating set requirement for each time slot, i.e., each node is either itself a RP or is dominated by a RP. Clearly, $I \;_{URP}$ is in general a non-linear integer programming problem with linear constraints. Next, we discuss how the choice of utility function affects the trade-off between energy (or storage) utilization and load balancing.

Linear utility function: If $U(x) = -x$, $I \;_{URP}$ tries to find a configuration to minimize the total (or equivalently average) number of '1' slots. Solving this problem is equivalent to finding the minimum dominating set as shown below.

Theorem 1: Let $MDS(G)$ be the minimum dominating set of $G(V,E)$. The following schedule maximizes $I \;_{URP}$ with utility function $U(x) = -x$. If $\in MDS(G)$, $_v(\;) = \;,$ $\in V$ and $= \;, 2, ..., T$. Otherwise, $_v(\;) = \;,$ $\in V$ and $= \;, 2, ..., T$.

Proof: Consider a feasible schedule $\vec{\;}_v$, $\in V$. We have $\sum_{v\in V} \;_v = \sum_{=}^{T} \sum_{v\in V} \;_v(\;) \geq T \cdot |MDS(G)|$. The second inequality is due to the definition of MDS. ■

An example of the optimal solution under linear utility function is given in Fig. 2(a). In Fig. 2(a), other than node 3, which forms the minimum dominating set, all other nodes never serve as RPs. Assume each node has 1000 units of energy initially. Let $_{RP} =$ and $_{non-RP} =$. Clearly, the total power consumption (defined in Eq. (1)) is minimized as $_{vg} =$. However, node 3 is depleted of its energy much faster than the other nodes. If network life-time is defined as the time that the first node runs out of power, the network life-time in this case is only 1000 slots (and the network is partitioned as well in this particular example).

Max-min utility function: For $U(x) = -(- \quad (x \quad T))^{\alpha}$, $< x \leq T$ and $\to \infty$. $I \;_{URP}$ corresponds to the max-min allocation. Or equivalently, the optimization objective function can be rewritten as:

$$\max \min_{v \in V} (-k_v) \tag{3}$$

In essence, max-min allocation tries to maximize the time till the first node's depletion of energy (or storage). An example is given in Figure 2(b). From the example, we can see that the max-min allocation requires, on average, higher overall power consumption, $_{vg} = \;.5$. On the other hand, the time it takes for the first node to run out of power is maximized, which equals to 2000 slots.

Logarithmic utility function: The above two utility functions represent two extremes in the design space of load balancing and energy utilization. Linear utility function minimizes the overall power consumption among all nodes but some nodes may be depleted of energy much earlier than the other nodes. If there are no other activities in the network, max-min utility function maximizes the time till the first node's depletion of energy (or storage) at the expense of higher overall power consumption.

We consider the use of logarithmic utility function, i.e., $U(x) = \quad (\; - x \quad T)$. Solutions to logarithmic utility function are inherently load balanced as the form of utility function prohibits the scenarios where a subset of nodes serve as RPs all the time, i.e., $_v = T$ for some nodes and thus $U(\;_v) \to -\infty$. On the other hand, since $\quad (\; - \;_v \quad T) \approx - \;_v \quad T$ for small $_v \quad T$, the optimal schedule devised should have low power consumption. An example of the optimal solution is shown in Figure 2(c).

The $I \;_{URP}$ Problem is in general NP-complete (with the exception of max-min utility function). However, its relaxation with fractional constraints can be solved using existing optimization methods in polynomial running time. Next, we define the following non-linear programming problem.

Problem 2: (Optimal Fractional Utility-based RP Election Problem ($L \;_{URP}$)) given T,

$$\begin{array}{lll} max & \sum_{v\in V} U(x) & \\ s.t. & \sum_{u\in\{N(v),v\}} x_u \geq 1, & \forall v \in V \\ & 0 \leq x_v \leq 1, & \forall v \in V. \end{array} \tag{4}$$

where $x_v = \;_v \quad T$, i.e., the fraction of time slots in which node serves as an RP.

The optimal solution to the relaxed problem gives an upper bound for the original $I \;_{URP}$ problem. It should be noted that not all solutions to $L \;_{URP}$ are feasible for $I \;_{URP}$. In another word, the $L \;_{URP}$ solutions are not necessarily schedulable in a slotted system. One example is odd cycles.

The following theorem shows that for regular graphs, all concave utility functions have the same optimal solution to $L \;_{URP}$.

Theorem 2: If $G(V,E)$ is regular with degree for each node, all concave and strictly decreasing utility objective functions admit the same optimal solution to the $L \;_{URP}$ problem.

Proof: From concavity, $\frac{}{|V|}\sum_{v\in V} U(\;_v) \leq U(\frac{}{|V|}\sum_{v\in V} \;_v)$ with the equality holds when $_v = \;,$

∀ ∈ V, where is a constant. From the constraint in Problem (2), we have ( ) $\sum_{v\in V}$ $_v$ ≥ $T \cdot |V|$ with equality holds when $_v = \frac{T}{d+}$, ∀ ∈ V. Since $U(x)$ is non-increasing, $\frac{}{|V|}\sum_{v\in V} U(\ _v)$ achieves its maximum $\frac{}{|V|}U(\frac{T}{d+})$ when $_v = \frac{T}{d+}$, ∀ ∈ V. ■

The above results for regular graphs can be used as an initial solution for RP election in random graphs, i.e, initially a node chooses to be an RP for $\frac{T}{d_v+}$ slots among all T slots.

V. A Localized Algorithm for RP Election

In this section, we consider the optimal RP election problem under the logarithmic utility function. First, a Newton-like method is applied to obtain the optimal solution to L_{URP}. The algorithm can be implemented in a fully localized fashion. Secondly, we propose a voting mechanism to determine the set of RP nodes for each slot based on the optimal solution to L_{URP}. Note that although our analysis is based on the logarithmic utility function, the proposed methodology is applicable to other concave utility functions.

A. Solution to Problem L_{URP}

Let $x_v = \frac{v}{T}$ and $_v =\ - x_v$. Define the Lagrangian

$$\begin{aligned} L(y,p) &= \sum_{v\in V} logy_v - \sum_{v\in V} p_v(\sum_{u\in N(v)} y_u - d_v) \\ &= \sum_{v\in V}(logy_v - y_v \sum_{u\in N(v)} p_u) + \sum_{v\in V} p_v d_v. \end{aligned} \tag{5}$$

Notice that the first term is separable and thus $\max \sum_{v\in V}(\ _v - \ _v \sum_{u\in\ (v)}\ _u)$ = $\sum_{v\in V} \max_{\ _v}(\ _v - \ _v\sum_{u\in\ (v)}\ _u)$. The objective function of the dual problem of L_{URP} is thus

$$D(p) = max_y L(y,p) = \sum_{v\in V} \max_{x_v}(logy_v - y_v \sum_{u\in N(v)} p_u) + \sum_{v\in V} p_v d_v$$

and the dual problem is:

$$\textbf{D:}\ min_{p\geq 0} D(p) \tag{6}$$

Since $(\cdot)$ is concave and the constraints are linear, there is no duality gap and Lagrange multipliers exist. If $^* \geq$ is dual optimal then $(\ _v(\ ^*),\ \in V)$ is primal optimal (i.e., $^*_v = \ _v(\ ^*)$) provided that it is primal feasible and complementary slackness is satisfied.

Newton-like solution to L_{URP}**:** To solve the dual problem, we adopt the Newton-like method [4]. Alternatively, one can use gradient-descent method [12]. However, the convergence is slower. The algorithm is described in Figure 3. For simplicity of presentation, we assume a synchronous communication mode. That is, in every communication round, each node is allowed to send one message to each of its direct neighbors in $G(V,E)$. The problem of simulating a synchronous network by an asynchronous network has been investigated in [5].

Let H_{vv} be the diagonal terms of the Hessian of D (defined as $\nabla^2 D(\ )$, where ∇^2 means the second order derivative). H_{vv} can be approximated by:

$$H_{vv} = \nabla^2 D(p)_{vv} \approx - \sum_{u\in N(v)} \frac{y_u}{\sum_{w\in N(u)} p_w}. \tag{7}$$

```
Input: $N(v)$, $\forall v \in V$
Output: $NLP_{URP}$-solution $\alpha_v \equiv 1 - y_v^*$, $v \in V$
1. $iter = 0$
2. $p_v(0) = \frac{1}{d_v}$
3. do
4.     $p_v(iter+1) = [p_v(iter) + \gamma H_{vv}^{-1}(iter)(\sum_{u\in N(v)} y_u - d_v)]^+$
5.     send $p_v(iter+1)$ to all neighbors
6.     $y_v(iter+1) = \min(1, \frac{1}{\sum_{u\in N(v)} p_u})$
7.     send $y_v(iter+1)$ to all neighbors
8.     $iter = iter + 1$
9. while (termination conditions not satisfied)
10. $\alpha_v = 1 - y_v$
```

Fig. 3. **Algorithm 1:** NLP_{URP} Newton-like Solution.

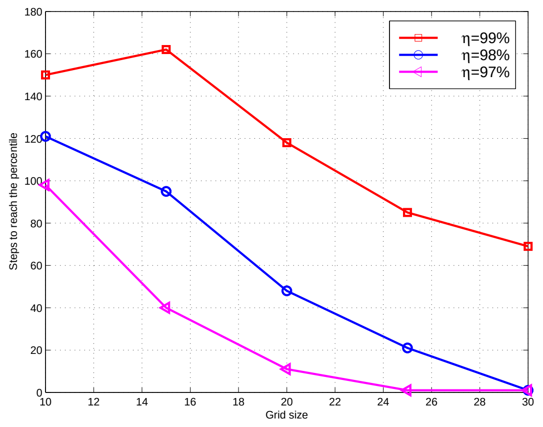


Fig. 4. Number of steps needed to reach η percent of the optimal utility.

To make H_{vv} strictly positive, for any ϵ ,

$$H_{vv} = max\{\epsilon, -\sum_{u\in N(v)} \frac{y_u}{\sum_{w\in N(u)} p_w}\} \tag{8}$$

$_v$ is the price incurred at node if the constraint $\sum_{u\in\ (u)}\ _u \geq\ _v$ is violated. Line 4 in Fig. 3 follows $_v(\ ) = [\ _v(\ ) - \gamma H_{vv}^{-}\frac{\partial D}{\partial p_v}(\ (\ ))]^+ = [\ _v(\ )\ \gamma H_{vv}^{-}(\sum_{u\in\ (v)}\ _u -\ _v)]^+$. Therefore, if $\sum_{u\in\ (u)}\ _u \leq\ _v$, $_v$ decreases and consequently, $_v$ would increase in Line 6. Line 6 results from node tries to maximize its own utility unilaterally given price , i.e., $_v(\ ) = \arg_{y_v\in(0\ ]}\max(\ _v -\ _v\sum_{u\in\ (v)}\ _u)$.

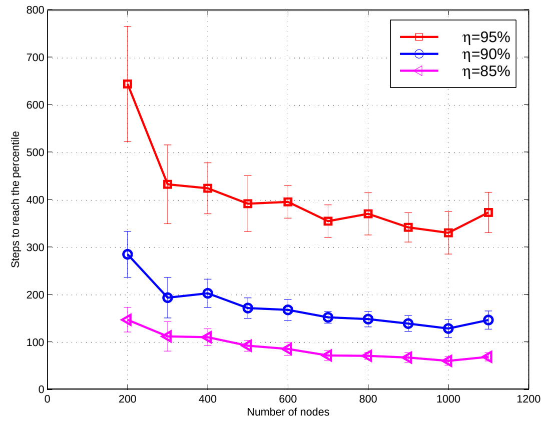
Algorithm 1 is a localized algorithm because the constraints are localized. Only information about and is exchanged within one-hop neighbors at each step. Furthermore, if is limited to $(\ ,\ ]^{|V|}$, we have the following convergence result.

Theorem 3: Let $\Delta_{\max}$ be the maximum degree of $G(V,E)$. Starting from any initial $(\ ) \in (\ ,\ ]^{|V|}$ *and* $(\ ) \leq$ *and under the condition that* $< \gamma < 2(\epsilon\ \Delta\ \ )^2$*, the sequence* $(\ (\ ),\ (\ ))$ *generated by Algorithm 1 converges to an accumulation point* $(\ ^*,\ ^*)$*, where* * *is the primal-dual optimal.*

Proof: The proof directly follows from [4]. ■

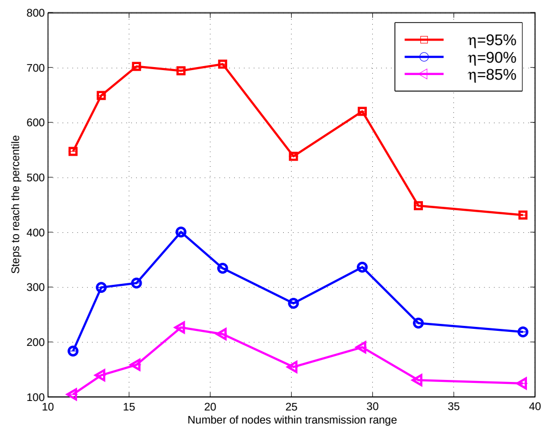
Termination of Algorithm 1 can be decided locally at each node if the price does not change significantly.

Number of iteration steps of the Newton-like method: To study the convergence speed of the Newton-like method with respect to the number of nodes and node density, we conduct



$ode\quad u\quad ber$	200	400	600	800	1000
$M\quad_{degree}$	34	37	39	39	39
$M\ n_{degree}$	4	5	5	5	4
Ave_{degree}	21	22	23	23	23

Fig. 5. Number of steps needed to reach η percent of the optimal utility with fi xed density and increasing number of nodes, transmission range = 8.



# of nodes/tx range	14	18	25	33	39
$M\quad_{degree}$	44	39	30	24	20
$M\ n_{degree}$	6	3	2	2	1
Ave_{degree}	27	21	15	12	10

Fig. 6. Number of iteration steps to reach η percent of the real optimal with fi xed number of nodes and increasing density, $|V| = 200$.

experiments on grid and random graphs. In each experiment, we first compute the optimal solution and then measure the number of iteration steps needed to reach η percent of the optimal solution.

Fig. 4 shows the results for square grid graphs. The grid size varies from 10 to 30. For the random graphs, in the first set of experiments, we fix the node density and change the number of nodes. In the second set of experiments, we keep the number of nodes constant and vary the node density in the graphs. The results are shown in Fig. 5 and Fig. 6.

From Fig. 5 – Fig. 6, we can see that the proposed algorithm is not very sensitive to the number of nodes or node density in random graphs. For grid graphs, as the size increases, we observe a decreasing trend in the number of iteration steps. This is intuitively true because as the number of nodes increases, less percentage of nodes are on the boundary (with smaller degree) and thus initial values ($\frac{}{d+} = \ .2$) are closer to the optimal solution.

```
Input: Feasible solution ($\overrightarrow{\alpha}$) for $NLP_{URP}$
Output: $NIP_{URP}$-solution $\overrightarrow{s_v}$, $v \in V$

1.  $vote\_count_v(i) = 0$
2.  for i = 0 to T
3.    generate random variable $bid_v(i)$ with pdf $f(x, \alpha_v)$
4.    send $bid_v(i)$ to all neighbors
5.    if $bid_v(i) == \max_{u \in N(v)} bid_u(i)$ then
6.        $s_v(i) = 1$
7.        $vote_v(i) = v$
8.    else
9.        $s_v(i) = 0$
10.       $vote_v(i) = \arg_u \max_{u \in N(v)} bid_u$
11.   fi

12.   send $vote_v(i)$ to all neighbors

13.   foreach $u \in N(v)$
14.       if $vote_u(i) == v$ then
15.           $vote\_count_v(i) = vote\_count_v(i) + 1$
16.       end
17.   end

18.   if $vote_v(i) == v$ (C.1) or $vote\_count_v(i) \geq l$ (C.2) then
19.       $s_v(i) = 1$
20.   fi

21.   send $s_v(i)$ to all neighbors
22.   if $\sum_{u \in N(v)} s_u(i) == 0$ then
23.       $s_v(i) = 1$
24.   end
25. end
```

Fig. 7. **Algorithm 2:** A localized vote algorithm for slot assignment based on solutions to NLP_{URP}.

B. A Heuristic Solution to Problem $I\quad_{URP}$

Next, we derive 0, 1 integer solution to the original $I\quad_{URP}$ problem based on the optimal solution to Problem $L\quad_{URP}$. In Fig. 3, algorithm 1 assigns a tuple ($_v$, $_v$) for each node $\in V$, where $_v$ is the fraction of slots that is an RP and $_v$ is a penalty associated with node . For large T, $_v$ can also be viewed as the probability for node to be an RP in each time slot. In what follows, we propose a voting scheme that makes each node to be an RP for approximately $_v \cdot T$ slots. The algorithm is given in Fig. 7.

From Fig. 7, at the end of execution, Algorithm 2 ensures the constraints in Problem $I\quad_{URP}$ are satisfied. To generate a random variable at each node, we choose the pdf $f(x,\ ) \triangleq \ x^{\alpha-}$ with CDF $F(x,\ ) \triangleq x^{\alpha}$, $x \in [\ ,\ ]$, such that the following property is satisfied:

Property 1: Let $u \in \ (\ )$, the probability that u has the maximum bid in slot among nodes in $(\ )$ (i.e., $u = \quad x_{w\in\ (v)}\ _w(\ ))$ is $_u$, if $_v$.

Proof: $\{\ _v(\ ) = u\} = \int_0\ _u x_u^{\alpha_v-} \prod_{w\in\{\ (v)/u\}} x_w^{\alpha_w}\ x_w = \frac{\alpha_v}{\sum_{w\in N(v)} \alpha_w}$. From Algorithm 1 for $_v$, $\sum_{w\in\ (v)}\ _w =$ (i.e., the corresponding constraint is tight). Therefore, $\{\ _v(\ ) = u\} = \ _u$. ■

Let $u =$, then from the above property, we have (C.1) is satisfied with probability $_v$ for $_v$. Therefore, $E\{\ _v(\ ) = \ \} = \ _v$ when only condition (C.1) (Line 18 in Fig. 7) is met since the bids for each slot are independent. Unfortunately, due to randomness, for a single trial, the constraints in Problem $I\quad_{URP}$ may not always be satisfied. Therefore, we introduce the second condition (C.2) (Line 18 in Fig. 7) to slightly increase the probability that a node serves as a RP. is a protocol-dependent parameter. Simulation results on different topologies demonstrate that by setting to 2, Algorithm 2 achieves best approximation to the optimal.

Algorithm 2 requires three message exchanges per node. The slot assignment for T slots can be processed in parallel by aggregating *bid* and *vote* information for each slot into a single packet (as allowed by the maximum packet size).

VI. Information Retrieval Based on Time Attribute

In this section, we describe the protocol to retrieve information from time-indexed storage in sensor networks.

A. Protocol Details

From Section V, we know that in each time slot, the set of RP nodes form a dominating set. To retrieve information in a particular slot, it is sufficient to only query RPs of the corresponding slot instead of conducting network-wide floods of query messages. To reduce the communication overhead, each RP needs to keep the list of RPs in its neighborhood and routes to reach them. Processing of a query initiated at an arbitrary location in the sensor network consists of three steps. First, if the query is injected at a non-RP node, it needs to find a RP neighbor for the corresponding slot. Second, the RP neighbor forwards the query information to other RP nodes. Lastly, query responses are gathered at the query point. We present detailed operations of the proposed protocol.

Overlay of RPs: The set of RPs for time (Δ , ( )Δ) are denoted as R (We drop Δ term as the slots are of the same length.). From the definition of dominating sets, the minimum distance between two RPs must be no greater than three hops. Define $G_o(V \ , E \ )$ as a subgraph of $G(V, E)$ composed of R and non-RP nodes that inter-connect R (i.e., V is a connected dominating set of G). We call $G_o(V \ , E \ )$ an overlay of R .

To construct $G_o(V \ , E \ )$, each node in R sends out broadcast *overlay construction* messages with a *time-to-live* (TTL) field set to 3. The broadcast messages contain a sequent number and the source RP's identifier. A node which receives a broadcast *overlay construction* message from a source RP for the first time, records the previous hop from which it receives the broadcast message and decreases the TTL field by 1. If TTL is positive, it forwards the message to all its neighbors. Otherwise, the message is discarded. A node $\in V$ if $\in R$ or is on a shortest (hop-count) path between two RP nodes. At the end of the procedure, each node keeps a forwarding table like the following:

TABLE I
Forwarding Table at Node v

slot	RP id	next_hop
slot 0	1	1
slot 1	8	4
...	...	...

The th row corresponds to the th slot. The second and third column keep the of an RP node and the *next hop* to that RP node for slot . For example, in Table I, node is a direct neighbor of RP 1 for slot 0. To reach RP 8 for slot 1, it needs to route through node 4. The number of forwarding entries depends on T.

Next, we show that the constructed overlay is connected.

Property 2: If G is connected, $G_o(V \ , E \ )$ is also connected for any $\in \mathbb{N}$.[2]

Proof: Omitted due to page limit. See [18] for details. ■

Query Propagation on Overlays: Consider a query $q(\ , \Delta \ )$ initiated at node u. u can decide locally whether it is an RP for slot . Without loss of generality, let us assume that u is a non-RP node. u looks up its forwarding table and find the first entry that corresponds a R node and can be reached directly by u (s.t. *RP id* is the same as the *next_hop*). u forwards the query message to the corresponding R node. To this end, we say that the message is *on the overlay*.

An on-overlay message is broadcast on the overlay using un-scoped broadcast. Duplicated messages are suppressed by inspecting the sequence number in the query messages. As each RP node keeps its on-overlay RP neighbors and routing information, queries are unicast between any two on-overlay RP nodes. Upon receiving a query message $q(\ , \Delta \ )$ from u for the first time, a RP node stores the id of the previous on-overlay RP node to reach the query point and forward the message to all its other on-overlay RP neighbors.

Data Retrieval: Data retrieval is performed on the reverse routes on the overlay to the query point. Each RP node who receives query $q(\ , \Delta \ )$, sends back its stored data to the previous on-overlay RP node till the information reaches the query point. To alleviate wireless medium contention and facilitate data aggregation, transmission schedules can be devised as in TAG[13], which is beyond the scope of this paper.

B. Analysis of the protocol

Next, we investigate the communication cost of the proposed protocol.

The message overhead to reach an on-overlay node from a non-RP node is $O(\ )$ as a non-RP node is covered by at least one on-overlay RP node. To send a query message throughout the network using reverse path forwarding, it takes $O(|V|)$ broadcast transmissions for a network of $|V|$ nodes as each node only needs to transmit a query message once. On the other hand, the size of minimum dominating sets is bounded by $O(|V|\frac{+ln(D_{min})}{D_{min}})$ [2], where $D \quad _n$ is the minimum degree of $G(V, E)$. We use this as an approximation for the number of RPs and verify it in Section VII. Therefore, the total number of query messages is $O(\quad |V|\frac{+ln(D_{min})}{D_{min}})$. Compared to network-wide floods, which incur $O(|V|)$ messages, our proposed solution greatly reduces the query overhead as the network density grows.

The message overhead for data retrieval is $O(|V|\frac{+ln(D_{min})}{D_{min}})$ (compared to $O(|V|)$ without time-indexed

[2]In fact, there are only T distinct overlays.

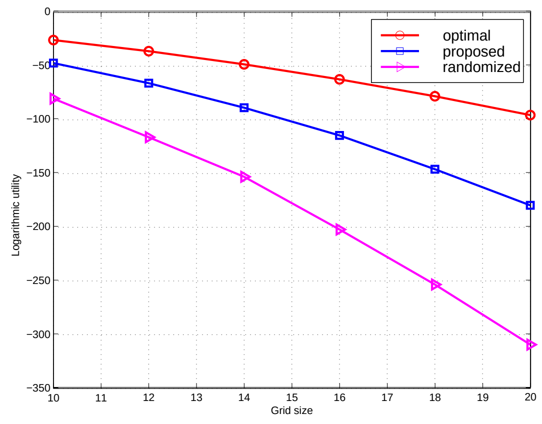


Fig. 8. RP election for finite grid graph, T = 100

storage). From Eq. (1), the average power consumption $_{avg}$ for a slot is $O(|V|\frac{+ln(D_{min})}{D_{min}})(\ _{RP} - \ _{non-RP})$ $|V|\ _{non-RP}$.

VII. Performance Evaluation

In this section, we present simulation results to evaluate: 1) the performance of the RP election algorithm, 2) the communication cost to build time-indexed structure and to process time-indexed queries, and 3) the robustness of the overlay in presence of node failures.

For comparisons, we consider a randomized algorithm for RP election as a baseline. In each slot, it consists of two phases. Initially, each node elects itself to be an RP node with probability in each time slot. After the initial phase, a node volunteers to be an RP node if no other nodes in its neighborhood are RP nodes. Once the RP nodes are elected, the protocol for overlay construction and query processing is the same as that described in Section VI.

A. Performance of RP election algorithm

In this section, we evaluate the proposed RP election algorithm under two different topologies, i.e., grid graphs with variable grid size and random graphs with different size and density. We compare the value of logarithmic utility function achieved by the proposed scheme against that of the optimal solution to the $L\ _{URP}$ problem and the randomized algorithm. Recall that optimal solution to the $L\ _{URP}$ problem gives an upper bound of the $I\ _{URP}$ problem. The results are shown in Figure 8 – Figure 9. In the simulations, random graphs are generated by randomly placing nodes on a 2-D plane. Two nodes are connected by an edge if they are within wireless transmission range.

In Figure 8, the size of grid graph changes from 10 to 20. In Figure 9(a) and (b), we vary the number of vertices (with fixed density) and transmission range (with fixed number of nodes) respectively. In all scenarios, the proposed method performs better than the randomized approach and is much closer to the optimal solution.

B. Performance of Time-indexed Query

In this section, we evaluate the performance of proposed time-indexed query processing protocol using three metrics: i) the number of messages needed to build an overlay; ii) the overhead of propagating a query on the overlay; and iii) robustness of the query processing protocol in the presence of node failures. Unless otherwise specified, all the results presented are averages of ten simulation runs.

Varying the number of nodes: In this simulation, we change the number of nodes while keeping the density of the random graph constant. In the randomized approach, the choice of has an impact on the number of RP nodes elected. For a fair comparison, we consider two possible implementations. In the first implementation, we fix to be 0.1. In the second implementation, we conduct experiments with different settings of and find the one with least number of RP nodes. The resulting is called $_{op}$.

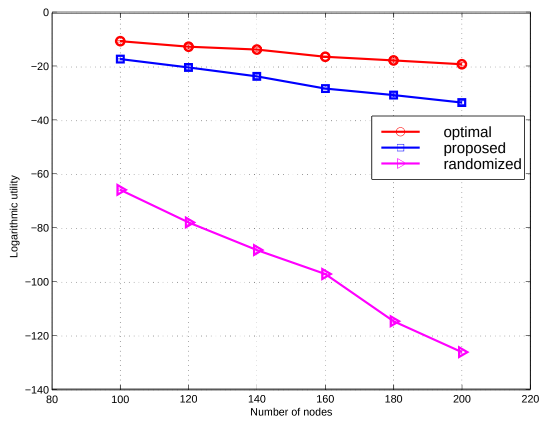
In Fig. 10 (a) and (b) we show the overhead of building an overlay for a slot and processing queries in random graphs with fixed density. We can observe that for all schemes, the overhead of building overlay and processing queries grows approximately linearly with respect to the number of nodes. This corroborates the analytical results in Section VI-B. Our proposed scheme incurs less overhead than the randomized approach with $_{op}$. This is because in our proposed scheme, the RP election is guided by the optimal solution to Problem $L\ _{URP}$ and a voting scheme so that each node is covered by a small number of RP nodes. In contrast, in the randomized approach, each node probabilistically elects itself to be an RP.

Varying the random graph density: In this set of simulations, we fix the number of nodes to be 200 and vary node density so that the number of nodes within the transmission range of a node changes approximately from 20 to 100. In Fig. 11 (a) and (b) we show the overhead of building the overlay and processing queries respectively. Again, we observe that our proposed scheme incurs less overhead than the randomized approach with $_{op}$. The randomized algorithm with $=$. does not work as well in this case because should adapt to node density. The higher the density, the smaller is desired. From Figure 11, we see that as density increases, the overhead of processing query decreases for both schemes as predicted by the analysis in Section VI-B.

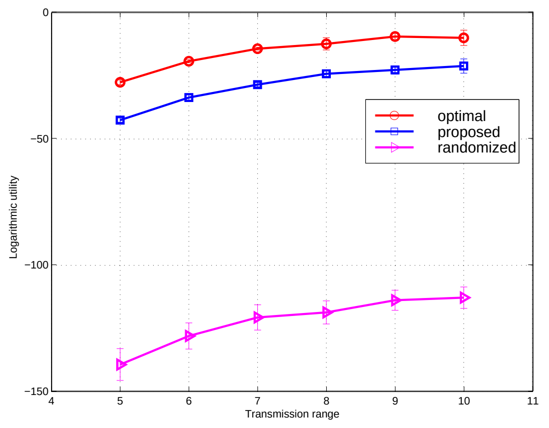
Robustness of connectivity: In this section, we study the robustness of the overlay structure in presence of node failures. As long as the overlay is connected, time-indexed query can be processed on the overlay (possibly with less information gathered). Given the original graph $G(V, E)$, the connectivity of overlay graphs is decided by two factors, i) the number of overlay nodes and ii) the distribution of overlay nodes (whether there are large clusters of nodes sparsely connected to other clusters).

We consider two failure models, i.e., i) random failure where failure nodes are randomly chosen and ii) hotspot failures where all nodes in an area fail. The latter one is to model failures caused by catastrophic events in the network.

Figure 12 gives the size of maximum connected component (normalized by the number of nodes in the remaining overlay) as the percentage of node failures increases under random failure. Under the same percentage of failed nodes, our proposed scheme maintains a larger connected component compared to

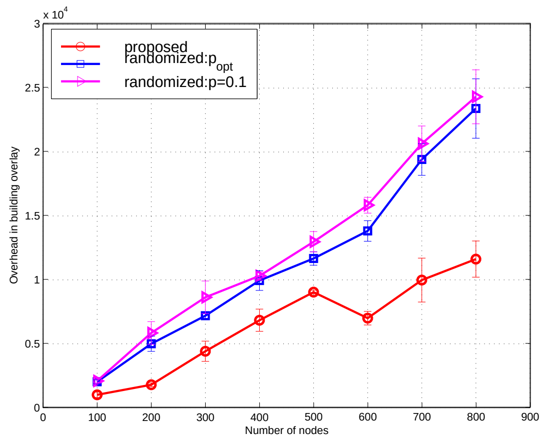


(a) fixed range = 8

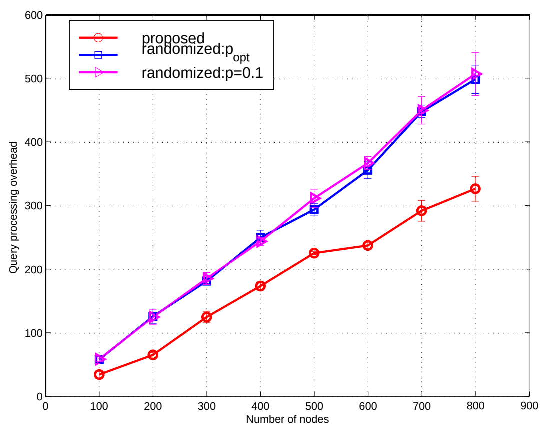


(b) fixed $|V| = 2$

Fig. 9. RP election for random networks on a 40x40 plane, T = 100.

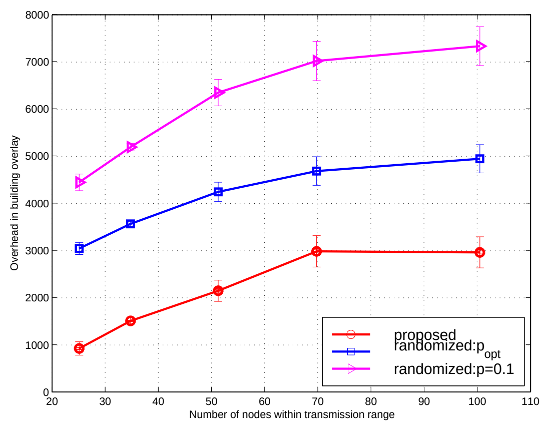


(a) Overhead in building overlay

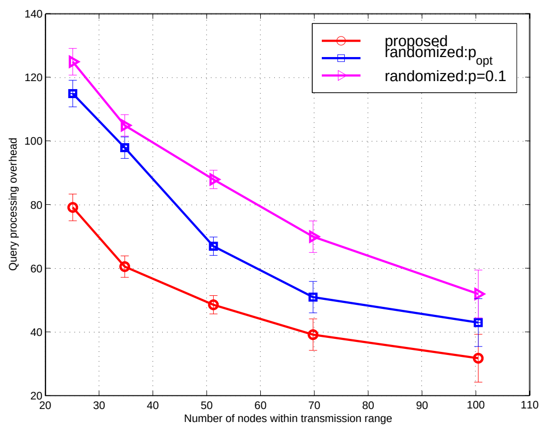


(b) Overhead in processing query

Fig. 10. The overhead of building overlay and processing query in fixed density random graphs, transmission range = 8, $T = 100$.



(a) Overhead in building overlay



(b) Overhead in processing query

Fig. 11. The overhead of building overlay and processing query while the density changes, $|V| = 200$, $T = 100$.

the randomized algorithm. This is because, RPs elected in our proposed scheme are roughly evenly distributed.

In the second set of simulations, we experiment with hotspot failures. Consider a catastrophic scenario where all nodes within a W-by-W square at the center of the sensor field are caused to fail. This essentially results in a hole inside the network. We vary W from 6 to 20, which implies on average, 2% to 25% of nodes in the center of the network failed. From Figure 13, we observe the ratio of the maximum component size in the remaining nodes change from 97% to 70% in our scheme. The randomized scheme is slightly worse. In comparison, most DCS schemes (e.g.[16]) would lost all the data for a particular time slot that is mapped to the hot spot area.

In summary, the connectivity of the constructed overlay gracefully degrades with node failures. Even in presence of

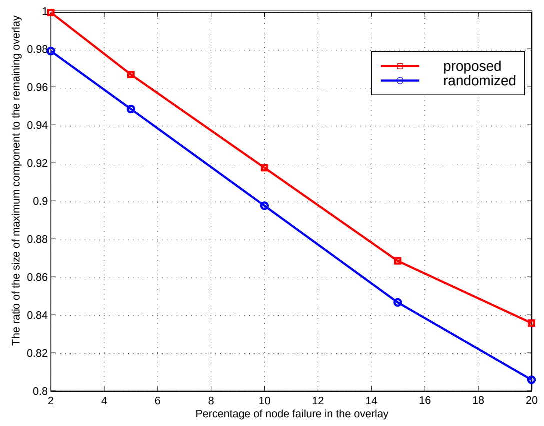


Fig. 12. Ratio of maximum connected component size with random failures.

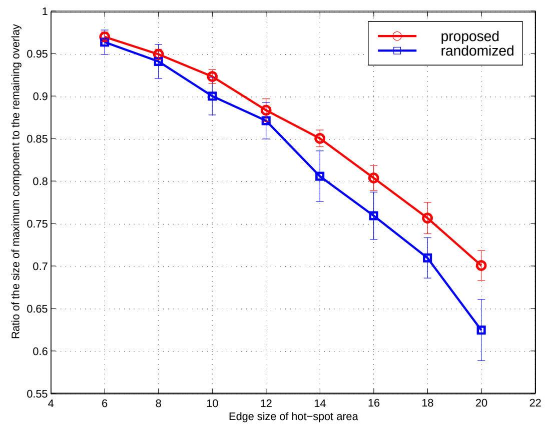


Fig. 13. Ratio of maximum connected component size with hotspot failures.

hotspot failure, time-indexed data can still be retrieved from a large subset of nodes.

VIII. Conclusion and Future Directions

In this paper, we have proposed a comprehensive solution to storing and retrieving time-indexed information in wireless sensor networks. Time-indexed storage is achieved by judiciously selecting rendezvous point (RP) nodes for each time slot and rotating responsibilities among nodes over time. A novel utility-based RP election scheme is devised to trade-off energy utilization with load balancing in the network. Each node is associated with a schedule of length T to serve as an RP. Time-indexed information retrieval is made possible by building overlays for RPs in each time slot. Simulation studies demonstrate that the proposed solution outperforms a randomized approach in terms of communication cost, energy/storage utilization and robustness in presence of node failure. To the best of knowledge, this is the first work that addresses the time indexing problem in sensor networks.

There are several directions that our proposed framework can be extended to facilitate robust storage and information retrieval in sensor networks.

Weighted utility function: The utility function can be extended to weighted forms such as $U(\ _v) = \ _v \quad (T- \ _v)$ where $\ _v \in (\ , \ ]$ can be used to reflect the willingness of a node to be an RP. For example, if a node is running out of battery, it should set a large $\ _v$.

Variable Query Interval: In this paper, we assume the query interval Δ is known and fixed. The RP schedules are decided accordingly. If the query interval is smaller than Δ , the query processing scheme still works. For query intervals larger than Δ , a straightforward solution is to partition it to multiple slots and query multiple overlays. A more efficient solution is to build a hierarchy of RP points with different time resolutions thus queries for a larger interval can be answered more efficiently.

Combination with other attributes: In DCS approaches, events are hashed to sensor nodes using hash functions. One example is to hash events to geographical locations. At the proximity of the designated geographical locations, there are potentially multiple sensors. If time-indexed structure using RP is embedded, queries based on both time attribute and event types can be processed.

References

[1] J. C. Abhishek Ghose, Jens Grossklags. Resilient data-centric storage in wireless ad-hoc sensor networks. In *Mobile Data Management (MDM)*, pages 45–62, 2003.

[2] N. Alon and J. H. Spencer. *The probablistic method*. J. Wiley & Sons, 1992.

[3] K. Alzoubi, P. Wan, and O. Frieder. Distributed heuristics for connected dominating set in wireless ad hoc networks. In *KICS Journal of Communications and Networks*, volume 4(1), March 2002.

[4] S. Athuraliya and S. H. Low. Optimization flow control with newton-like algorithm. *Telecommunication System*, 15(3/4):345–358, 2000.

[5] B. Awerbuch. Complexity of network synchronization. In *Journal of ACM*, volume 32, pages 804–823, 1985.

[6] V. Chvatal. A greedy heuristic for the set covering problem. In *Mathematics of Operations Research*, volume 4(3), pages 233–235, 1979.

[7] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *ICC (1)*, pages 376–380, 1997.

[8] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proc. of Operating Systems Design and Implementation (OSDI)*, 2002.

[9] B. Greenstein, D. Estrin, R. Govindan, S. Ratanasamy, and S. Shenker. Difs: A distributed index for features in sensor networks. In *SNPA*, 2003.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of MOBICOM*, 2000.

[11] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Constant-Time Distributed Dominating Set Approximation. In *Proc. 22nd ACM Int. Symposium on the Principles of Distributed Computing (PODC)*, 2003.

[12] S. H. Low and D. E. Lapsley. Optimization flow control — I: basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, 1999.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. H. ng. Tag: A tiny aggregation service for ad-hoc sensor networks. In *Proc. of Operating Systems Design and Implementation (OSDI)*, 2002.

[14] L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. In *Proc. of IEEE INFOCOM*, 1999.

[15] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets. In *The First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), (Atlanta, Georgia, September 2002).*, 2002.

[16] S. Shenker, S. Ratnasamy, R. G. B. Karp, and D. Estrin. Data-centric storage in sensornets. In *The fi rst ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets 2002)*, October 2002.

[17] P. Sinha, R. Sivakumar, and V. Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. In *INFOCOM (1)*, pages 202–209, 1999.

[18] R. Zheng, G. He, I. Gupta, and L. Sha. Time indexing in sensor networks (full version). UIUC-CS Technical Report No. UIUCDCS-R-2004-2429, 2004.