SURCH: Distributed Aggregation over Wireless Sensor Networks

Xingbo Yu, Sharad Mehrotra, Nalini Venkatasubramanian
School of Information and Computer Science
University of California, Irvine, CA 92697, USA
$\{xyu, sharad, nalini\}@ics.uci.edu$

Abstract

In this paper, we present SURCH, a novel decentralized algorithm for efficient processing of queries generated in sensor networks. Unlike existing techniques, SURCH is fully distributed and does not require the existence or construction of a communication infrastructure. It exploits the broadcast nature of wireless communication to optimize query propagation and evaluation. In SURCH, partial results are aggregated en route while the query spreads through the network. The key features of SURCH include its ability to avoid unnecessary communication, balanced node workload, and resilience to node failures. Performance results illustrate that SURCH outperforms alternative techniques for a variety of aggregation and selection queries.

Key words: wireless sensor networks, data aggregation, response prioritization, query propagation, partial results.

1 Introduction

Emerging sensor devices are capable of sensing, computing and communicating. They are better recognized as peers in a network that serve as points of data generation, processing, delivery, and consumption, rather than passive data sources. Sensor networks have attracted attentions from scientists in various scientific applications such as habitat monitoring [23] and biomedical studies [24]. While some types of applications can only be accommodated on powerful servers with enough processing and storage capacity and a broad and long view of the observed phenomena, there are also numerous applications which do not require such centralized solutions. This paper proposes a framework for implementing data management functionality for queries that arise *within* a sensor network.

Several types of queries have already been studied by sensor database projects such as Cougar [3], TinyDB [20], and Quasar [16]. We can categorize them broadly as *snapshot* queries, *continuous* queries, and *lifetime* queries based on the duration for which they are expected to run. Snapshot queries are asked occasionally by observers monitoring processes of interest. Continuous queries are evaluated periodically. Lifetime queries resemble continuous queries, but with the periodicity of the query evaluation adapted to achieve a lifetime goal. While research for the evaluation of such queries is on-going, most approaches share the same basic architecture of a tree being established via e.g., network clustering, in the sensor network through which information flows to the server(e.g. [22, 20]). This type of framework is characterized by the fact that query dissemination and data collection occur in two completely separated phases. Such a process is popular for long-term monitoring queries with applications such as habitat monitoring, biomedical monitoring, intrusion detection, and agricultural irrigation [23, 24, 13, 25].

An important class of queries which has not received as much attention is what we refer to as a *triggered query*; dealing with such queries is the objective of this paper. Triggered queries are generated and consumed locally within a sensor network, possibly as a result of an external trigger or localized regional event detection. Consider for example, the detection of an anomalous temperature value by an individual temperature sensor. In such a case, a localized group of sensors may cooperate with each other to determine the occurrence or non-occurrence of the anomalous phenomena. A distinguishing feature of triggered queries is the role of *locality*. Here, query results are consumed locally; the evaluation of such a query does *not* require contacting or forwarding data to a central server.

Although tree-based query processing has been shown to work well for monitoring queries in general, utilizing fixed infrastructures for triggered queries is not obvious. In this paper, we propose SURCH (SURfing and searCHing), a fully decentralized peer-based algorithm for processing triggered queries in wireless sensor networks. SURCH combines query dissemination and processing so that a query can be partially processed on the fly in a sensor network. Partial results are delivered to the query initiator or a designated proxy for final processing. SURCH avoids the overheads of infrastructure construction or interaction with a server by exploiting local communication. The novelty of SURCH is four-fold. First, it is capable of processing ad hoc in-network generated queries more efficiently than existing tree-based techniques in addition to continuous monitoring queries. Second, by implementing prioritization, when only a small number of sensor nodes contribute to the query result, SURCH demands very little communication. Third, sensor workloads can be balanced to maximize the overall life-times of sensor networks through carefully designed propagation policies. Finally, SURCH has an inherent resilience to sensor failures, as it does not depend on any particular node, but works "around" failed nodes, discovering the ones that are still operational.

Road Map: The rest of the paper is organized as follows. In section 2, we present the proposed SURCH algorithm for distributed in-network query processing. In section 3, we discuss various optimization policies for several types of aggregate queries. In section 4, we evaluate the performance of SURCH and its optimization policies and compare it with alternative techniques. Section 5 presents related research efforts. We conclude the paper in section 6 together with potential future research directions.

2 SURCH

A sensor node we consider in this work is composed of a processor, an embedded sensor, an A/D converter, and a radio, as with a Mica Mote [14]. Each of these components is controlled by a micro operating system [11] with an external clock enabling synchronization of neighboring nodes [7]. Built-in radio transceivers [1] provide wireless communication based on fundamentally broadcast. While messages may be addressed to specific nodes, all neighbors of a transmitting node can receive the transmitted message at no extra cost [19].

In designing SURCH, we distinguish two types of queries. *Exemplary aggregate queries*(EAQ), such as Min, Max, and top-k, can be processed without necessarily responses from all sensor nodes. On the other hand, Sum, Avg and selection queries require readings from all relevant sensor nodes. We describe SURCH for Max and Sum queries as representatives of the two categories.

2.1 Overview

Instead of gathering data to a specific node, SURCH disseminates a query to a network, processing it during propagation before collecting partial results for final processing.

[1]We assume single channel communication in designing SURCH, although multi-channel communication is available for new generation of MICA2 motes [1]. Optimal channel allocation to improve communication efficiency is beyond the scope of this paper.

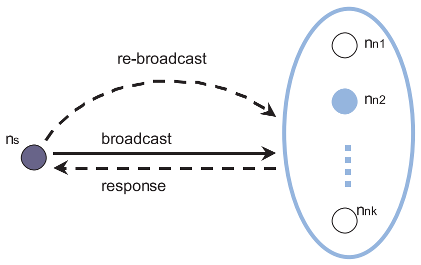

Figure 3. Discover a neighborhood

SURCH starts from a query initiator - the node that triggers a query. The node performs a local processing before proceeds by propagating the query level by level, carrying a current intermediate result. A node that hears the query may decide to serve as a broadcasting/aggregation node. We denote a node to be a *broadcasting node* when it decides to propagate the query and use it interchangeably with an *aggregation node* when the node also takes the responsibility of aggregating. Local processing is repeated at the broadcasting node. Query results are aggregated on the path of query propagation. When propagation stops after the entire query region has been searched, partial results are identified and sent to a destination node which computes a final result that can be used for an upper-level response.

Each node stores information about current queries in a table with tuples containing five attributes $\{queryID, queryType, tempAns, level, parentID\}$. The $queryType$ is used by a node to determine how to respond to the query. *TempAns* is a local partial query result known to the node. *Level* number represents the depth of query propagation, with query initiator at level 0. *ParentID* helps identify a parent node from which the query comes for the first time. A tuple can be deleted after the corresponding query is processed. Focusing on Max queries first, the following sections describe major components of SURCH: local query processing, query propagation, and partial result capturing.

Our discussion has been so far based mostly on perfect network conditions. However, there are practical issues that have to be resolved before the algorithm can be implemented. Due to space limitations, we refer readers to our technical report [27] for detailed discussions.

2.2 Local Processing

The purpose of local processing is to query neighboring nodes to produce a temporary query result. In the following discussion, we use *temporary results* to denote the results produced by local query processing. And *partial result* is used to denote the temporary results that are to be captured after the entire query region is explored.

```
PROCEDURE: receive_max(message $m$)
 1. if($m$.isQuery())
 2.   if(first time receiving the query)
 3.     $myLevel$ = $m.level$ + 1;
 4.     $undiscovered\_neighbors$.update();
 5.     set $propagation\_timer$ for optimization;
 6.   else
 7.     $undiscovered\_neighbors$.update();
 8.     if($undiscovered\_neighbors$ == 0)
 9.       $propagation\_timer$.cancel();
10.   if($m.max < myMax$);
11.     set $reply\_timer$ based on myData;
12.   else
13.     $myMax = m.max$;
14.   if($m.Max \geq myMax$)
15.     $is\_propagated$ = true;
16. if($m$.isRebroadcast())
17.   $myMax = m.max$;
18.   $reply\_timer$.cancel();
19. if($m$.isReply())
20.   $myMax = m.max$;
21.   re_broadcast($myMax$);
END PROCEDURE
```

Figure 1. SURCH for Max queries

```
PROCEDURE: receive_sum(message $m$)
 1. if($m$.isQuery())
 2.   if(first time receiving the query)
 3.     $myLevel$ = $m.level$ + 1;
 4.     $undisc\_neighbors$.update();
 5.     set $propagation\_timer$ for optimization;
 6.     reply with $undisc\_neighbors$;
 7.   else
 8.     $undisc\_neighbors$.update();
 9.     if($undiscd\_neighbors$ == 0)
10.       $propagation\_timer$.cancel();
11.     if($hold\_partial\_result$ == true)
12.       reply with $undisc\_neighbors = 0$;
13.       $hold\_partial\_result$ = false;
14.       $is\_propagated$ = true;
15. if($m$.isReunicast())
16.   $mySum = m.sum$;
17.   $hold\_partial\_result$ = true;
18. if($m$.isReply())
19.   $mySum = mySum + m.sum$;
20.   if($m.undisc\_neighbors > max\_undisc\_neighbors$)
21.     $max\_undisc\_neighbors = m.undis\_neighbors$;
22.     $unicast\_destination$ = $m$.sender;
23.   if(no $re\_unicast$ timer);
24.     set $re\_unicast$ timer;
END PROCEDURE
```

Figure 2. SURCH algorithm for Sum queries

For Max queries (see figure 1), a node first broadcasts a query together with a current query result known to itself. Upon receiving such a query, all neighbors who have not received the query will store it and mark itself at level $l + 1$ (see lines 2$\sim$3 of figure 1), where l is the level of the broadcasting node which is also marked as the parent node. Neighbors with data inquired by the query may respond using unicasting (lines 10$\sim$11). Responses are prioritized with priority given to nodes with greater values. Once response messages are received, the broadcasting node will immediately re-broadcast an updated result to prevent other neighbors (that have a value lesser than the one discovered so far) from having to respond to the query (lines 19$\sim$21). Upon receiving the re-broadcasted temporary result, neighboring nodes update their local copies(lines 16$\sim$18). This makes sure the Max value discovered so far gets propagated further along with the query, which helps reduce potential responses in later local processing operations in outer regions. On the other hand, the re-broadcasting message can be avoided if there is no response from any neighbor which indicates that the initially broadcasted message is indeed the best answer. In this case, neighbors become aware of the omission of the re-broadcasting message if they do not receive such a message after waiting for certain amount of time. Figure 3 shows the messages in a local processing initiated by a node n_s. Dotted lines represent messages that may be omitted.

Note that it is likely for a node to receive a query multiple times. Whenever it receives a query message, it updates its undiscovered neighbors (see lines 4 and 7) by removing the nodes that can be covered by this message with its knowledge about radio transmission range and their neighbors' locations. This operation only gives a rough and conservative (over-estimated) estimation of the number of undiscovered neighbors. But it is good enough to help implement optimization policies as discussed in section 3. Most of the pseudo-codes in figure 1 are self-explanatory except lines 5, 11, 14 and 15, which will be discussed next.

2.2.1 Response Prioritization

To implement response prioritization (line 11), a time period has to be identified so that each neighbor can find a responding time in it. We denote this time period $\{t_b, t_c\}$ where t_b is the time instance when the query is received and the interval is considered as a system parameter which is decided based on the network density. This is also the time period that a broadcasting node should wait even if no one responds. Also, each neighbor node should be able to compute its *fitness* value θ, with respect to the query predicate. A fitness value is defined as a measurement on how close a sensor data is to the exemplary value an exemplary query attempts to find. With these information, a node can compute a TTR (time to response) value as following:

$$TTR = f(t_b, t_c, \theta) = \begin{cases} \infty & \text{if } \theta = 0; \\ (1-\theta)\cdot(t_c - t_b) & \text{o.w..} \end{cases} \quad (1)$$

The formulae following are devised so that the responses are in the order of decreasing fitness values, and the responses are dispersed as evenly as possible.

Max/Min **Queries:** In a typical Max query, a broadcasting node asks its neighbors for values larger than its current maximum value, v_c. Assume each neighbor node also knows, besides its own value v, that the upper bound (which can be a loose bound) of the inquired attribute is v_M.

$$\theta = \frac{v - v_c}{v_M - v_c}. \quad (2)$$

$$TTR = \begin{cases} \infty & \text{if } v \le v_c; \\ \frac{v_M - v}{v_M - v_c}\cdot(t_c - t_b) & \text{o.w..} \end{cases} \quad (3)$$

Nearest Neighbor Queries: A typical NN query provides a target value v_0, and the current nearest neighbor value v_c known by the querying node n_s. Neighbors with closer values ($|v - v_0| < |v_c - v_0|$) may respond.

$$\theta = \frac{|v_c - v|}{|v_c - v_0|}. \quad (4)$$

$$TTR = \begin{cases} \infty & \text{if } |v - v_0| \ge |v_c - v_0|; \\ \left(1 - \frac{|v_c - v|}{|v_c - v_0|}\right)\cdot(t_c - t_b) & \text{o.w..} \end{cases} \quad (5)$$

$topK$ **Queries:** A general $topK$ query asks for the nodes whose values are the top K closest ones to a given target value v_0[2]. The broadcast message contains v_0 and v_c, the lowest value among the top K values discovered so far. The computation of θ and TTR is identical to NN queries. However, the broadcasting node may need to wait for several responses before it can decide that the top K answers have been discovered and start re-broadcasting the answers. When to stop receiving responses can be determined by combining the current top K values with the newly received values, together with the observation that the values received from responding nodes arrive in decreasing order in fitness. As an example, suppose a query asks for the top three values that are closest to a target value, and the current top 3 values have fitness of 0.8, 0.5, and 0.3. If the first response returns a value with fitness 0.4, then node n_s can stop listening and start re-broadcasting since the top three values have been discovered with fitness of 0.8, 0.5, and 0.4. In another case, if the first response is a value with fitness 0.6, n_s has to wait for one more response to decide the third element in the top3 list.

2.3 Query Propagation and Partial Result Capturing

A major challenge in devising SURCH is to decide which nodes (a set denoted by N_s) should serve as broadcasting nodes to propagate a query. We observe that the

[2]Maximum $TopK$ query is a specific case with $v_0 = \infty$.

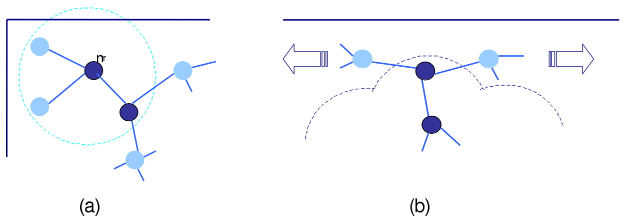

Figure 4. (a) A partial result is captured at node n_r (b) Boundary propagation helps reduce partial results

set of N_s needs to satisfy two fundamental requirements: The query region has to be searched exhaustively; The overall number of messages generated should be small. From graph theory point of view, we are looking for a small size connected dominating set (CDS) of the network to take the responsibility of propagating queries. We explore various solutions to this issue in section 3.

Partial results are the results on sensor nodes that have to be identified and reported to a destination node for final query processing. Unlike with tree-based algorithms in which the root node of an aggregation tree has query result, the nodes holding partial results produced by SURCH is not known a priori. In fact, every node holds a result aggregated along the path from the query initiator to itself. However, not all the nodes need to report their results.

When all communication links are reliable, we observe that partial results must be held by a propagating node as a result of local query processing. In this case, every node which, after performing its local query processing, does not hear any of its neighbors propagating the query holds a partial result that has to be reported. However, when link failures are present (see [27]), a general and practical criteria is needed. – *Any node whose aggregate result is not propagated has to report the result.* This criteria is implemented by lines 14∼15 in figure 1.

Following this criteria, partial results will end up being cornered at some irregular sections at the boundaries or inside a query region. For example, node n_r in figure 4(a) may hold a partial result. Dense and uniform sensor node deployment leads to partial results being captured at query region boundaries. The number of partial results generated depends on node density and the regularity/smoothness of boundaries. Figure 4(b) shows how results are propagated near a regular boundary, which explains a nice feature that normally few partial results need to be captured. On the other hand, sparse or non-uniform sensor implementation may result in larger number of partial results. We further examine this issue in the experiment section.

2.4 *Sum* Queries

Since all the nodes have to respond, in the local processing step, there is no need to prioritize the response messages (line 6) in processing Sum queries. It is also not necessary for all neighboring nodes to store the temporary result discovered by the broadcasting node. Only one node, denoted as a *relay* node, should store and carry the result into further query propagation. In other words, re-broadcasting is replaced by re-unicasting with a destined neighbor node specified (lines 23 and 24). Re-unicasting is not performed until the time window that allows neighbors to respond expires. Any node that has already responded to a query, except the relay node, will not respond to the same query any more. The relay node will only respond once more to deliver the temporary result it holds so as to avoid double counting of the same data items (see lines 11$\sim$13). The goal of sending the temporary result to a relay node is for the result to be aggregated with other results during future local query processing, which helps reduce the total number of final partial results that have to be captured at the end of query propagation. For the best chance to be merged, results are sent to neighbor nodes with the most un-explored neighbors (lines 20$\sim$22). Any nodes that hold partial summation data and have not delivered it to another node must report the partial result (lines 12 and 14).

3 Optimization Policies

As mentioned in the previous section, a small size connected dominating set is desired for efficient query propagation. This CDS has to be discovered on the fly along with query propagation. In this section, we show that with limited knowledge about surrounding network, SURCH can discover small size connected dominating sets for low cost query propagation. The assumption we make is *neighborhood-awareness*. Specifically, a sensor node only has knowledge about its one-hop neighbors' locations. The information comes from local message exchange which only takes place when there is a topology change.

With neighborhood-awareness, a CDS can be discovered by allowing each node to compute a TTX (*time to transmit*, a timer implemented with line 5 in figures 1 and 2) value within a time frame designated for nodes at a certain level. TTX is computed as a function of *priority*, p ($0 \leq p \leq 1$), which covers various optimization metrics as discussed below. Let $\{t_s, t_e\}$ denote the time period in which a node will perform broadcasting. We want to compute TTX as:

$$TTX = f(t_s, t_e, p) = \begin{cases} \infty & \text{if } p = 0; \\ (1-p)\cdot(t_e - t_s) & \text{o.w..} \end{cases} \quad (6)$$

The above heuristic is based on the observation that the order in which nodes broadcast a query is very important.

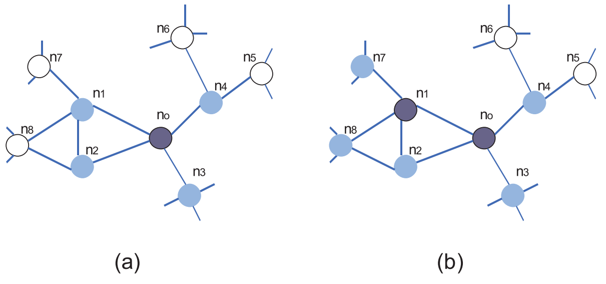

Figure 5. CDS selection

The broadcasting of one node may make it unnecessary for another node to participate in query propagation. For example, in figure 5(a), n_1 has two neighbors (n_7, n_8) which have not been queried and n_2 has one (n_8). After n_1 performs a broadcast as shown in figure 5(b), n_2 will have no un-queried neighbors since n_8 has been covered by n_1. Note in the figures, white nodes denote the ones that have not been queried, gray nodes have been queried, and black nodes are the ones that performed broadcasting (self-selected into a CDS). We would like to point out that prioritization here may not be achieved precisely when priority values are close. The consequence goes no further than the fact that the selection of CDS nodes deviates from what would have been chosen in the ideal case. With the introduction of priority p, SURCH has the flexibility of targeting at various optimization goals as presented next.

3.1 Minimize Overall Power Consumption

In order to reduce the overall cost, we want to obtain a small size connected dominating set, discovered level by level stemming from a query initiating node. We develop below a greedy heuristic which always picks a node with the maximum number of *un-queried* neighbors first. Let d denote the estimated number of un-queried neighboring nodes of a node under consideration. Suppose we have an estimation of the maximum degree (d_0) of any node in the network (or even better, in the neighborhood), then $p = \frac{d}{d_0}$.

To estimate d, a node has to fully exploit its neighborhood-awareness. Note that d and TTX have to be updated as query messages are received because d values change. Since d can only decrease monotonically with time, TTX will be increased and the transmission will be postponed. This is especially important when d is decreased to zero and it becomes unnecessary for a node to broadcast.

3.2 Achieve Uniform Power Consumption

Despite its ability to achieve minimal overall cost, the above technique may cause some sensor nodes (e.g. the ones with more neighbors) to drain out their energy much faster than others, which leads to a short network lifetime. Pursuing uniform power consumption across a sensor network is another important objective in real-world applications. To achieve this, a node has to be aware of its own power level e and compute p accordingly.

$$p = \begin{cases} 0 & \text{if } d = 0; \\ \frac{e}{e_M} & \text{o.w.,} \end{cases} \tag{7}$$

where e_M is a system parameter for maximum node power. Note that only approximate uniformity of power consumption can be accomplished. Network topology still plays the dominant role as shown by our simulations.

3.3 Compound Policy

As expected, uniform power consumption criteria results in larger overall cost than the overall cost criteria. A *compound policy* that balances the two goals may also be useful. A parameter *balance factor*, α $(0 \leq \alpha \leq 1)$, is required to decide the weight of each criteria.

$$p = \begin{cases} 0 & \text{if } e = 0 \text{ or } d = 0; \\ \alpha \cdot \frac{d}{d_0} + (1-\alpha) \cdot \frac{e}{e_M} & \text{o.w.,} \end{cases} \tag{8}$$

4 Empirical Evaluation

We evaluate the performance of SURCH using ns2 [2]. We simulate networks in a 240m$\times$240m query region where nodes communicate using a CSMA-CD based MAC protocol with *TwoRayGround* signal propagation model (similar to IEEE 802.11). The radio coverage distance is set at 40m. To test the performance of the proposed algorithm over different densities of sensor node placement, we vary the number of nodes. Density is defined as the number of nodes per cell, which in turn is defined as the area covered by a circle with radius 40. For other specific evaluations, we use a network with 100 nodes over the 240m$\times$240m space, a commonly used density setup. The nodes are placed randomly on the chosen space. We simulate one query initiated at a randomly chosen node in the entire query region. All our results are averages over multiple runs. Since network topology is the dominant factor in our algorithm performance evaluation, we only work with synthetic data with generated sensor values.

4.1 Experimentation strategies

We compare SURCH with a wide variety of strategies that broadly fall under two alternative approaches mentioned earlier in which data is gathered using a tree structure and aggregated on the way to the collection node. In one approach, query initiator contacts the AP (root of the existing abstract tree) for query processing. The other one constructs a new tree and collects data through it. In addition to naive data collection in which every node processes and reports data to its parent node, we also implemented and compared with optimized versions that use overhearing. Overhearing, when an exemplary query is being processed, allows a node to suppress a message when it overhears that another node has reported a better answer to the query. Table 1 lists the various comparison strategies.

Table 1. Strategies compared in experiments

method	description
surch	Surch, minimizing overall cost
n-tree-oh	construct a new tree, with overhearing
n-tree-nv	construct a new tree, without overhearing
e-tree-oh	use an existing tree, with overhearing
e-tree-nv	use an existing tree, without overhearing
n-tree	construct a new tree, non-exemplary queries
e-tree	use an existing tree, non-exemplary queries

We specifically evaluate algorithms in terms of the communication cost, i.e. *number of transmissions* generated in a network. We use the *average number of transmissions per node* as a normalized measurement of the communication overhead in the network. We account for a transmission being a single communication operation that transfers one packet of data from a sensor node to adjacent sensor nodes in one hop. We assume all messages have similar packet sizes. Although this may not be true for $topK$ and Sum, it still allows fair comparison across different approaches.

4.2 Results

Exemplary queries: Figure 6 shows that with increasing node density, the average costs per node decrease. For Max queries at a density comparable to what is considered typical, which is around 10 nodes/cell, about 0.4 per node transmission is achieved, which is about 50% improvement over the best alternative approach n-tree-oh. Note that the cluster structure we use in the simulation for tree-based data collection is constructed the same way as a CDS is discovered in the SURCH propagation phase.

Figure 7 illustrates overheads for $Top5$ queries. Although the relative costs are similar to Max queries, the absolute cost is slightly higher since there is a smaller chance of suppressing a message in this case. About 0.45 per node transmission is achieved for normal network density setup, which again shows the superiority of SURCH.

Non-exemplary queries: We also implemented $Select$ and Sum queries as an example of non-exemplary (Avg, Sum, $Select$) queries. Figure 8 shows that when a small portion (10%) of nodes need to participate in summation, SURCH achieves a performance as good as in exemplary queries. Note that in non-exemplary queries, there

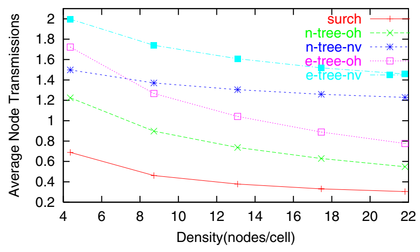

Figure 6. Communication cost for Max queries

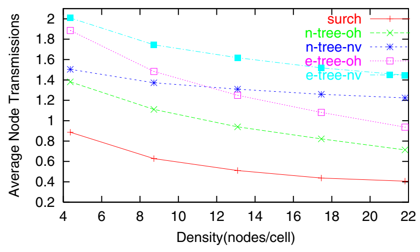

Figure 7. Communication cost for $Top5$ queries

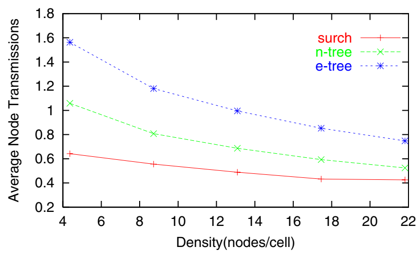

Figure 8. Cost for selective Sum queries

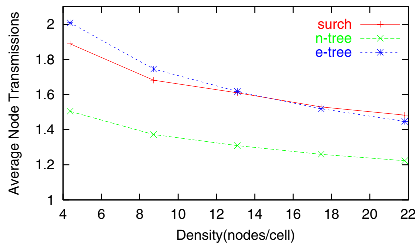

Figure 9. Communication cost for Sum queries

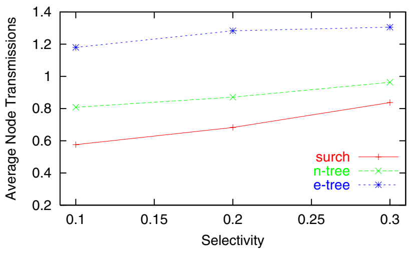

Figure 10. Cost over selectivity for selection queries

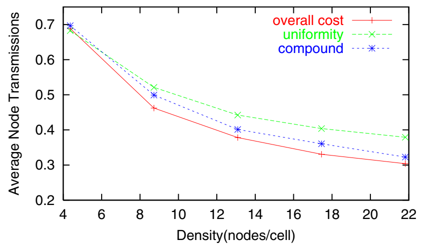

Figure 11. SURCH optimization policies for Max

is no overhearing optimization available for tree based approaches. For full summation, figure 9 shows that constructing a new tree and collecting all data to the root is a better option. Hence, combining new-tree construction with SURCH depending on the query tasks and network conditions can be exploited for processing triggered queries. Figure 10 shows results for other selectivities with fixed network density where costs increase with selectivity.

Impact of optimization: Figure 11 shows costs of processing Max queries under different policies with the typical node density. The results are as predicted, with overall cost criteria yielding the lowest cost and uniformity criteria yielding the highest cost. A compound policy (with $\alpha = 0.5$) gives an intermediate cost. The communication cost for each node (not shown due to space limitations) shows that the minimum overall cost policy does result in more biased cost distribution across nodes than compound and uniformity policy. However, uniformity policy does not produce even cost on all the nodes either, since topology plays the dominant role in determining individual node costs. Tests with Sum queries exhibits similar results.

Summary of other experiments: We extended SURCH to handle unreliable communication links. Our experiments show that to cope with faulty links, higher cost is required to implement the algorithm. We also recorded the numbers of partial results generated by Max queries under various link failure rates. The number generally increases with network density and with failure rates. This is because that failed links cause more partial results to be trapped at some nodes and has to be reported. Sum and selection queries produce similar amount of partial results.

Our experiments on the effect of sensor data distribution show that when the maximum value can be discovered in early steps of Max query processing, the overall cost is very small. Experiments on scalability show that the transmission cost does not change when network size increases. The number of partial results increases sublinealy. Experiments on *latency* show that the average numbers of hops to reach a partial result increase almost linearly.

5 Related Work

Existing work in sensor databases includes the TinyDB project [19, 21, 10] and the Cougar project [3, 26]. In [19], Madden et al. investigated problems encountered in developing TinyDB for a Habitat Monitoring application, from semantics, query dissemination, data collection, to query optimization based on sensor energy level and query lifetime requirements. Madden et al. also studied the implementation of aggregation as part of system services [20, 22]. In [6], they extended data acquisition to map raw sensor data into physical phenomena through statistical modeling. [26] explored failure-resilience issue in addition to packet merging and optimization.

Data collection and aggregation in sensor networks have also been studied in the networking community. Leach [9] and Pegasis [18] explore cluster-based data gathering protocols that rotate cluster-heads randomly to evenly distribute workload among sensor nodes. [15] takes a similar approach and targets uniform energy dissipation to achieve maximum network lifetime. Directed Diffusion [12] provides a data dissemination paradigm in which a node requests data by propagating interests to find a group of interesting nodes. Aside from aggregation, power-aware routing techniques [**?**] have been studied to minimize the cost of intermediate routing nodes in ad-hoc networks. Research work in network clustering [17, 4] includes a theoretical analysis on graph clustering using dominating sets [8, 5].

6 Conclusions and Future Work

We studied the problem of processing triggered queries in wireless sensor networks. We presented a decentralized algorithm to process queries during their propagation in networks. Experimental results show that SURCH is very efficient for exemplary aggregate queries and selection queries with low selectivity. Another advantage of the algorithm is that it deals with sensor failures elegantly.

As part of our future work, we plan to enhance the algorithm with sensor state management protocols and query batching techniques. We also plan to explore quality-awareness in in-network query processing using SURCH.

References

[1] Berkeley mica2 mote. *http://www.xbow.com/Products/*.

[2] The network simulator - ns-2. *http://www.isi.edu/nsnam/ns/*.

[3] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *2nd International Conference on Mobile Data Management*, 2001.

[4] M. Chatterjee, S. K. Das, and D. Turgut. Wca: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing*, 5:193–204, 2002.

[5] Y. Chen and A. Liestman. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In *MobiHoc*, 2002.

[6] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[7] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *IPDPS Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, 2001.

[8] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4), 1998.

[9] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, 2000.

[10] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *IPSN*, 2003.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. In *ASPLOS*, 2000.

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

[13] O. Kachirski and R. Guha. Effective intrusion detection using multiple sensors in wireless ad hoc networks. In *HICSS*, 2003.

[14] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for 'smart dust'. In *MOBICOM*, 1999.

[15] K. Kalpakis, K. Dasgupta, and P. Namjoshi. Maximum lifetime data gathering and aggregation in wireless sensor networks. In *NETWORKS*, 2002.

[16] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. V. Kalashnikov, and W. Yang. Quasar: Quality-aware sensing architecture. In *SIGMOD Record*, 2004(to appear).

[17] C. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7), 1997.

[18] S. Lindsey and C. S. Raghavendra. Pegasis: Power efficient gathering in sensor information systems. In *Proceedings of IEEE Aerospace Conference*, 2002.

[19] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[20] S. Madden, M. F. J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[21] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *SIGMOD*, 2002.

[22] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Workshop on Mobile Computing and Systems ApplicationsS*, 2002.

[23] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA*, 2002.

[24] L. Schwiebert, S. K. Gupta, and J. Weinmann. Research challenges in wireless networks of biomedical sensors. In *The seventh annual international conference on Mobile computing and networking*, 2001.

[25] S. Singh and C. S. Raghavendra. Pamas: Power aware multi-access protocol with signalling for ad hoc networks. *ACM Computer Communications Review*, 1999.

[26] S. J. Thomson and B. B. Ross. Using soil moisture sensors for making irrigation management decisions in virginia. In *http://www.ext.vt.edu/pubs/rowcrop/442-024/442-024.html*.

[27] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[28] X. Yu, S. Mehrotra, and N. Venkatasubramanian. Surch: Distributed query processing over wireless sensor networks. In *Technical Report TR-DB-05-08, ICS, UC Irvine*, 2005.