Data Gathering Tours in Sensor Networks

Alexandra Meliou *, David Chu *, Carlos Guestrin †, Joseph Hellerstein *, Wei Hong ‡
* University of California, Berkeley
† Carnegie Mellon University
‡ Arched Rock Corporation
{ameli,davidchu,hellerstein}@cs.berkeley.edu, guestrin@cs.cmu.edu, whong@archedrock.com

ABSTRACT

A basic task in sensor networks is to interactively gather data from a subset of the sensor nodes. When data needs to be gathered from a selected set of nodes in the network, existing communication schemes often behave poorly. In this paper, we study the algorithmic challenges in efficiently routing a fixed-size packet through a small number of nodes in a sensor network, picking up data as the query is routed. We show that computing the optimal routing scheme to visit a specific set of nodes is NP-complete, but we develop approximation algorithms that produce plans with costs within a constant factor of the optimum. We then enhance the robustness of our initial approach to accommodate the practical issues of limited-sized packets as well as network link and node failures, and examine how different approaches behave with dynamic changes in the network topology. Our theoretical results are validated via an implementation of our algorithms on the TinyOS platform and a controlled simulation study using Matlab and TOSSIM.

Categories and Subject Descriptors: E.1, F.2.0, G.2.2

General Terms: Algorithms, Theory

Keywords: Sensor Networks, Routing Algorithms, Splitting Tours

1. INTRODUCTION

In this paper, we consider a basic task in sensor networks: gathering data from a subset of nodes. This problem arises in interactive scenarios, in which a user or algorithm running at a base station requests readings from an explicit subset of the nodes in the network. The choice of nodes – and the sensors on those nodes – may be made manually based on knowledge of the sensor placement and properties. The choice may also be made in software: The BBQ system, for example, proposes model-driven querying schemes for sensornets [10], in which an optimization process chooses the set of nodes and sensors to sample in order to approximately answer a high-level SQL query.

The standard approach to interactive data gathering uses a two-part protocol: query flooding from a basestation, followed by an incast of data from the sensors via a network spanning tree [21]. This approach makes sense in scenarios where all or most of the nodes need to participate in a query. In some cases, however, the set of desired readings is small, and only a small subset of nodes need to participate in answering the query. The combination of flooding and tree-based result routing is ill-suited to these scenarios.

IPSN'06, April 19–21, 2006, Nashville, Tennessee, USA.

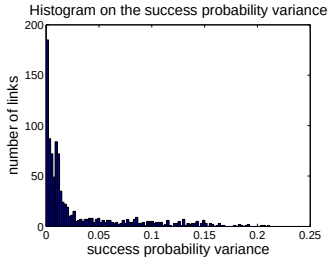

Figure 1: Histogram of the variance of the success probabilities of all network links.

Network connectivity in a wireless sensornet can be highly unpredictable, but in many deployments the sensor nodes are fixed in space, and the communication links between the nodes do not demonstrate extreme variation over time – this is the case, for example, in an office environment like Intel's Mirage sensornet testbed [1]. In these cases the graph representing the network can be considered *semi-static*. Although the link quality of an edge demonstrates variations over time, its distribution is practically stationary ([25]) — its statistical properties do not change much over time (thus our use of the term semi-static). To support this assumption, we analyzed connectivity data from an indoor network of 41 nodes collected every 2 minutes, for a period of 20 hours. Figure 1 presents a histogram of the variance of the link qualities. Most links demonstrate very low variance, which shows that the semi-static assumption is reasonable.

In such situations the properties of the network links – e.g., the expected number of retries required for pairs of nodes to communicate – can be easily measured by the nodes and periodically propagated to the basestation. By taking advantage of this knowledge, we can develop more sophisticated query routing schemes, where the most efficient communication path is decided at the basestation, which uses source routing to move the query through the network. However, we stress that while the *cost estimates* of such an approach may rest on semi-static properties of the network, the actual routing behavior cannot: transient node and link failures must be handled robustly, even in static deployments in which they are relatively infrequent.

In this paper we study the algorithmic challenges lurking behind the apparently simple problem of selective data-gathering in a semi-static sensor network. Our contributions include the definition of a *base-to-base, source-routed data gathering protocol* that constructs small tours of nodes in the network, starting and ending at the basestation. Each tour combines the tasks of propagating a fixed-size query packet with collecting the requested data: as the query packet progresses through the network, the indicated readings are written into the packet, which eventually returns to the basestation. We achieve our tours via source routing: the basestation uses its knowledge of the network to choose an optimal route for each fixed-size packet, with

the final hop of the route being back at the basestation.

Our theoretical contributions include the proof of NP-completeness for our query-routing problem, as well as the development of polynomial approximation algorithms that produce tours within a constant factor of the optimum. Finally we enhance the robustness of our initial algorithms to accommodate the practical issues of limited-sized packets as well as network link and node failures, and examine how different approaches behave with dynamic changes in the network topology. Our theoretical results are validated via an implementation of our algorithms on the TinyOS platform and a controlled simulation study using Matlab and TOSSIM [18].

1.1 Related work

Our work addresses a problem in the BBQ query system [10], where the authors describe a method of reducing query cost using probabilistic inference. The presented algorithms derive a subset of the network nodes that are sufficient to answer the query within some specified confidence intervals. Our work in this paper focuses on computing the optimal communication path for retrieving the measurements from this subset. It should not be assumed however that the applicability of this work is restricted to the framework of [10]. Many applications that rely on selective data gathering could benefit from the theory presented in this paper (e.g., multi-resolution storage [11]). We make the assumption that the basestation possesses information about the entire network topology, which is assumed semi-static. The sensor nodes are not required to maintain any routing information, not even for their immediate neighbors.

A wide range of routing protocols have been proposed for wireless sensor networks, and many of them could be used for selective data gathering. Conventional protocols like flooding or gossip [14] spend a lot of bandwidth and energy on unnecessary transmissions. The tradeoff between energy and latency has also been a topic of study ([26]). In this work however we do not include latency as a part of the optimization process. Also, we do not make any assumptions about data correlations as is the case in [24, 7, 8].

The SPIN protocol proposed in [15] and [17] disseminates the data in each node, so that a user posing a query anywhere in the network can immediately get back results, assuming that all nodes keep neighborhood information. In [16] Intanagonwiwat et. al. propose an aggregation paradigm called directed diffusion. This is a data-centric approach that sets up gradients from data sources to the basestation, forming paths of information flow, which also perform data aggregation along the way. Rumor routing ([3, 4]) also creates paths using a set of long lived agents who direct the paths towards the events they encounter. More specific to query-centric routing are DIM [19], where indices are embedded in the sensor network, and semantic trees [20], where trees are constructed in consideration of the query. GHTs [23] also focus on data centric routing and storage, mapping IDs and nodes to metric space coordinates.

Since the nodes in our framework have no knowledge of the topology, we will propose a packet structure for injecting routing information in the network. This approach makes the problem very similar to the capacitated vehicle routing problem [22, 5, 13]. In capacitated vehicle routing, there exist nodes in a graph that contain an item of a specified volume (analogous to our "measurement set" in Section 2). The items need to be picked up by a vehicle (a packet) of a certain capacity and transferred to another node (our basestation). The capacitated vehicle routing problem is to find the minimum cost tours that the vehicles need to make in order to transfer all items. The main difference of this problem with our case is that the packets (vehicles) are required to carry the routing information as well as the data, and packets can be copied mid-tour while vehicles cannot.

2. THE OPTIMIZATION PROBLEM

In our setting we have a semi-static sensornet, and we need to gather data from an explicitly enumerated set of nodes R, which we refer to as the *measurement set*. We assume that there is a powered basestation computer that we will also refer to as the *root* of the network, where the data are gathered.

The network is modeled at the basestation as a graph $G(V, E)$, where V is the set of all nodes and E represents the radio communication links between them. Each edge is characterized by a cost function $c(i, j)$ representing the expected number of transmissions required to send a message successfully over link (i, j). The cost function is modeled as $\frac{1}{p_{ij}p_{ji}}$, where p_{ij} is the probability that node i will successfully communicate with node j on a given trial. The choice of an undirected model was meant to capture the requirement of receiving an acknowledgement for every message (even if a message is successfully received, the transmission is not considered successful until the sender gets an ack). The same approach was taken in [25] and proposed in [9]. This approach results in an undirected cost graph ($c(u, v) = c(v, u)$), but it does not imply symmetry on the link layer.

The graph model of the network is maintained at the basestation by periodic propagation of link quality measurements. Given the network graph G and measurement set R, the goal of our optimization problem is to compute a minimal-cost routing scheme that visits all the nodes in R and brings their data back to the basestation. The communication path can include nodes that don't belong to the set R and operate in the plan only as routing nodes. This optimization is most naturally solved at the basestation. We therefore adopt a source routing approach, in which the source of the fixed-size query packets (the basestation) marks them with sufficient information to allow nodes in the network to follow the route. In Section 4 we elaborate on the mechanics of annotating a packet with source-routing information; for our expository purposes in this early discussion we can simply assume that (a) some space in the packet is used to instruct nodes how to acquire data and forward the packet appropriately, and (b) space is available in the packet to store the acquired data from nodes in R as the packet makes its way through the network. Because we use source routing, we do not require nodes to maintain routing or connectivity tables for our purposes.

2.1 Optimal communication path

Most traditional techniques divide the actions of query dissemination and data gathering into two separate phases. In the scheme that we are proposing, these two phases are combined, and are executed together, along the same communication path. The communication path that we will compute can also be represented as a graph $G_s(V_s, E_s)$ where $V_s \supseteq R$ (R the measuring set), and E_s is a multiset of edges $(u, v) \in E$ and $u, v \in R$. The existence of an edge (u, v) in G_s indicates that a message will be sent from node u to node v. Note that G_s is directed.

For G_s to be a valid solution to our problem, it needs to be a strongly connected graph. This means that there should be a path from every node to every other node. We refer to a graph G_s that satisfies the above condition as a *Splitting Tour*, in contrast to a traditional graph-theoretic tour which is a simple path that begins and ends at the same node. A splitting tour is a "tour" that is allowed to split and merge along the way (e.g., Figure 2).

The fact that G_s is strongly connected guarantees that all nodes in the communication path are able to both receive the query and deliver the results. A necessary condition for this is that every cut in the graph is of minimum size 2.[1] To see this, first observe that a cut of size 0

[1]The size of a cut (V_A, V_B), where $V_A \subseteq V_s$ and $V_B = V_s - V_A$, is the number of edges $(u, v) \in E_s$ where $u \in V_A$ and $v \in V_B$, or

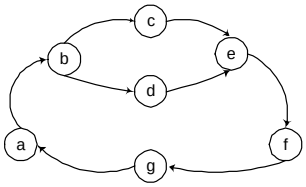

Figure 2: A splitting tour. The tour splits at node b and follows two separate paths which merge at node e.

would indicate a disconnected graph. Now assume there was a cut (V_A, V_B) of size 1, and suppose the basestation was a node $r \in V_A$, then there would be no way of sending the query to nodes in V_B and retrieving the answers, because of the single edge connecting V_A and V_B. (Remember that G_s is directed, so using a physical link in both directions counts as two separate edges in G_s.)

The above observation indicates that a necessary condition for G_s to be a splitting tour is that the undirected version of the graph is *2-edge connected*.

DEFINITION 1 (2-EDGE CONNECTED GRAPH). *A graph is 2-edge-connected if the removal of any 1 edge leaves the graph connected.*

Notice however that a splitting tour represents a communication pattern, and as such it should be allowed to use an edge more than once (a node can receive and transmit on the same link at different times). This means that the splitting tour can in general be a multigraph: a graph $G(V, E)$ where E is a multiset, and hence there can be multiple edges between each pair of nodes. We define a generalization of a 2-edge connected graph which takes this fact into account.

DEFINITION 2 (2-EDGE CONNECTED MULTIGRAPH). *A 2-edge-connected multigraph is a multigraph $G(V, E)$, where $\forall e \in E$ the graph $G'(V, E - \{e\})$ is connected.*

Our goal is to find the most efficient communication path in the network, that visits all of the nodes in our measurement set. This means that we need to find the graph G_s (splitting tour) with the minimum total cost, as defined by the sum of its constituent edge costs. We made the observation that, by definition, the undirected version of a splitting tour is a 2-edge connected multigraph. As the following lemma states, the converse is also true.

LEMMA 1. *G is a 2-edge connected multigraph if and only if there exists a direction of its edges that results in a splitting tour.*[2] □

We want the splitting tour G_s with minimum total cost. From Lemma 1 we see that the problem that we need to solve is equivalent to finding the 2-edge connected multigraph with minimum cost.

2.2 Hardness

We now assess the hardness of finding the minimal-cost splitting tour of a graph. We will prove the following:

THEOREM 1. *Computing the minimum cost splitting tour of a graph $G(V, E)$ is NP-complete.* □

As we know from Lemma 1, finding the min-cost splitting tour is equivalent to finding the min-cost 2-edge connected multigraph that spans all the nodes in the measurement set R. From now on we will refer to this graph as 2-edge-connected multigraph *embedding*, to emphasize the fact that it is constructed from another graph (G).

The instance of the problem that we are required to solve is the following:

Minimum cost 2-edge connected multigraph embedding (2ECME)

- **Instance:** Graph $G(V, E)$, cost function $c(u, v)$ representing the cost of the edge (u, v), integer B.
- **Question:** Is there a 2-edge-connected multigraph embedding $G' = (V, E')$ of $G = (V, E)$ with $\sum_{(u,v)\in E'} c(u, v) \leq B$?

We will prove that 2ECME is NP-hard. To do this, we will use a reduction from the minimum k-edge connected subgraph problem, which is known to be NP-complete [12]. The minimum k-edge connected subgraph problem is stated as follows:

- **Instance:** Graph $G(V, E)$, positive integers $k \leq |V|$ and $B \leq |E|$.
- **Question:** Is there a subset $E' \subseteq E$ with $|E'| \leq B$ such that $G' = (V, E')$ is k-edge connected?

This problem is NP-complete for $k \geq 2$. We will concentrate on the case of $k = 2$ and we will refer to this problem as 2EC.

In 2EC, the solution is the spanning 2-edge connected subgraph of G with the minimum number of edges. The difference between 2EC and the 2ECME problem is that the second minimizes the total weight of the graph and allows reuse of edges (i.e., an edge from the input graph can appear twice as 2 different edges in the result).

Using a reduction from 2EC, we can prove the following:

THEOREM 2. *The 2ECME problem is NP-hard.* □

Using Theorem 2 it is now easy to prove Theorem 1.

3. APPROXIMATIONS

Finding the optimal splitting tour is an NP-complete problem and computing the exact solution is computationally expensive. We need an approximation algorithm that runs in polynomial time. It is natural as a first step towards this goal, to examine the connection this problem has with a very similar graph problem, which is well studied in the literature: the Traveling Salesman Problem (TSP). The TSP produces "simple" tours that do not have splits, which makes it a special case of the splitting tour. Despite the fact that the TSP problem is also NP-complete, it is very well-studied, with many known approximation algorithms, which can give us insight for a solution to our problem.

3.1 Bounding the Minimum Splitting Tour with the TSP

We intend to provide a polynomial approximation of the Minimum Splitting Tour Problem (MSTP) by examining its relationship with the TSP. We wish to provide a constant factor bound for our approximations, so we will start by proving that the solution for the TSP is bounded by a constant factor of the solution of the MSTP.

Since the communication path is required to span only the measurement set $R \subseteq V$, we can transform the original network graph to $G_R(R, E_R)$ which contains only the nodes in R. The set E_R is computed from the original graph G such that each edge $(r, s) \in E_R$ represents the minimum distance path from r to s in G. E_R can be computed in polynomial time by computing all-pairs shortest paths in G. Note that G_R is a complete graph, as long as the original network graph is connected. We will call G_R the *reduced graph* of the network. By definition, since every edge of G_R represents the shortest path in G of the two nodes it connects, the triangle inequality will hold for G_R. The TSP can be solved on G_R and transformed to the equivalent tour in the original graph, by replacing every edge from G_R with the path it represents. [3]

The TSP is a special case of the splitting tour, so it follows that the MSTP solution will be at least as good as the TSP solution, i.e.,

$u \in V_B$ and $v \in V_A$.

[2]For proof of lemmas and theorems see extended version [2].

[3]Note that we allow the TSP tour to visit a node more than once.

$C^{opt}_{MSTP} \leq C^{opt}_{TSP}$. The question that we need to answer is how much worse the optimal solution of the TSP space will be, compared to the solution from the MSTP space. The answer is given by the following theorem.

THEOREM 3. *The optimal solution for the TSP cannot be worse than a factor of 1.5 from the optimal solution of the minimum splitting tour problem (MSTP).*

$$C^{opt}_{TSP} \leq 1.5 C^{opt}_{MSTP} \quad \square$$

This means that the TSP solution bounds the MSTP solution by a constant factor of 1.5.

3.2 A polynomial approximation for the minimum splitting tour

The bound of Theorem 3 does not yet provide us with a good approximation of the minimum splitting tour, because the TSP problem is itself NP-hard. Therefore, we need to provide a bound for a polynomial algorithm, and we will do that for Christofides' approximation algorithm for the TSP with triangle inequality[4], which runs in $O(n^3)$ time [6] and produces a result whose cost is at most 1.5 times that of the optimal tour. Since we will use it later on, a sketch of Christofides' Algorithm is presented in Algorithm 1.

Algorithm 1 Christofides' Algorithm for TSP Approximation

1: Find a MST (Minimum Spanning Tree) T_1. Clearly $C_{T_1} \leq C_{TSP}$
2: Let S be the set of vertices in T_1 with odd degree.
3: Find a minimum weight matching M on S. It is proven in [6] that $C_M \leq \frac{1}{2} C_{TSP}$.
4: Construct an eulerian tour T_2 on the edges of $T_1 + M$. It will be $C_{T_2} = C_{T_1} + C_M \leq C_{TSP} + \frac{1}{2} C_{TSP} = 1.5 C_{TSP}$

Now based on the bounds that we proved for the TSP solution, we will prove a constant factor bound for the TSP approximation. It is trivial to show that since $C^{opt}_{TSP} \leq 1.5 C^{opt}_{MSTP}$ and $C^{approx}_{TSP} \leq 1.5 C^{opt}_{TSP}$, we get $C^{approx}_{TSP} \leq 2.25 C^{opt}_{MSTP}$.

However, we are able to prove that the algorithm provides a better bound, as Theorem 4 shows. The proof consists of applying Theorem 3 to every step of Algorithm 1.

THEOREM 4. *Algorithm 1 provides a factor 1.75 approximation of the Splitting Tour Problem.* $\square$

Therefore, using Algorithm 1 we can compute in polynomial time a simple tour which we know cannot be more expensive than 1.75 times the cost of the actual optimal solution of our routing problem.

4. PACKET SIZE LIMITATIONS

The previous section established a theoretical basis for our problem. However, we have yet to handle a number of important practical considerations. The first, which we address in this section, is the fact that radio network packets are of small fixed size, and source routing instructions for long tours may not fit in a single packet. We begin by describing the specifics of our packet routing implementation in Section 4.1, and then three schemes for dealing with long tours in Sections 4.2 through 4.4.

4.1 Background: Path injection

In Section 2 we discussed in general terms the idea of source routing in a sensor network. Here we provide more detail. We use the simple packet structure shown in Figure 3. The packet header in our implementation includes a sequence number which gets incremented as the packet gets routed around the network, a field indicating the total number of bytes being sent, the ID of the sender, and two additional fields which are used for more advanced packet handling discussed later in the paper.

[4]Notice that although the triangle inequality does not hold for the original network, it does hold for its reduced graph G_R on which all the algorithms are performed.

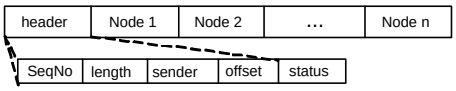

Figure 3: Packet structure.

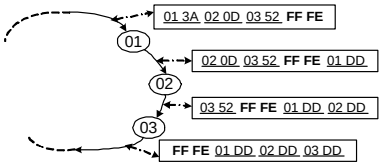

Figure 4: Example of how the packet changes from hop to hop. Two bytes are allocated per node. The first one represents the nodeID and the second holds the necessary data to instruct the node whether it needs to sample or not, how many retries it should attempt for the next hop etc. A byte with the value `0xDD` in the figure represents sampling data stored by the corresponding node in the packet. The bytes filled with the values `0xFFFE` are special delimeters that separate the routing information from the data storage.

The main part of the packet represents a simple path of size n, which should be traversed in the order indicated, from node 1 to node n. Every node in the path is given a slot (in our implementation 2 bytes are assigned to each slot), which serves as the storage space for all the information that needs to be sent to the network. A typical slot entry includes the nodeID, the ID of the sensor to be sampled, and the maximum number of retries to be attempted for the next hop.

In our implementation, the packet works as a cyclic buffer. When a node receives the packet, it removes itself from the beginning and shifts everyone to the left by one slot. Whenever a node in the path receives a message, its node ID should be in the first packet slot. If the node needs to return a measurement it will add it at the end of the packet. If not, it takes no further action than forward the message to the next node in the path whose ID is now placed in the first packet slot. Following this procedure, when the packet comes back to the basestation, it will contain the measurements in the same order as the tour was traversed. This packet structure serves both as a command and as a storage medium, and in order to discriminate between the routing and the measuring part of the packet, special delimeters can be used. An example of how the packet gets routed is given in Figure 4.

In this scheme we move measurements around as the packet travels through the path. Another alternative would be to statically allocate a specific slot for each node. The advantage of the dynamic scheme compared to the static one, is that routing-only nodes – i.e., those not contained in the measurement set – get completely removed after being visited, making the packet shorter. This feature can improve performance for some traversal methods.

4.2 Cutting a tour

Given that background on our path injection scheme, we can now consider the problem of routes that do not fit in a packet.

Let us say that a packet can hold tours of maximum size P, i.e., that the packet has enough space for P node slots, and the tour T_G that we get from Algorithm 1 is bigger that P. One approach is to cut the tour T_G into smaller parts, each of which will have maximum size P. Cuts in the tour are going to be performed by re-routing intermediate edges to the source.

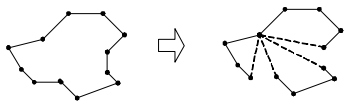

Figure 5: Cutting a tour into smaller subtours.

Algorithm 2 uses dynamic programming (DP) to compute the optimal cutting of a long tour T_G into smaller tours of size $\leq P$, so that each one can fit in a single packet[5]. The algorithm computes a *cost-to-go* function, $J(i)$, that represents the cost of the best cutting of the segment from the ith node to the end of T_G. At completion, $J(1)$ will hold the best possible cost of cutting T_G into parts of maximum size P. We can obtain the optimal cuts with another pass over J in the usual DP fashion.

Algorithm 2 Cutting a tour

```
∀i, J(i) = ∞
J(n) = L(n, n)
for i = n − 1 to 1 do
  for j = 0 to P − 1 do
    if i + j + 1 ≤ n then
      J(i) = min(J(i), L(i, i + j) + J(i + j + 1))
    end if
  end for
end for
```

At the core of the computation is the *local cost function* $L(i, j)$, representing the minimum cost tour that fits in a packet of size P and visits the subset of nodes of T_G that are in the segment from i to j: $S \rightarrow \ldots \rightarrow i \rightarrow \ldots \rightarrow j \rightarrow \ldots \rightarrow S$. The local cost function $L(i, j)$ can also be computed efficiently: We first precompute a hop-restricted distance function, $d(u, v, k)$, representing the cost of the shortest path from u to v that uses at most k hops. This function can be computed by a standard DP. $L(i, j)$ is then obtained by another DP that iterates through the nodes of T_G in the range $[i, j]$, using $d(u, v, k)$ as the local cost function.

This algorithm does not modify the order in which the measuring nodes (nodes of T_G) are visited, but the paths followed in between may differ. This is because a path with fewer nodes which may fit in one packet may not have been picked due to being more expensive in terms of cost. So, there can be cases where the algorithm may not do any cuts at all, and just change the paths followed. For example, consider the tour $T_G = S \rightarrow a \rightarrow B \rightarrow c \rightarrow d \rightarrow E \rightarrow f \rightarrow S$, and the packet size $P = 4$. S is the basestation and the nodes in capital letters form the measurement set $R = \{B, E\}$. It is possible that the cutting algorithm gives a single tour, e.g., $S \rightarrow a \rightarrow B \rightarrow g \rightarrow E \rightarrow S$. This tour may be more expensive than T_G, and that is the reason it may not have been chosen by the TSP approximation, but it does fit in a single packet. Another possible output could be $S \rightarrow a \rightarrow B \rightarrow c \rightarrow S$ and $S \rightarrow d \rightarrow E \rightarrow f \rightarrow S$, where T_G was divided into two smaller tours by simply short-cutting to the root at nodes c and d. The cutting algorithm will dynamically pick the cheapest of the possible choices.

The cost of the main DP algorithm is $O(nP)$. Additionally, we must consider the cost of precomputing the local cost function L. The subfunction $d(u, v, k)$ is computed in $O(n^2P)$. The L function itself is computed in $O(n^3P)$. Thus, the total cost of the cutting algorithm is $O(n^3P)$.

4.3 Multiple packets

As an alternative approach to cutting a tour that cannot fit in a single packet, one can use a "train" of multiple packets to inject the path to the network. We have implemented this approach by using two fields in the packet header that indicate the total length of the packet-train and the current packet's offset in the train. Upon receiving all packets of the train, a node can reconstruct a virtual "big" packet containing the whole path, process it, break it up again into a train and forward it. Notice that even if for some reason packets arrive in a different order, we can reconstruct the proper order by the header information. In every step we treat the packet-train as one big packet. However the cost of sending a packet-train over a link will be proportional to the number of packets it contains. One thing to note is that by using the policy described in Section 4.1, a packet-train can become shorter while getting routed on the path, because of the removal of the routing-only nodes.

4.4 Hybrid: cutting with multiple packets

Simple cutting of a tour does not allow us to reach nodes that are more than P hops away, and forces us to use a small number of (potentially) expensive edges when collecting data from faraway nodes. Multiple packets, on the other hand, can reach faraway nodes, but may be wasteful when collecting data from a large number of nodes. In this section, we use dynamic programming to combine the strengths of these two approaches.

This hybrid algorithm is similar to the cutting DP procedure in Algorithm 2, but instead of restricting cuts to be of length P, a cut can consist of multiple packets which can have a total length up to n. We must also modify the local cost function $L(i, j)$ to allow for the use of multiple packets to visit the subset of T_G that is in the range $[i, j]$. A simple approach for computing the multiple packet version of $L(i, j)$ is to first run the algorithm we used in Section 4.2, setting the packet size to P; then, we run the same algorithm with a packet of size $2P$, computing the edge costs accordingly[6], then for $3P$, and so on. Finally, we define $L(i, j)$ to be the minimum over all of these packet size options.[7] The final computational cost of this algorithm is $O(n^5)$. The paths obtained by the two previous approaches are strictly more costly than this one, since the hybrid algorithm finds the optimal cut that could use one or more packets per section. In Section 6 we will assess the merits of the various schemes in practice on a real network graph.

5. RECOVERING FROM FAILURES

Having dealt with the practical issue of finite-sized packets, we now turn our attention to a subtler practical issue: the dynamics of real networks. For purposes of route selection we assumed that the network is semi-static, but connectivity changes do occur in wireless sensor networks. Link qualities can change and nodes can fail. We want our data gathering approach to remain robust in the face of these events, even if we expect the dynamics of the network to be relatively modest over time.

If after a certain number of retries specified by the quality of the link – a bad link means more retries are required – a node wasn't able to successfully transmit a message, it has to assume a failure of either the link, or the node to which it wants to transmit a message.

To resolve failures we propose two different schemes.

5.1 Backtracking

In this section we will describe the recovery technique of backtracking. Since the nodes do not have any knowledge of the network

[5]If we assume that the nodes in the network know the ID of the basestation, we can exclude the basestation from the packet as an optimization, but this is not a requirement for the algorithms described.

[6]For every path we know which nodes are routing-only and will be removed, so we can pre-compute how long the packet train traversing a specific edge will be. Then the cost of that edge is calculated as the basic cost of transmitting one packet times the number of packets in the train.

[7]We can also use a modified version of the DP algorithm to compute the multiple packet version of $L(i, j)$ more efficiently.

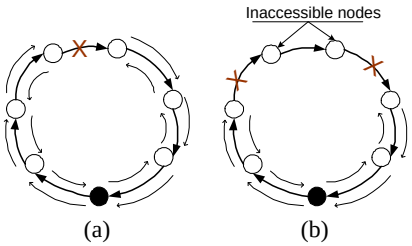

Figure 6: **The bold edges indicate the initially computed tour. (a) During the traversal a failure is encountered and the message backtracks to the root; a new message is issued in the opposite direction than the tour was defined to gather data from the unvisited part. (b) In case of multiple failures nodes can become inaccessible.**

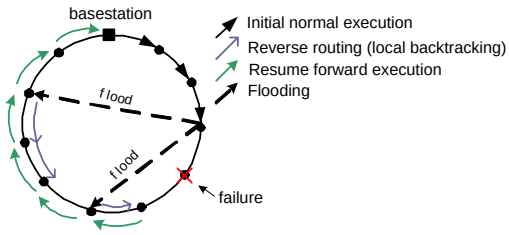

Figure 7: **When a node detects a failure on the path it initiates a flood with small depth, so that it will remain local. The nodes in the unvisited part of the path that hear the flood backtrack on the path to get any data possible between the failure and their position. If a forward and a backtracking message meet, the backtracking one is killed.**

topology, if the path they are given fails, the simplest thing they can do is trace back their steps. When a node encounters a failure, it initiates backtracking which will send the message back to the root with as much data as it has gathered, in the same path that it came from. The information needed for backtracking can temporarily be stored in the network: upon the arrival of a message, the receiving node can store the ID of the sender, just for the duration of the query execution, and then, during backtracking, nodes can use this "breadcrumb" information to traverse the path backwards. When the basestation receives the backtracked message, it can issue a message in the opposite direction of the original tour, to attempt to reach the nodes that were missed in the first round-trip. An example is presented in Figure 6(a).

Notice that in the case of multiple failures happening in a single tour, some of the nodes may remain unvisited like the example in Figure 6(b). In this case, the user who issued the query can be notified about the missing measurements. If this is not acceptable, a new tour can be computed for the missing nodes taking into account the information about the failures the previous run encountered.

The backtracking algorithm that we presented is a simple heuristic to handle a small number of failures in the system. It offers full recovery for single failures per tour, but cannot retrieve nodes that fall in between failures in a path. Notice however that the communication cost of performing recovery with backtracking is bounded by a factor of 2 from the cost of the tour, because every edge of the tour will be traversed at most twice (one during forward processing and one during backtracking).

We note that our description of backtracking assumes that there will be no failures during the backtracking step itself. The assumption here is that an edge traversed in the forward direction should not fail during the running time of the query, during which it may need to be traversed in the opposite direction. In case of such failures though, after a suitable timeout the basestation can re-attempt the packet either directly, or in the reverse direction.

5.2 Flooding

Another approach to recovery is to perform local flooding in case a failure is detected. When a node A exhausts the number of retries denoted in the packet, it will enter recovery mode and broadcast the message in the hope that some node in the unvisited part of the path will hear it. The flooding message contains a TTL (Time To Live) number, which determines the depth of the flood. Upon reception of a flooding message a node B examines it to check whether it is itself part of the path or not. If it is not, it will continue the flood, decrementing the TTL. Flooding terminates if TTL reaches 0.

If B is a member of the yet unvisited part of the path, it can make different decisions as to what it should do with the packet. It can either start sending the packet forward in the path, or send backwards to retrieve any measurements that may lie between nodes A and B, or wait to see if a forwarding message will come from some node preceding B in the path. An example of flooding based recovery is demonstrated in Figure 7.

The more specific semantics of the flooding based recovery scheme in the way that we actually implemented it, taking a conservative approach, are described in the following list.

- During normal execution, a node sends only forward
- When a forward sending fails (after specified retries), a recovery bit is set in the packet, and the node broadcasts the packet. The initiator of the flood is A and the part of the path that is still unvisited is P.
- When a node B hears the flooding message, if B is not in P, and $TTL > 0$, then B continues the flood.
- If $B \in P$, and B heard the flood or a backtracked message during recovery:
  - If no measuring node exists in the path interval (A, B) then B resumes forward execution.
  - Once B has sent a normal-case, forward-directed message, then B will never backtrack.
  - If B has already backtracked then B takes no action for the new flooding or backtracking message (i.e., backtrack only once).
  - If there exists a measuring node in interval (A, B), and B hasn't heard a forward message and B hasn't already backtracked, then B backtracks.
  - If a backtracking message fails (after a specified number of retries) then B resumes normal execution by sending a message forward.

For the last two points of the above list, we chose to follow a conservative approach targeted to the retrieval of as many measurements as we can, without trying to optimize the cost. For example we could possibly have less transmissions if B waited instead of instantly backtracking, because someone else preceding it may have heard the flood and already initiated a forward execution. This would on average decrease communication cost, but it would increase the latency. In this space there is some room for further investigation of the tradeoffs of these parameters, and how they affect the recovery scheme.

Compared to backtracking, a flooding based scheme has the advantage that it can recover from more than one failure in the current tour. A disadvantage however is that the cost (number of messages sent) is not theoretically bounded by a constant factor and depends on the network topology. Also, TTL is a parameter that affects both the cost and the recovered measurements.

6. EXPERIMENTAL RESULTS

We evaluated our proposed schemes via an implementation, consisting of two separate components. The first involves several Matlab

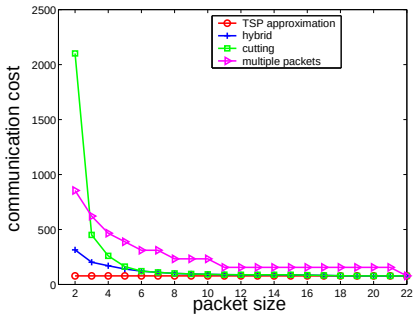

Figure 8: Communication cost of the 3 packet adjustment algorithms. This particular graph corresponds to a measuring set of size 15 in a network of 54 nodes.

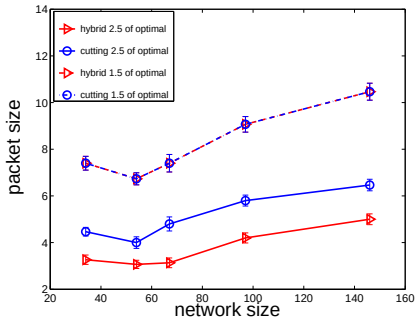

Figure 9: Packet size required for reaching a constant factor of the optimal cost, for networks of different size.

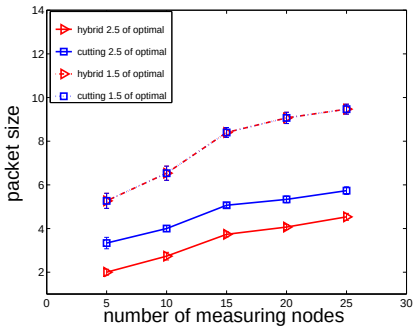

Figure 10: Packet size required for reaching a constant factor of the optimal cost for measuring sets of different size.

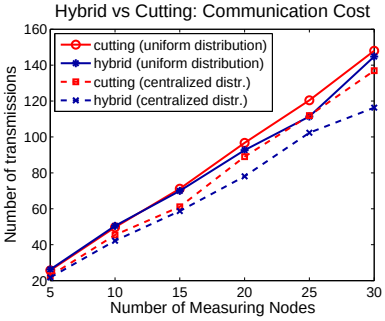

Figure 11: Comparison of the cost of the cutting and hybrid heuristics for measuring sets of various sizes chosen by two different distributions from all the network nodes.

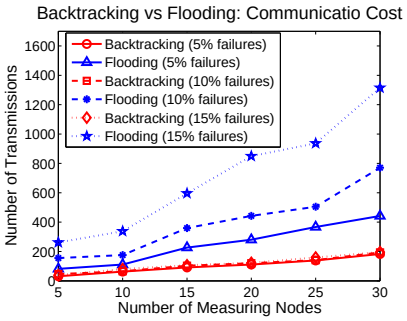

Figure 12: Comparison of the 2 recovery algorithms for failures of rates 5%, 10% and 15% in terms of communication cost. Notice that the backtracking lines practically coincide.

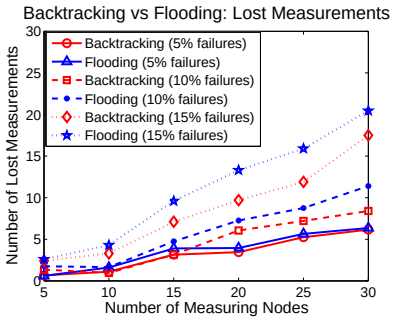

Figure 13: Comparison of the 2 recovery algorithms under conditions of failures with rates 5%, 10% and 15% in terms of the number of lost measurements.

routines used to perform the optimization described in Section 3, as well as to apply the packet size restriction of the network, using the algorithms presented in Section 4. Each tour is stored in a file which is subsequently sent to a Java interface that can parse it and inject the proper packets into the network.

On the network side, our mote code is written in nesC, on the TinyOS platform. This code implements the proper handling of the routing messages, as well as the two different recovery modes, backtracking and local flooding.

For our simulation experiments we gathered connectivity data from a real deployment, through TinyDB queries running for a number of epochs. The connectivity data was given as input to the simulations, thus modeling the dynamics of a real network. We chose to use the public mote testbed at Intel Research Berkeley, which is remotely available via the Mirage resource allocation system [1]. At the time, the testbed consisted of 96 Mica2 nodes at fixed positions[8]. The environment of the deployment is relatively noisy, and includes human activity as well as other radio traffic (802.11, cordless telephone headsets, cellular phones, etc).

6.1 Simulation Results

The results that we present in this section consist of two kinds of simulations. The first are Matlab simulations of the network and algorithms, the purpose of which is to provide an insight as of the behavior of the algorithms under different packet requirements. The second class of experiments uses the actual NesC code for the protocols, but instead of running them on the live testbed we ran them within the TOSSIM simulator, which simulates a network of TinyOS motes[18]. We focused on TOSSIM rather than the live testbed in order to be able to control our experiments and validate their behavior.

Experiments were performed by picking random subsets, as the measuring set, from the real network graph and computing the approximation of the optimal solution proposed in Section 3. The main goal of our analysis is to compare the heuristics that we proposed in Section 4, as well as evaluate our recovery algorithms in the cases of failures.

The cutting and hybrid heuristics adjust long tours so that they can fit within packets of a specified size. Figure 8 demonstrates how the communication cost of the different algorithms converges rapidly to the optimal as the size of the packet increases. The cost for using multiple packets is also depicted in this graph, but it is omitted from our subsequent experiments because it behaved very poorly. More extensive experiments on networks and measuring sets of different sizes demonstrate that the required packet size appears to grow linearly with the network size, as well as the measuring set size, and a relatively small packet is sufficient to achieve a cost close to the optimal.

In terms of comparing the two main heuristics, cutting and hybrid, as expected hybrid demonstrates a lower communication cost. These results are verified by Figure 11 which was produced from experiments on the TOSSIM simulator. The figure compares the packet adjustment heuristics (hybrid and cutting) for two different distributions for picking the measuring set. One of them chooses uniformly from all the network nodes, and the other favors nodes that are positioned closer to the basestation.

[8]These were recently replaced by MicaZ motes.

These TOSSIM runs were performed by using packets of size 30 bytes. In our implementation the packet headers are of 8 bytes length, and each node slot requires 2 bytes. Therefore this corresponds to packets of 11 node slots.

Our approach is extremely effective compared to traditional data gathering techniques when the number of nodes from which we want to get measurements is small compared to the size of the network. For the data depicted in figure 11, the total weight of the minimum spanning tree of the network is 145, where every edge weight represents the number of expected retransmissions on that edge to achieve successful communication. This means that even traversing the minimum spanning tree once would inflict an expected cost of 145 messages.

We also performed experiments for our proposed recovery approach. In addition to the previous setting, we pick a constant number of random failures in the network, and perform TOSSIM simulations for both our recovery algorithms. The TTL used by the flooding recovery algorithm for the graphs that we present in this section is 3. This value was chosen after an evaluation that we did for various flooding depths, which we have omitted here due to space restrictions. For the network that we are modeling [9] in TOSSIM a bigger flooding depth did not add value to the recovery and even started to cause interference phenomena.

Figures 12 and 13 present experimental results for the two recovery approaches, corresponding to a 5%, a 10% and a 15% failure rate in the network. Figure 12 displays the overall communication cost for runs of various sizes for the measuring set. Each point in the graph is an average across 20 different runs of the same measuring set size. As the figure demonstrates, flooding is a more costly recovery technique compared to backtracking. Also, the backtracking cost is more robust to changes in the failure rate, since it is bounded by a constant factor of the cost of the original route, whereas such guarantees do not hold for flooding. In terms of the number of lost measurements, backtracking appears to win again, although the losses for both algorithms increase as the failure rate increases.

6.2 Discussion

Our Matlab experiments demonstrate that the cutting and hybrid heuristics for adjusting long tours to finite packets, converge to the optimal cost very fast, and for relatively small packet sizes. The TOSSIM experiments helped us evaluate our recovery schemes. Backtracking appears to be very robust to failures, with bounded cost and good recovery rates. We can always construct scenarios where backtracking loses to flooding, but in the network that we were simulating, it appears to be the winner. One of the main reasons was the bad quality of the communication links, which gave an advantage to backtracking which utilizes retries. This indicates that probably flooding would benefit from retransmissions (aggressive flooding) which could possibly include some acknowledgement scheme.

7. CONCLUSIONS AND FUTURE WORK

In this work we focused on optimizing the routing paths for data gathering tasks that use source routing. Starting by assuming a semi-static network topology, we defined the optimization problem that we need to solve, and proved that it is NP-hard. We also provided a polynomial time approximation algorithm, for which we proved that the total cost of its solution is bounded by a $\frac{7}{4}$ constant factor of the optimal cost. We presented the packet structure that is used to inject the routing information into the network, and provided algorithms to adjust the communication paths so that they can be accommodated by any specified packet size. Finally we provided heuristic solutions to recover from failures in the network and presented experimental results on the performance of our algorithms.

In our future work we intend to consider additional recovery methods, and hope to provide theoretical guarantees on the cost of the recovery algorithms and the number of recoverable measurements.

Acknowledgments: We would like to thank Andreas Krause for providing the data used in Figure 1. This work was supported by NSF Grants 0326472 and 0205647, and a gift from Microsoft Corporation.

[9]The model is based on connectivity data gathered from the Mirage testbed.

8. REFERENCES

[1] https://mirage.berkeley.intel-research.net/.
[2] http://www.cs.berkeley.edu/∼ameli/routing.pdf.
[3] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *ICDCS-22, 2002.*, 2002.
[4] D. Braginsky and D. Estrin. Rumor routing algorthim for sensor networks. In *1st ACM international workshop on Wireless sensor networks and applications*. ACM Press, 2002.
[5] M. Charikar, S. Khuller, and B. Raghavachari. Algorithms for capacitated vehicle routing. In *30th annual ACM symposium on Theory of computing*. ACM Press, 1998.
[6] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Technical report, Carnegie Mellon University, 1976.
[7] R. Cristescu, B. Beferull-Lozano, and M. Vetterli. On network correlated data gathering, 2004.
[8] R. Cristescu, B. Beferull-Lozano, M. Vetterli, D. Ganesan, and J. Acimovic. On the interaction of data representation and routing in sensor networks. In *ICASSP*, 2005.
[9] D. De Couto, D. Aguayo, B. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. In *HotNets-I*. ACM SIGCOMM, October 2002.
[10] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.
[11] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. Multi-resolution storage and search in sensor networks. *ACM Transactions on Storage (to appear)*, Aug. 2005.
[12] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
[13] M. Haimovich and A. H. G. R. Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, November 1985.
[14] S. T. Hedetniemi, S. M. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 1998.
[15] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks, 1999.
[16] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *6th annual international conference on Mobile computing and networking*, 2000.
[17] J. Kulik, W. R. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 2002.
[18] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. In *Proc. ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2003.
[19] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *1st international conference on Embedded networked sensor systems*. ACM Press, 2003.
[20] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *ACM SIGMOD*, 2003.
[21] S. Madden and J. Gehrke. Query processing in sensor networks. *Pervasive Computing*, 3(1), January-March 2004.
[22] T. Ralphs, L. Kopman, W. Pulleyblank, and L. Trotter. The capacitated vehicle routing problem.
[23] 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 *1st ACM international workshop on Wireless sensor networks and applications*. ACM Press, 2002.
[24] A. Scaglione and S. D. Servetto. On the interdependence of routing and data compression in multi-hop sensor networks. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 140–147, 2002.
[25] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03*, pages 14–27, 2003.
[26] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *INFOCOM*, 2004.