Chapter 1

Introduction

Recent advances in wireless and embedded technologies usher in a new era for our lives. It is expected that an increasing number of small and inexpensive wireless devices (referred to as sensor nodes [12]) are deployed for monitoring various measurements. Many applications of wireless sensor networks have been proposed such as field data collection, remote monitoring and control, smart home, factory automation and security [10][2][5][3]. In wireless sensor networks, a large number of static sensor nodes are deployed over a monitored region. Sensor nodes are capable of collecting, processing and storing environmental information, and communicating with neighboring nodes. There are access points (or called *sinks*) serving as an interface for issuing queries and collecting sensing readings from sensor nodes (e.g., temperature, pressure and humidity). Sensing readings are reported to the sink via multi-hop communications according to a required reporting frequency. Due to the nature of sensor nodes, sensor nodes use their batteries as power sources. Since sensor nodes are deployed in a hard-reach areas, battery replacements for sensor nodes are sometimes impossible. As a result, power saving is a vital design issue in wireless sensor networks [14][13][1]. Compared to the energy consumption of computing and sensing of sensor nodes, communication is a dominant factor in energy consumption. Thus, reducing as many message transmissions as possible is able to significantly prolong the life time of sensors.

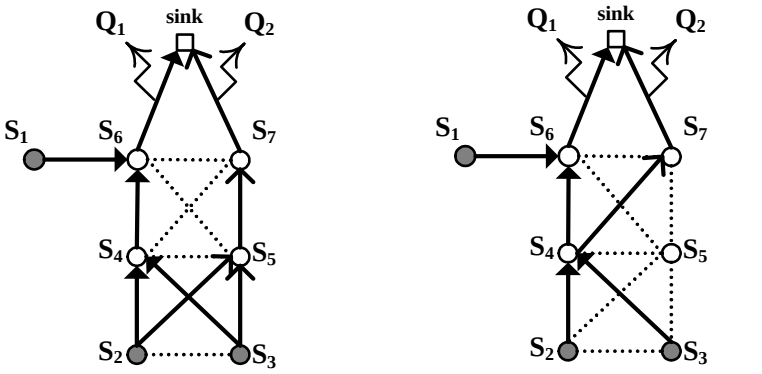

Figure 1.1: The benefit of sharing query results

In general, users submit declarative queries that specify data processing needed[4]. For example, queries specify the type of readings (e.g. temperature and light) as well as the set of sensor nodes interested by users. Through existing routing protocols in sensor networks, queries are propagated into sensor networks and query results are collected via a routing tree with the sink as the root node [6][8][16]. As described before, reducing number of messages is able to save power consumption. Prior works [6][8][16] explore the feature of in-network aggregation in which sensor nodes in routing tree are able to perform aggregated operators. By exploiting in-network aggregation, sensor nodes are able to reduce the energy consumption in that instead of sending all sensing readings to the sink, sensor nodes performing aggregate operators are able to further reduce the amount of data transmission. As such, sensor nodes could save a considerable amount of energy [6]. Same as in [6][16][9], a query is viewed as a *query tree*, where the nodes are those sensor nodes participated in the query processing and the edges among sensor nodes represent the routing paths determined by existing routing protocols for sensor networks. Moreover, in a query tree, leaf nodes are the data sources and the intermediate nodes represents the relay nodes.

In monitoring applications of wireless sensor networks, queries are typically long-running and executed over a specified period[7]. Since each query is independently performed, wire-

less sensor networks consume a considerable amount of energy when the number of queries increases. Queries issued could be categorized according to their aggregate operator and monitored time period. Queries in the same group have the same aggregate operator and monitored time period. As such, queries in the same group are able to share their query results to reduce the number of messages. Figure 1.1 illustrates the sharing of two queries. The query trees of these two queries are shown in Figure 1.1(a), where the data sources of Q_1 (respectively, Q_2) are S_1, S_2 and S_3 (respectively, S_2, S_3), and sensors nodes S_4, S_5, S_6 and S_7 are reply nodes for these two queries. Assume that we exploit in-network processing in our illustrative examples and thus, reply nodes will perform the aggregate operators. Clearly, the query results (referred to as partial results) in S_4 and S_7 are the same since data sources of S_4 and S_7 are the same, and both S_4 and S_7 perform the same query operator. Thus, Q_1 could share the partial result of S_4 with Q_2. Figure 1.1(b) shows the sharing between Q_1 and Q_2. The query tree of Q_1 is referred to as a backbone which shares query results to other queries. Note that the query tree of Q_2 needs to be adjusted to get the partial result of S_4. Those queries needed to be adjusted to get some query results from a backbone is referred to as non-backbone. Intuitively, the query result of Q_2 in Figure 1.1(b) is the same to that in Figure 1.1(a). In Figure 1.1(a), the number of messages involved is 9. On the other hand, the number of messages in 1.1(b) is 7, showing that sharing query results is able to further reduce the number of messages.

Given a set of queries with the same aggregate operator and time duration, we aim at reducing the total number of messages involved in query processing. Since these queries have the same aggregate operator, sharing query results is able to reduce message transmissions without influencing the final query result of each query. As described in our illustrative example, queries are divided into two sets: backbone and non-backbone sets. Query trees in the backbone set are issued as usual and should share their partial query results with those queries tree in the non-backbone set. The problem addressed in this paper is that given a set

	Backbone set	Non-backbine set	Number of messages
Case 1	Q_1, Q_2, Q_3, Q_4	ϕ	21
Case 2	Q_1	Q_2, Q_3, Q_4	18
Case 3	Q_4	Q_1, Q_2, Q_3	19
Case 4	Q_1, Q_4	Q_2, Q_3	18
Case 5	Q_3, Q_4	Q_1, Q_2	17

Table 1.1: Five cases of backbone sets and the corresponging numbers of messages incurred.

of query trees, we should determine which query trees should be put in the backbone set and non-backbone set so as to minimize the total number of messages involved in multiple queries. The problem we study can be best understood by an illustrative example in Figure 1.2, where four query trees are presented. Table 1 shows the five cases. In case 1, four query trees are injured to wireless sensor networks, which incurs the maximal number of messages since no query result is shared among query trees. Consider the case 2, where Q_1 is selected in the backbone set. Through the query result sharing of Q_1, the total number of messages involved in four queries is reduced to 18. If one selects Q_4 as a tree in the backbone set (i.e., Case 3), the total number of messages is 19. Clearly, selecting which query trees into the backbone set has a great impact on the number of messages reduced. In case 4, the backbone set contains Q_1 and Q_2 and the number of message in this case is 18, which is equal to that of case 2. Thus, increasing number of trees in the backbone set is not always beneficial in minimizing the number of messages. In case 5, the number of messages is reduced to 17 when Q_3 and Q_4 are in the backbone set, which incurs the minimal number of messages among these five cases. From the above example, given a set of query trees, one should judiciously decide which query trees should be included in the backbone set and how many trees should be put in the backbone set with the purpose of minimizing the total number of messages. This is the very problem we shall address in this paper.

In order to determine a backbone set that minimizes the total transmission, we first formulate the problem of selecting backbones and transform this problem into Max-Cut problem.

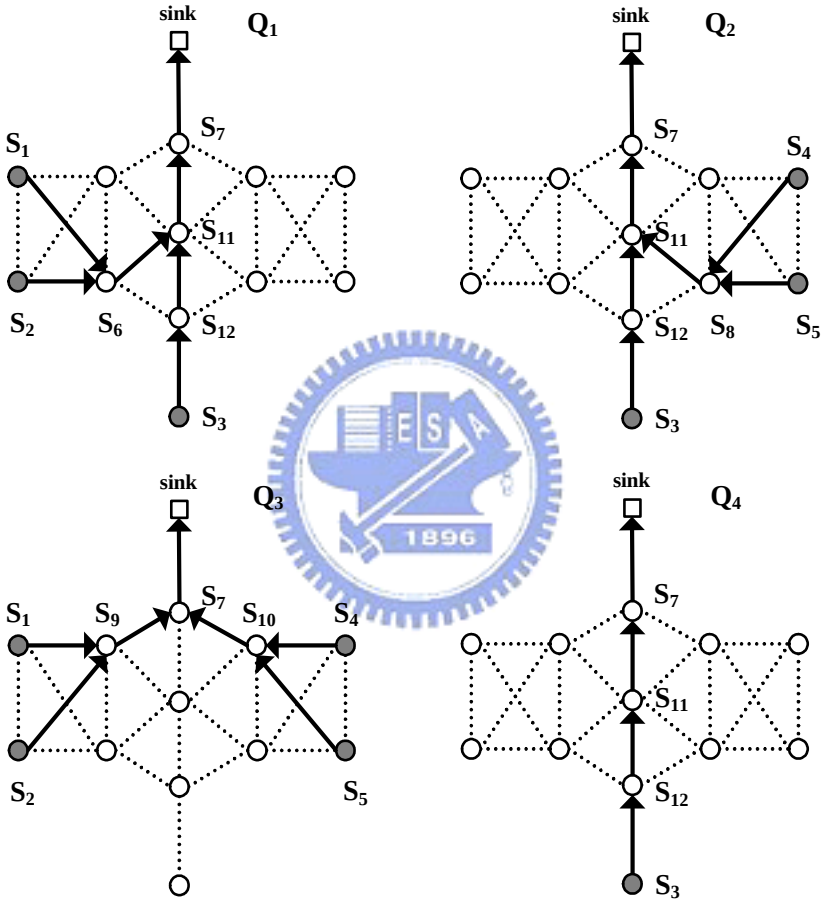

Figure 1.2: The query trees of four example queries

Specifically, given a set of queries, we derive a graph, where each vertex represents one query and the corresponding weight edge denotes the number of messages reduced by sharing the partial results. According to the graph derived, we develop a heuristic algorithm SB (standing for Selecting Backbones) to derive a cut in which both backbones and non-backbones are determined. In order to evaluate the solution quality obtained by algorithm SB and compare its resulting backbone set with the optimal one, we devise an algorithm OOB (standing for Obtaining Optimal Backbones) to obtain the optimal solution. Algorithm OOB is mainly a guided search and is similar to the well-known A* search by using a cost function to guide the search and to ensure the optimality of the goal node reached [11]. With the proper design of the guide function, algorithm OOB can obtain the optimal solution very efficiently. Performance of these algorithms is comparatively analyzed and sensitivity analysis on several parameters, including the number of queries and the distribution of data sources for queries, is conducted. It is shown by our simulation results that by sharing the partial results, algorithm SB is able to significantly reduce the total number of messages. Moreover, the backbones determined by algorithm SB is in fact very close to the optimal backbones resulted by algorithm OOB. With its polynomial time complexity, algorithm SB incurs a much shorter execution time than algorithm OOB. Since power saving is very important issue, especially for long term monitoring query processing, through sharing the partial results, algorithm SB is able to select those backbones so as to drastically reduce the number of messages, thereby saving a considerable amount of energy.

A significant amount of research efforts have been elaborated upon issues of in-network query processing for power saving in wireless sensor networks [16][9]. Prior works [6][8][16] explore the feature of in-network aggregation in which sensor nodes in routing tree are able to perform aggregated operators. By exploiting in-network aggregation, sensor nodes are able to reduce the energy consumption in that instead of sending all sensing readings to the sink, sensor nodes performing aggregate operators are able to further reduce the amount of data

transmission. In [15], with a given routing tree merged by the ones of queries, queries are executing on the merged tree and each intermediate node keeps tracking only the necessary intermediate results for all queries. To the best of our knowledge, no prior works exploit the query optimization for in-network queries, let alone devising algorithms to determine backbones for sharing partial results. Our contributions are summarized as follows.

- *Sharing partial results:* We exploit the sharing of partial results among queries so as to reduce the total number of messages incurred.

- *Formulating the problem of sharing partial results*: We formulate the problem of sharing partial results and transform this problem into a Max-Cut like problem. Based on the transformation, we devise a graph in which all possible partial result sharings are presented.

- *Developing two algorithms to determine the backbones:* In light of the graph devised, we propose a heuristic algorithm SB to obtain the backbones with the purpose of maximizing the number of messages reduced through partial result shares. Algorithm OOB is developed to evaluate the solution quality of algorithm SB.

The rest of this paper is organized as follows. Preliminaries are presented in Section 2. In Section 3, we develop algorithm SB for backbone selection. We devise algorithm OOB in Section 4 to obtain the optimal solution. Performance studies are conducted in Section 5. This paper concludes with Section 6.