In-Network Aggregation in Wireless Sensor Networks

Rocky Dunlap
rocky@cc.gatech.edu
College of Computing
Georgia Institute of Technology

1. Introduction

1.1. Wireless Sensor Networks

Sensor networks consist of small computing devices capable of producing digital representations of real-world phenomena. Due to size and battery power limitations, these devices typically have limited storage capacity, limited energy resources, and limited network bandwidth. Data produced by nodes in the network propagates through the network via wireless links. When compared to local processing of data, wireless transmission is extremely expensive. Researchers at the University of California, Santa Barbara estimate that sending a single bit over radio is at least three orders of magnitude more expensive than executing a single instruction [10]. The limited amount of energy, bandwidth, and storage capacity available to sensor nodes calls for specialized optimizations of queries injected into the network.

1.2. In-Network Aggregation

A query requesting aggregate data is injected into the sensor network at a host node, also known as a *sink*. The query is forwarded by the host to the other nodes in the network. The simplest and least optimal query plan would require each node to report its own readings back to the host node for processing. After receiving all data packets from the *source* nodes, the host node would aggregate all of the data into a final value and report the value back to the user. This approach, known as *direct delivery* [6], has a number of disadvantages. One problem is that a large number of packets must be sent to the host node. Since each node sends its own data to the host, there must be at least one packet of data sent per node. Additionally, since some nodes may not be able to communicate directly with the host, their data packets must be forwarded by other nodes until they reach the host. A related problem is that the size of each packet is relatively small since it only contains readings from one sensor. The increased number of small packet transmissions necessary to propagate needed data to the host severely limits the life of the network.

In order to conserve both energy and bandwidth, it is useful to move the integration and filtering of sensor data into the network itself. In-network aggregation is a mechanism for reducing the overall amount of power and bandwidth required to process the user's query by allowing sensor readings to be aggregated by intermediate nodes. This technique offers a number of advantages over the un-optimized approach described above.

1. **A reduction in the number of packets that must be sent through the network.** As packets propagate from source nodes they may be combined together into fewer packets containing aggregate values. Because the user is not interested in individual values there is no loss in the quality of the result returned.

2. **A reduction in the likelihood of packet collisions.** Since fewer packets are sent through the network, it is less likely that a sensor node will have to resend a message lost in a network collision [12].

3. **A reduction in the amount of redundancy received at the host node.** In the case where individual readings are reported to the host node, it is likely that multiple intermediate nodes will hear a message from a source node and each forward the message toward the host. In this case the host would receive multiple copies of the same message and would need to filter out the redundant data [13].

4. **An increase in accuracy of results.** Sensor nodes that become temporarily disconnected from the network are not able to answer queries. In this case, the parent node will notice that the child node did not report a value and may be able to estimate the value based on previous readings. This is especially helpful in non-volatile environments where readings are unlikely to change significantly over short periods of time (e.g. the temperature in one room of an office building).

1.3. Properties of Aggregates

In order to speak of aggregate functions in general terms, it is helpful to identify properties common among groups of aggregates. Researchers at UC Berkeley point out several dimensions used to classify aggregates [3].

1. **Duplicate sensitivity.** This property specifies whether an aggregate function will return the same result when the dataset contains duplicate values. Examples of duplicate sensitive aggregates are MEDIAN, AVERAGE, and COUNT. Examples of duplicate insensitive aggregates include MIN, MAX, and COUNT DISTINCT.

2. **Exemplary/Summary.** Exemplary aggregates always return a representative value present in the dataset while summary aggregates perform some calculation over the entire dataset and return the calculated value. Summary values (such as AVERAGE and COUNT) are more easily estimated even in a lossy network where all data packets are not received. Exemplary aggregates, on the other hand, may be highly inaccurate if even a few messages are lost. Such aggregates include MIN, MAX, and MEDIAN.

3. **Monotonic aggregates.** Aggregates that allow early testing of predicates in the network are monotonic. For example, assume the user requests the MAX temperature reading in the network. As source nodes report their values toward the host node, other nodes may listen and only report their own values if they are greater than the current MAX. This provides savings in the overall number of messages sent through the network without affecting the result.

4. **Partial state requirements.** The amount of partial state information required differs among aggregate functions. Aggregates such as SUM and COUNT require partial state records that are the same size as the final aggregate. The AVERAGE function requires a partial state record containing two values (both the SUM and COUNT). Other aggregates such as MEDIAN and HISTOGRAM require that the entire dataset be returned to the host node unless some type of compression or estimation is used (see [10]).

2. Approaches for In-Network Aggregation

2.1. Routing Trees

Almost all techniques for in-network aggregation require the construction of a routing tree for propagating data from source nodes to the sink (see [1, 2, 3, 4, 5, 6, 10]). Once established, each node utilizes the routing tree to find a path to the host node. A simple method for constructing the routing tree is as follows. The host node broadcasts an initialization message into the network. The message contains a hop count parameter, which specifies the distance from the host node. All nodes that hear the initialization message will select the host as their parent, increment the hop count by one, and then rebroadcast the message. The message will propagate down the network until every node has established a parent. This method of constructing the routing tree has been called *First-Heard-From* [5] or *naive* [9] tree building.

2.1.1. Grouping

Some aggregate functions require sensors to be partitioned into distinct groups. In this case it would be helpful for nodes in the same group to organize themselves together in the topology [2, 3, 5]. Researchers at the University of Pittsburgh propose a *Group-Aware Network Configuration* in which child nodes attempt to select parents in the same

group [5]. During tree initialization, parent nodes broadcast their group ID along with the tree initialization message. Child nodes listen to messages from nodes one level higher in the tree. A child node may choose to switch parents if it hears from a node one level higher that is in the same group as itself. The advantage of this approach is that nodes with children in the same group may aggregate values reported from the child nodes into one value to be passed on to the host node. If a parent node receives readings from children in different groups, it must send a larger message up the tree containing the values and the associated group. This reduces the amount of aggregation that can be performed in network.

2.1.2. Tree Optimization

Researchers at Georgia Tech suggest an optimization phase after tree initialization. During this phase the sensor nodes decide among themselves which nodes would best serve as *fusion points* (where aggregation occurs) based on the *health* of the nodes [9]. A node's health is a measure of how well the node can fulfill its assigned role based on a cost function. After forming an initial routing tree, each node reports its health to neighboring nodes. If a neighboring node determines that it could act as a better fusion point, it will inform the sending node of its intent to take over the role. As nodes transfer roles, the overall health of the network improves.

2.1.3. Establishing Data Flow Paths

Instead of aggregating data *opportunistically* (i.e. when similar data values happen to meet at a point in the network), it may be beneficial to establish paths of efficient data flow from sources to sinks [14]. Directed Diffusion is a data-centric routing protocol that attempts to find empirically good paths for data flow [7]. If a node determines that a neighbor is a particularly good source of information, it will send a *reinforcement* message to the neighbor. These reinforcement messages help establish an efficient path for data flow among the nodes. Directed Diffusion also provides a way to negatively reinforce a particular path (e.g. when a better path has been found).

Researchers at UC Berkeley have designed a routing algorithm called Data Funneling, which attempts to find a single path for data to flow from a group of source nodes back to the sink [12]. During the initialization phase, a "cost to reach the host" variable is maintained in each packet. Source nodes use the cost variable to determine the least expensive path for transporting readings back to the host. The Data Funneling approach causes groups nodes to form *clusters*, which communicate with the host node along a single path.

A similar approach called *greedy incremental tree* is discussed in [14]. First, a shortest path is established between the first source node and the sink. Other source nodes are connected to the closest point on the existing tree until all source nodes have a data flow path back to the sink. Negative reinforcement is used to prune unnecessary or inefficient paths.

2.2. Synchronization

In order to effectively aggregate data in the network, sensor nodes must set up coordinated communication patterns. For example, before forwarding data on the host node, a parent node should wait until it has received and aggregated readings from all of its child nodes. Determining how long to wait for messages is a difficult problem. First of all, nodes in the network are unreliable, so it is likely that some child nodes will not report at all. In this case the node could wait some predetermined amount of time and then proceed even if it has not heard from all child nodes. If a node waits too long, its data may not return to the host in time to be included in the final result. The timing problem presents a tradeoff between the *accuracy* and *freshness* of the result [1]. If a node returns data too quickly, the results may not be accurate because it may not include data from all child nodes. On the other hand, if the node waits too long, the data may become stale and will not be useful to the host node. The problem is further compounded because small inexpensive sensing devices typically do not provide accurate time synchronization among themselves.

2.2.1. Classification of Periodic Aggregation Protocols

Periodic synchronization algorithms for data aggregation can be classified into several groups as discussed in [1].

- **Periodic simple** aggregation means that each node waits a predetermined amount of time, aggregates all data received, and then forwards the data toward the host node. Such an algorithm is simple to implement, but does not guarantee accuracy of the data.

- **Periodic per-hop** aggregation means that each node waits until it receives data from all children, aggregates the data, and then forwards it toward the host node. This approach requires the use of a timeout in case some of the children do not respond to the query.

- **Periodic per-hop adjusted** is similar to the per-hop approach, except the timeout is based on the node's position in the routing tree. Nodes lower in the routing tree should experience a timeout before nodes closer to the host. This type of *cascading timeout* causes a "data wave" to propagate up the tree toward the host at regular intervals (sometimes called *epochs*).

2.2.2. TiNA

Researchers at the University of Pittsburgh propose an improved synchronization scheme called TiNA (Temporal Coherency-Aware in-Network Aggregation) in [4]. The goals of TiNA are to reduce the amount of information that must propagate through the network, and to increase the accuracy of reported results even with some sensors have lost communication with the network. The scheme works as follows. The user specifies a

temporal coherency tolerance (*tct*) parameter along with each query. The *tct* determines how much the sensor readings must differ from previous readings before they are reported. Leaf nodes in the network remember the last reported reading to their parent node. Then, when the time comes to report a new reading, leaf nodes only send readings if they have changed more than the given tolerance. Internal nodes also remember the last reported reading from each child node. If a child node does not report a reading, its parent simply uses the last reported value. Each internal node will then look at the new aggregate values and only report those that have changed more than the tolerance threshold. Essentially, only data values that have significantly changes are reported back to the host node, reducing the overall amount of network traffic. TiNA has the added advantage that if a node loses communication with the network, its parent can "estimate" the current value by using the last value reported.

2.3. Compression

Applying in-network compression algorithms also reduces the overall number and size of messages flowing through the network. Aggregation itself is a form of compression since many distinct data values are combined into a smaller number of representative values. However, some aggregate functions require the concatenation of all readings be returned to the host node. For example, in order to accurately determine the MEDIAN value in a network, the host node must know all values. In this case, it may still be possible to reduce the size and number of messages by applying compression techniques to the data. In [10], researchers propose a unique data structure called a Quantile Digest (or q-digest), which provides approximate results that adhere to a strict error bound. Compression is a useful energy-saving technique for sensor network applications where 100% accuracy is not required.

3. Future Work

The following sections describe areas requiring future research.

3.1. Multi-Query Optimizations

After a large-scale network of sensors has been deployed in the environment, it is likely that multiple users will inject similar aggregation queries into the network. Ideally, nodes would recognize when partial aggregates can be utilized across multiple queries. For example, assume two queries are injected into the network at approximately the same time. One query asks for the COUNT of temperature nodes currently sensing while the other query asks for the AVERAGE temperature among all of the sensors. Instead of calculating these two queries separately, nodes should simply utilize the COUNT determined by the AVERAGE query. Obviously, this is an overly simplified case, but the concept is the same for even very complex queries. A model should be developed

describing the relationship among aggregate functions so it is clear where partial aggregates can be reused for multi-query optimizations.

3.2. Metadata Management

In order to efficiently aggregate sensor data in-network, each node must maintain a certain amount of data about neighboring nodes. For example, many existing algorithms require nodes to know which other node is their "parent" in the routing tree. We have also mentioned other types of metadata that must be maintained such as the "cost" of routing data along a certain path. Almost all current schemes for in-network aggregation require that nodes maintain at least a small amount of metadata. However, because some nodes are extremely limited in storage capacity, work must be done to find the most efficient amount of metadata that should be maintained. Where should the metadata be stored? Should each node retain its own local copy, or should the data be placed at strategic locations in the network? What other kinds of metadata should be tracked in order to increase efficiency of query processing?

3.3. Query Languages

Many researchers have suggested enhancements to SQL that will better support distributed queries over sensor networks. However, a standard language has yet to emerge. Due to the extreme popularity and current DBMS support of SQL, it is likely that some kind of SQL enhancements will ultimately prevail as the standard for querying sensor data. Work needs to be done to determine the common elements needed for querying sensor data of all kinds. Current language constructs require users to explicitly define aggregate queries. However, it is likely that users would prefer to see aggregate values instead of individual sensor readings. For this reason, a sensor query languages should provide implied aggregates. For example, instead of issuing a query like "Give me the average temperature of all sensors in room X," the user should simply say, "What is the temperature in room X?" and all valid sensors will automatically compare and aggregate their readings and return the best value for the temperature in room X.

4. References

[1] I. Solis and K. Obraczka, "In-network aggregation trade-offs for data collection in wireless sensor networks," *INRG Technical Report 102*, 2003, http://inrg.cse.ucsc.edu/techreports/tr102.pdf.

[2] S. R. Madden, R. Szewczyk, M. J. Franklin and D. Culler, "Supporting aggregate queries over ad-hoc sensor networks," in *Workshop on Mobile Computing and Systems Applications (WMCSA)*, 2002.

[3] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A tiny aggregation service for ad-hoc sensor networks," in *Proceedings of the Symposium on Operating Systems Design and Implementation, OSDI*, December 2002.

[4] M. A. Sharaf, J. Beaver, A. Labrinidis and P. K. Chrysanthis, "TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation," in *Proceedings of the 3rd ACM MobiDE Workshop*, September 2003.

[5] Jonathan Beaver, Mohamed A. Sharaf, Alexandros Labrinidis, and Panos K. Chrysanthis, "Location-Aware Routing for Data Aggregation for Sensor Networks," *Post Proc. of the 1st Geo Sensor Networks Workshop*, Portland, Maine, October 2003.

[6] Yong Yao and Johannes Gehrke, "Query processing in sensor networks," in *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.

[7] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed Diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom)*. ACM, August 2000.

[8] J. Heidemann *et. al*., "Building Efficient Wireless Sensor Networks with Low-Level Naming," *18th ACM Symposium on Operating Systems Principles*, October 2001.

[9] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul and U. Ramachandran, "DFuse: A Framework for Distributed Data Fusion," *The First ACM Conference on Embedded Networked Sensor Systems (Sensys '03)*, Los Angeles, California, November 2003.

[10] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal and Subhash Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks," in *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, August 2004.

[11] A. Boulis, S. Ganeriwal and M.B. Srivastava, "Aggregation in Sensor Networks: An Energy-Accuracy Trade-off," *Sensor Network Protocols and Applications (SNPA '03)*, May 2003.

[12] Dragan Petrovic, Rahul C. Shah, Kannan Ramchandran and Jan Rabaey, "Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks," in *Proceedings of the IEEE Sensor Network Protocols and Applications (SNPA),* May 2003.

[13] B. Krishnamachari, D. Estrin, and S. Wicker, "The Impact of Data Aggregation in Wireless Sensor Networks," *International Workshop on Distributed Event-Based Systems, (DEBS '02)*, Vienna, Austria, July 2002.

[14] C. Intanagonwiwat, D. Estrin, R. Govindan and J. Heidemann, "Impact of Network Density on Data Aggregation in Wireless Sensor Networks," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02),* July 2002.

[15] B. Krishnamachari, D. Estrin and S. Wicker, "Modelling Data-Centric Routing in Wireless Sensor Networks," *USC Computer Engineering Technical Report CENG*, 2002.