# Data Dissemination over Wireless Sensor Networks

Sooyeon Kim, Sang H. Son, *Senior Member, IEEE*, John A. Stankovic, *Fellow, IEEE*,
and Yanghee Choi, *Senior Member, IEEE*

***Abstract*— In sensor networks, it is crucial to design and employ energy-efficient communication protocols, since nodes are battery-powered and thus their lifetimes are limited. We propose a data dissemination protocol for periodic data updates in wireless sensor networks, called SAFE (sinks accessing data from environments), which attempts to save energy through data delivery path sharing among multiple sinks that have common interests. Simulation results show that the proposed protocol is energy-efficient as well as scalable to a large sink population.**

## I. Environmental Model

THIS letter assumes that each sensor node has a processor, memory, and a short-range radio communication facility. Nodes are powered by batteries, and assumed to be revoked when they exhaust all the battery power. We also presume the support of underlying protocols, including topology management, localization, routing, service availability advertisements, etc.

## II. The Data Dissemination Protocol

### A. Query Transfer

The proposed protocol SAFE distributes sensor observations to sinks that explicitly present their interests by sending data requests. When a node needs data updates from a remote location, the node transfers a *query* to its neighbors via one-hop broadcast, which includes the location, the sensor data type, the desired data update rate, and the service duration. Such a node that initiates data dissemination is referred to as a *sink*.

Every node maintains a recent query table and a data management table. The query table records the most recent queries that have been received, and the data table keeps the status of sensor data being or to be distributed by the node. Receiving a query from another node, a node executes a function in Figure 1. At first, the node checks the query table if the same query has recently been dealt with. If so, the new query arrival is ignored to avoid wasting resource. Otherwise, the node saves the query into its query table, and then does appropriate things depending on its status regarding the requested data in the data table. When the node is the data source, it sends a *PathSetup* message to the inquiring node via unicast. If the node is not the source but on a dissemination path, which is called a *junction*, it sends a *JunctionInfo* message to the sink via unicast. When the node is neither the data source nor a junction, it forwards the query to the next hop, as long as it is not farther away from the queried location than the previous hop node. The previous hop node information might be injected by the routing protocol or this data dissemination protocol before forwarding a query.

```
recvQuery (q)
 1  if isRecentlyDealtWith (q)
 2     then return
 3  saveQueryAsRecentOne (q)
 4  if isSource (q)
 5     then sendPathSetup (sender(q))
 6  else if isJunction (q)
 7     then sendJunctionInfo (sender(q))
 8  else if isApproachingToSource (q)
 9     then forwardQueryToNextHop (q)
```

Fig. 1. How to deal with a query arrival: a functional description.

```
recvPathSetup (p)
 1  if destination (p) ≠ myAddr
 2     then if noEntryInDataManTable (p)
 3             then e ←createEntry (p)
 4                  waitForAckFromSink (e)
 5  else /* if the PathSetup p is destined for this node */
 6     then e ← findEntry (p)
 7          if currState (e) = QUERY_SENT
 8             then sendAck (hopSender (p))
 9                  changeMyState (e, SUBSCRIBE_SENT)
10          else if currState (e) = FEEDBACK_RCVD
11             then if bestFeedbackCost (e) > cost (p)
12                  then saveAsBestFeedback (p)
```

Fig. 2. How to deal with a PathSetup arrival: a functional description.

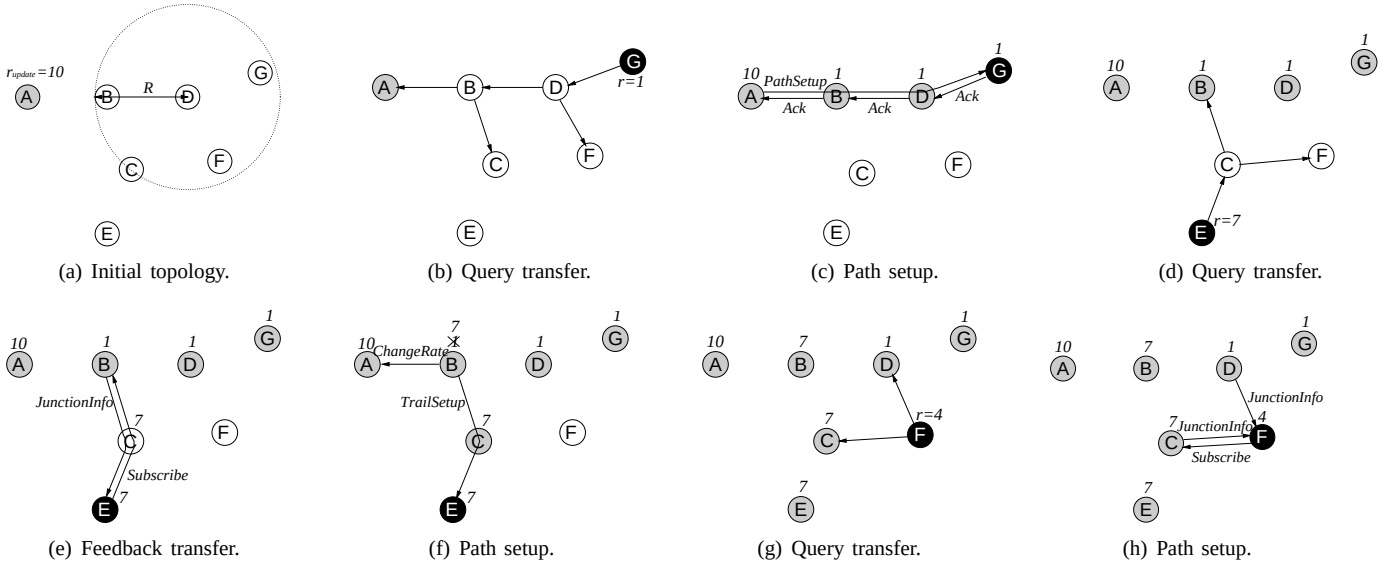
### B. Dissemination Path Setup

While a PathSetup message is delivered to the sink, all the intermediate nodes follow the steps in Figure 2. Without the corresponding entry in the data table, an intermediate node creates a new entry with the previous hop node (*progenitor* hereafter) information. Once having an entry for the PathSetup message, each intermediate node starts a timer that waits for an *Ack* message from its descendant, which confirms the path is activated. The duration of the timer is set as a pessimistic estimation of round trip time to the sink, say, several times the network diameter. This timer prevents memory waste at

Manuscript received January 30, 2004; revised March 18, 2004. This work was supported in part by the Brain Korea 21 project of Korea Ministry of Education, the National Research Laboratory project of Korea Ministry of Science and Technology, DARPA grant xxx, and NSF grant yyy.

S. Kim and Y. Choi are with the School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea. Email: {sykim, yhchoi}@mmlab.snu.ac.kr.

S. H. Son and J. A. Stankovic are with the Department of Computer Science, University of Virginia, Charlottesville, VA 22904-4740. Email: {son, stankovic}@cs.virginia.edu.




(a) Initial topology. (b) Query transfer. (c) Path setup. (d) Query transfer.

(e) Feedback transfer. (f) Path setup. (g) Query transfer. (h) Path setup.

Fig. 3. How the proposed data dissemination protocol SAFE establishes data delivery paths.

irrelevant nodes, by releasing the memory occupied by that entry when the timer expires.

JunctionInfo messages do not build paths while being transferred to sinks. This is based on a presumption that given that a junction happens to be in the vicinity of a sink, there might also exist other junctions in that area. A path from a junction to a sink is established only after the sink subscribes to that junction. A sink compares every feedback (either a JunctionInfo or a PathSetup) during a certain amount of time[1] after the first feedback received. When we attempt to minimize message exchanges over the network, the best subscription locus is one that can update the sink with the smallest number of extra messages. We define the messaging overhead as *subscription cost* $C$ of a junction $j$ when a data sink $m$ wants data updates from $s$ through $j$

$$C(m,s,j) = \begin{cases} d(s,j)\cdot(r_m - r_j) + d(j,m)\cdot r_m & \text{if } r_m > r_j \\ d(j,m)\cdot r_m & \text{otherwise} \end{cases}$$

where $d(a,b)$ quantifies the hop distance from node $a$ to $b$, and $r_a$ denotes the update rate requested by and thus available to node $a$.

When the timer has expired, a sink subscribes to the node of the best feedback until then. If the best one is a junction, the sink sends a *Subscribe* message to that junction. Otherwise, when the source is eventually the best subscription point, the sink sends an Ack to its progenitor and every progenitor acknowledges its progenitor in turn until the source gets an Ack. Receiving a Subscribe message from a sink two or more hops away, a junction sends a *TrailSetup* message to that sink and establishes a path. This path enforcement has two purposes. First, it considers the asymmetry of low-power wireless communication [1], establishing the path in the direction of actual updates instead of using the upstream path the Subscribe message has traveled along. Second, it makes the subscription status management at each node scalable, keeping a node's potential subscribers as its immediate neighbors only.

A *ChangeRate* message is transferred to the progenitor when the update rate has to be increased or decreased. As an extreme, a ChangeRate message with the new update rate of zero means unsubscribe from the update service.

Figure 3 illustrates an example, where nodes G, E, and then F request sensor data originating from node A.

## III. Simulation

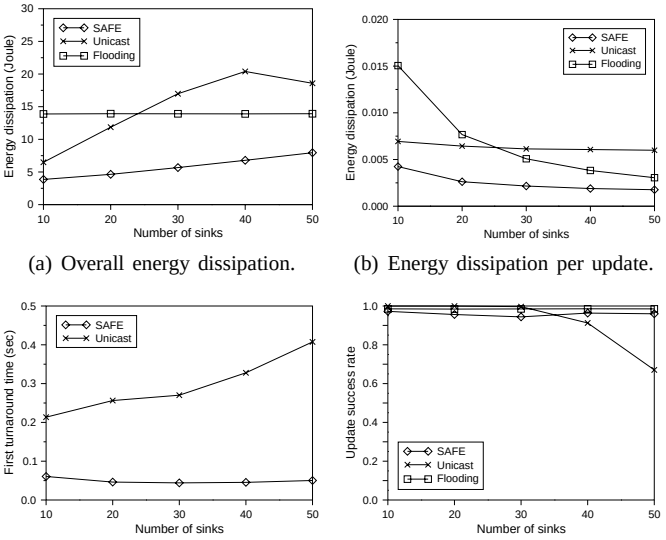
We performed simulations using GloMoSim [2], with 802.11 MAC and SPEED [3] routing. We adopt the radio energy model of an actual prototype [4], where the energy dissipation is 1$\mu$J/b (transmission) and 0.5$\mu$J/b (reception). Two simple protocols are implemented as baselines: *unicast* and *flooding*. Using unicast, every data source serves each data sink separately. With flooding, any queried data are broadcast to the entire network with the maximum update rate desired by the sinks.

We employ four metrics for the performance evaluation of SAFE. *Overall energy dissipation* is the total energy dissipation of the entire network. *Energy dissipation per effective data update* measures the ratio of overall energy dissipation to the total number of distinct data update messages received by sinks. *First turnaround time* quantifies the elapsed time between query transfer and the first feedback arrival. Finally, *data update success rate* measures the ratio of the number of effective updates received by sinks, to the total number of updates expected.

In the following simulations, 100 sensor nodes form a grid network over a 2500m$\times$2500m terrain. The query inter-arrival times follow the exponential distribution ($\mu$=1-5 sec). A message is 64 bytes long, and every plotting is the average of 20 runs with the 95% confidence intervals ranging from 0% to 13% of the mean.

Figure 4 depicts a result with a source and sink populations between 10 and 50. Desired update intervals are 3 seconds. For

[1]For example, in the simulations in Section III, this value was set as five times the network diameter.




(a) Overall energy dissipation. (b) Energy dissipation per update.

(c) First turnaround time. (d) Update success rate.

Fig. 4. Single data source, varied number of data sinks, fixed desired update intervals.

all the sink populations tested, SAFE always outperforms the two baselines in total and per-update energy dissipation at the same time, and spends less energy per update with larger sink populations due to path sharing. Figure 4(c) shows how fast a sink receives the first update. Flooding is not depicted, because with flooding all the sinks except the first one experience zero response time and the first turnaround time on average is not a meaningful factor. SAFE always exhibits fast response time regardless of background traffic volume due to distributed query processing at junctions. Also SAFE retains a reasonable level of update success rate, while unicast fails to maintain the success rate with 50 sinks.

## IV. Conclusions

This letter introduces a data dissemination protocol that attempts data delivery path sharing between multiple sinks. Results show that the proposed protocol achieves energy efficiency as well as scalability, both of which are crucial for large-scale battery-powered sensor networks.

## References

[1] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "An empirical study of epidemic algorithms in large scale multihop wireless networks," Technical Report IRB-TR-02-003, Intel Research, Mar. 2002.

[2] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla, "GloMoSim: A scalable network simulation environment," UCLA Computer Science Department Technical Report 990027, May 1999.

[3] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A stateless protocol for real-time communication in sensor networks," In *Proc. of the 23rd International Confernece on Distributed Computing Systems (ICDCS-23)*, Providence, RI, USA, May 2003.

[4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," In *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, Cambridge, MA, USA, Nov. 2000.