The Impact of Timing in Data Aggregation for Sensor Networks

Ignacio Solis and Katia Obraczka
{isolis,katia}@cse.ucsc.edu
Computer Engineering Department
University of California, Santa Cruz

***Abstract*— This paper evaluates the effect of timing in data aggregation algorithms. In-network aggregation achieves energy-efficient data propagation by processing data as it flows from information sources to sinks. Our goal is to show that the decision of when to "clock out" data as it is processed by nodes have significant performance impact in terms of data accuracy and freshness. Using the sensor network paradigm where all nodes produce information periodically, we compare three aggregation timing policies. Through extensive simulations we show that setting up the clock out timer based on a node's position in the aggregation tree results in a beneficial "cascading effect", yielding considerable energy efficiency, yet maintaining data accuracy and freshness.**

I. Introduction

One of the challenges posed by sensor networks is the fact that they are energy constrained. Because of power and transmission range limitations, data dissemination in sensor networks is typically carried out as a collective operation, in which sensors collaborate to get data from different parts of the sensor network to the information sinks.

One way of performing power-efficient data collection in sensor networks is to process the data as it flows from information sources to sinks. This technique is commonly referred to as (in-network) data aggregation and can be quite effective at conserving power. Indeed, a number of research efforts targeting sensor networks have proposed different forms of aggregation techniques to achieve power efficiency [1][2][3][4].

We focus on data aggregation techniques that target an important class of sensor network applications, namely monitoring. In particular, we consider monitoring applications in which **all** nodes produce relevant information **periodically**. Example scenarios that fall in this category include monitoring of continuous environmental conditions like temperature, humidity, seismic activity, etc.

By favoring energy efficiency, in-network aggregation may affect the quality (e.g., accuracy and freshness) of the data that ultimately reaches information sinks. The goal of this paper is to investigate the trade-off between energy efficiency and data accuracy and freshness posed by in-network aggregation. More specifically, we investigate the effect of timing in periodic aggregation: our hypothesis is that timing models have significant impact on the freshness and accuracy of data delivered by aggregation algorithms. The timing model defines when to "clock out" data as it is aggregated by nodes on its way to the information sink. The question is how long should a node wait to receive data from its children. If nodes wait too long, data produced in the next period will interfere with data from previous periods.

We compare three different timing models. *Periodic simple* aggregation works by having each node wait a pre-defined period of time, aggregate all data items received, and send out a single packet containing the result. *Periodic per-hop* aggregation works similarly to *periodic simple*, but transmits the aggregated data as soon as it hears from all its children. Finally, in *periodic per-hop adjusted*, nodes adjust their timeout based on their position in the data collection tree.

The contributions of this paper include: (1) the development of a timing model for periodic aggregation we call *cascading timeouts*, that achieves considerable energy savings while maintaining data accuracy and freshness, and (2) a comparative performance study of different aggregation algorithms using extensive simulations.

The remainder of the paper is organized as follows. Section II presents related work and Section III describes our *cascading timeouts* aggregation mechanism. Simulation results comparing *cascading timeouts* with other aggregation algorithms are presented in Section IV. Section V outlines our concluding remarks and future work directions.

II. Related Work

Protocols for sensor networks have sparked considerable interest in the network research community. In

this context, data aggregation rose as a technique for improving sensor network protocols' energy efficiency. We briefly describe some previous and on-going research efforts in order to put our work in perspective.

Directed diffusion [1] has been proposed as a data gathering protocol for sensor networks. It targets the monitoring of events which are typically sensed only by a few nodes. Diffusion's communication paradigm is based on information sinks broadcasting requests, or *interests*, for relevant data. Nodes producing relevant information respond and *data paths* are formed. Data is aggregated when a node is part of various data paths. Diffusion falls in the *periodic simple* category.

eScan [3] is an energy monitoring scheme that collects energy readings from every participating node. Their scenario is somewhat similar to the ones we target, i.e., every node maintains an energy value that is reported to a collection sink. Aggregation is performed as data flows to the sink by merging reports of similar energy values into *energy range polygons*. Rather than being an alternative to eScan, our aggregation techniques can complement it, for example, to improve data freshness, since, in its original design, eScan does not try to optimize latency in delivering data to the sink.

SPIN [4], Sensor Protocols for Information via Negotiation, is a protocol for data collection and dissemination. In SPIN, all nodes have pieces of named information that they want to send to the rest of the nodes. Data transfers are first negotiated based on the names of items. Only requested items are exchanged. SPIN does not really use an explicit aggregation mechanism; aggregation is performed implicitly during initial negotiation between nodes.

TAG [2], or Tiny AGreggation, is a sensor network querying system. It employs a SQL-like syntax and uses aggregation as the query is processed within the network. When a query involves an *epoch*, requiring readings to be collected periodically, TAG uses the *periodic per-hop adjusted* aggregation approach. It subdivides the epoch into slots. The length of a slot is given by the epoch length divided by n, the radius of the network. Slots are assigned to nodes in decreasing order, n, $n-1$, $n-2$, ... , as the query propagates through the network. Nodes transmit in their slot, hence, the out-most nodes will transmit first and nodes closest to the sink, last. As in any time-slotted mechanism, clock synchronization among nodes is required so that nodes transmit in their designated slots.

Convergecasting [5] also performs aggregation as it collects data periodically from all nodes to a single sink. Like TAG, its data aggregation mechanism also falls in the *periodic per-hop adjusted* category. It assigns aggregation slots as the query percolates the sensor network, trying to assign nodes to different (increasing) slots in order to avoid collisions. Once the algorithm finishes assigning slots, the order of the slots is inverted according to the data collection tree. A similar concept is used by Florence et al. [6].

More recently, the energy-accuracy tradeoff study by Boulis et al. [7] recognizes the importance of timing models for efficient data aggregation. It proposes a data collection mechanism where nodes decide whether to share their own readings based on estimates they get from other nodes. While this works well for operations like reporting the maximum or minimum value, it does not apply to more general sensor network monitoring applications. The work does not try to optimize data freshness.

III. Cascading Timeouts

Cascading timeouts aggregation targets periodic data generation applications in which nodes produce data at regular periods. A given node aggregates data received from its children into a single data item, which is then forwarded upstream towards the information sink. Application scenarios that fit well within this communication model include monitoring of continuous environmental conditions like temperature, humidity, seismic activity, etc. While we focus on the single information sink scenario, the proposed technique applies to multi-sink scenarios.

Some of *cascading timeouts*' design goals include:

- Simplicity: given that sensor network nodes are typically anemic devices in terms of energy, processing, storage, and communication capabilities, designing simple aggregation algorithms is key.
- Efficiency: generating close to minimal control overhead is another critical requirement for the resource-constrained environments our algorithms target.
- No clock synchronization: not relying on external clock synchronization mechanisms is important. No matter how efficient clock synchronization mechanisms become (an example of an efficient clock synchronization mechanism is reported in [8]), they will require additional message exchange among nodes and thus incur additional energy consumption.
- Routing protocol independence: not assuming a specific underlying routing protocol makes *cascading timeouts* quite general.

Similar to most periodic aggregation mechanisms, *cascading timeouts* starts by having the sink broadcast the initial request to all nodes. This initial request triggers a simple tree establishment protocol which sets up reverse

paths from all nodes back to the sink (root of the tree). Upon receiving the request message, nodes send a reply back to their parent. Each node can then deduce how many children they have. Nodes assume a broadcast medium and forward data using one-hop broadcasts. In order to avoid collisions, transmissions are scheduled using a small staggering delay.

Note that tree establishment overhead is incurred by *cascading timeouts* and most other in-network aggregation mechanisms. Even if no aggregation is employed, a distribution tree is typically used to propagate data from information sources to sinks.

In *cascading timeouts*, instead of having nodes randomly schedule their timeout, i.e., the time interval they wait to receive data from their children before forwarding the next data aggregate, a node's timeout is set based on the node's position in the data distribution tree. Thus, a node's timeout will happen right before its parent's. This causes the so-called "cascading" effect: data originating at the leaves is clocked out first, reaching nodes in the next tree level in time to be aggregated with data from other leaf nodes and locally generated data, and so on. The net effect is that a "data wave" reaches the sink in one period. This is the main reason why *cascading timeouts* is able to achieve power efficiency without sacrificing data freshness.

Timeout scheduling is part of the distribution tree setup protocol and is triggered by the initial request from the sink. The sink's request contains a "hop count" field which gets incremented as the request travels toward the leaf nodes. Using this hop count information, nodes can estimate their distance, in time, to the sink and schedule their timeout to produce the cascading effect.

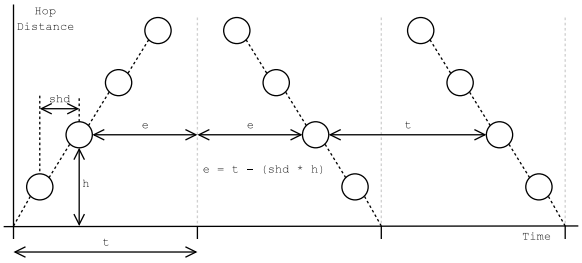

Fig. 1. *Cascading timeouts* timeout calculation

Figure 1 shows graphically timeout calculation in *cascading timeouts*, where t is the data generation period, h is a node's distance to the sink in number of hops, and shd, the *single hop distance*, is the delay to traverse one hop. Once the request packet is received, a node schedules its timeout to happen after $2e$. Subsequent timeouts will continue to be scheduled every t interval.

Note that a node's timeout depends on the *single hop distance*, shd. A detailed analysis of how shd impacts *cascading timeouts*'s performance is presented in [9]. Essentially, we show that a good estimate value of the shd can be determined and that it will undergo very small variations since traffic flows over the same data collection tree and the offered load is basically constant.

As noted above, *cascading timeouts*' timing scheme is parallel to the ones employed by both TAG and Covergecasting. According to our taxonomy, all three mechanisms are classified in the *periodic per-hop adjusted* category. In our simulations, we use *cascading timeouts* to represent *periodic per-hop adjusted* algorithms.

Other Periodic Aggregation Mechanisms

Below we describe the other classes of aggregation algorithms we use in our comparative study. As baseline, we employ no in-network aggregation when sending data from information sources to the sink. As previously pointed out, even in the no-aggregation case, we employ a distribution tree rooted at the information sink and spanning all (relevant) data sources. As packets flow from the leaves to the root, nodes simply forward them along the tree.

Periodic Simple: In *periodic simple* aggregation protocols, all nodes wait a pre-defined amount of time, aggregate all the data received in that period, and send out a single packet. The aggregation period is equal to the data generation period.

This class of aggregation protocols represents the basic mechanism used by Directed Diffusion [1] considering that all nodes have relevant data to send. Based on feedback (or reinforcements) from the sink, every node uses a specific gradient which determines the rate at which data is sent to the sink. Note that nodes are not necessarily synchronized when "clocking out" data.

Periodic Per-Hop: According to *per-hop simple* aggregation, once all data items are received from a node's children in the distribution tree, an aggregated packet is produced and sent onto the next hop. Each node uses a timeout for sending out packets in case their children's response is lost. The timeout is equal to the data generation period since once that time is up, we will be expecting and producing new readings.

IV. Simulations

A. Experimental Setup

For our comparative study of the different in-network aggregation algorithms, we ran extensive simulations using the `ns-2` network simulator [10]. In the experiments we conducted, 100 nodes were randomly placed in a $500 * 500\ m^2$ area. Nodes' transmission range and data

rate are set to 100 meters and 115 Kbps, respectively. A CSMA-like broadcast radio and FLIP [11] are used as the MAC and network protocols, respectively. Based on values used by commercially available radios, we set transmission and reception power levels to 24.75 and 13.5 milliwatts, respectively. Idle power consumption was set to 0.675 milliwatts to reflect MAC protocols that switch to low-power radio mode whenever possible.

In order to avoid collisions, nodes stagger their transmissions using a small random interval. This is important when performing data collection over a tree, especially when nodes try to send at scheduled intervals based on their depth in the tree. The maximum staggering value used was 0.03 seconds, i.e., nodes pick a uniformly distributed random timer between 0 and 0.03 before transmitting a packet.

Nodes are stationary and no transmission errors were simulated [1]; however, packets can still be lost due to collisions. Simulations were run for 20 seconds with data being generated every second (round). Although establishing the distribution tree can be initiated by the data request from the sink, in our simulations the tree was formed at time 1 second and data collection was triggered by the sink at time 3 seconds. We present steady state results, that is, measurements taken during the second half of the simulation (during the last 10 seconds).

Data points were obtained by averaging over twenty different runs using different seeds to perform random node placement. Information sink placement can greatly affect the performance of tree-based aggregation algorithms. For this reason, we ran experiments using three different sink placement strategies: corner, center, and random placement. Placing the sink in corners means that the resulting collection trees will be deeper. Center placement minimizes tree height.

Performance metrics we use include **energy consumed**, **data accuracy**, **data freshness**, and **overhead**. While energy consumed measures the algorithm's energy efficiency, data accuracy and freshness account for its effectiveness in terms of conveying as much information as possible to the sink in a timely manner.

In these experiments, we do not model the actual values being sensed by the nodes, how fast they are changing or in what manner. Therefore, accuracy is measured as the ratio of total number of readings received at the sink to the total number of readings generated. We assume lossless aggregation, which means no data is discarded. Examples include computing the minimum, maximum, as well as counting occurrences. In these scenarios, total accuracy is achieved when the sink can "calculate" an answer that involves one reading from every node per round.

Freshness is computed as the difference between the time a data item is generated and the time it is received at the sink. Overhead measures the communication complexity of the in-network aggregation algorithms.

[1]We have proposed different mechanisms to handle packet drops. They can be found in our tech report [9].

B. Results

For each class of in-network aggregation algorithms (i.e., *periodic simple* labeled as "Simple", *periodic per-hop* labeled as "Per-hop", and *periodic per-hop adjusted* labeled as "Per-hop adjusted") and sink placement strategy (corner, random, and center), Figure 2 shows their accuracy given by the total number of readings collected (bar height) and their freshness (bar shades). For comparison purposes, as baseline we use the no-aggregation strategy (labeled as "None"). The height of the bars indicate the average number of readings received per data generation period. The different shades represent the fraction of readings of different ages. In terms of data accuracy, we observe that there is not a big difference in performance when comparing the different aggregation mechanisms. However, no aggregation and *periodic per-hop adjusted* (represented by our *cascading timeouts algorithm* exhibit the highest percentage of fresh data. No aggregation has a small fraction of data 1-round old because some readings time out at the end of one round but arrive until the next one. *Periodic simple* exhibits the largest range of data ages; this is because nodes simply send data periodically, thus it can take up to D periods for the readings to arrive in the worst case, where D is the diameter of the network.

Even though most studies of data aggregation mechanisms often do not account for sink placement, we observe from Figure 2 that sink placement has indeed considerable impact on data freshness. Even for the no-aggregation case, where packets are forwarded immediately after they are received, placing the sink in the center yields fresher data.

From Table I, which shows the energy consumed by the different algorithms, we observe that, for our experimental setup, energy consumption can be reduced to a third when data aggregation is used. Note that all aggregation schemes exhibit similar energy efficiency.

We introduce *weighed accuracy*, which accounts for a data item's age, as an another metric to compare the

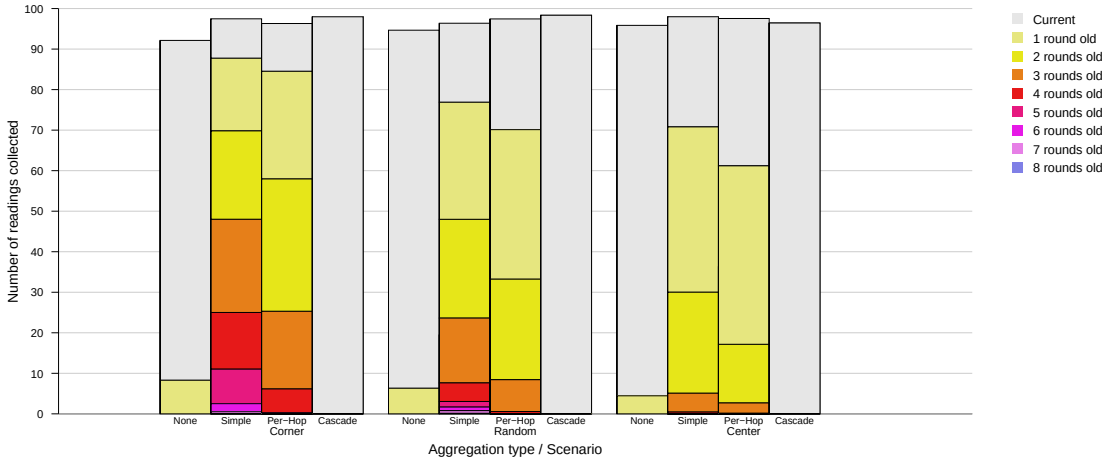

Fig. 2. Data accuracy and freshness

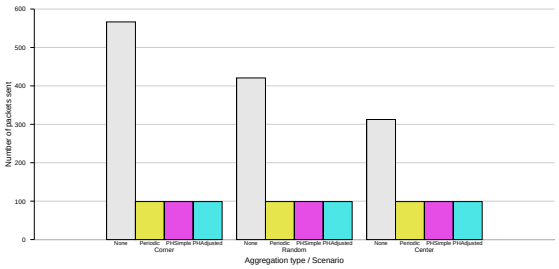

Fig. 3. Number of data packets transmitted per round

	None	Periodic	PHSimple	PHAdjusted
Corner sink	0.1425	0.0485	0.0485	0.0431
Random sink	0.1293	0.0486	0.0488	0.0425
Center sink	0.1122	0.0488	0.0489	0.0412

TABLE I

ENERGY CONSUMED BY THE DIFFERENT ALGORITHMS

performance of the aggregation algorithms with respect to freshness. The motivation behind measuring weighed accuracy lies in the fact that while some applications are interested in historical data, others may only want the most up-to-date information. This is the case of real-time monitoring, where information sinks are only interested in the latest data sensed. For the latter type of applications, aggregation algorithms should not delay data delivery beyond a certain threshold.

Weighted accuracy is computed as follows. Readings received in the same period they were produced have weight 1. Older readings are assigned an exponentially decaying weight: the older the reading, the less weight we assign to it. The expression for weighted accuracy is thus given by:

$$weighted\_accuracy = \sum_{i \in I} r_i w^i$$

Where I is the set of ages of the readings, r_i is the number of readings of age i per period and w is the weight. Readings from the current period have an age of 0 and therefore a weight of 1.

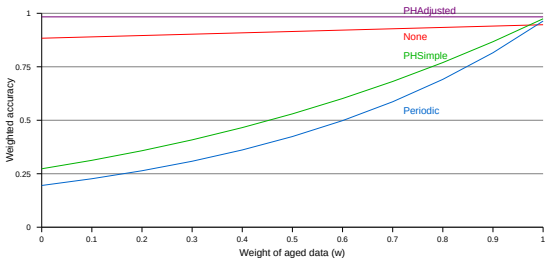

Fig. 4. Weighted accuracy - random sink placement

The graph in Figure 4 shows the performance of in-network aggregation according to weighted accuracy. We observe that no aggregation and *periodic per-hop adjusted* are the best performers. *Periodic per-hop adjusted* has a better performance because no aggregation doesn't time readings to arrive in the same period, and hence some of them arrive in the following period.

Corner and center sink placements exhibit similar performance considering that corner placement starts at

lower values and center placement starts a little higher for low weights. For corner sinks data will have to travel more hops, for center sinks the opposite is true.

We should point out that *cascading timeouts* perform consistently well for different data collection intervals. We measure the average delay (in seconds) per reading between when the reading is originally produced by a node until the sink processes all readings generated in that period. For example, in the case of a 10-second collection period using random sink placement, *cascading timeouts* achieves average delays that are more than an order of magnitude smaller than when no aggregation is used. With this same setup we can compare *cascading timeouts* to TAG. Since TAG divides the period into as many equal segments as levels in the aggregation tree, a longer period increases it's average delay per reading.

In summary, our results show that in-network data aggregation can achieve considerable energy savings. Yet, *periodic per-hop adjusted* aggregation (specifically our *cascading timeouts* algorithm) is also able to maintain the same *freshness* and *accuracy* as compared no aggregation. This is an impressive result considering the constraints imposed by applications that generate data periodically.

V. Conclusions

This paper explored in-network aggregation as a power-efficient mechanism for propagating data in wireless sensor networks. Our focus was on applications where a large number of sensing nodes produce data periodically which is consumed by fewer sink nodes. Such communication model is typical of monitoring scenarios, one key application of sensor networks.

Through simulations, we evaluate the performance of different in-network aggregation algorithms, including our own *cascading timeouts*, and characterize the tradeoffs between energy efficiency, data accuracy and freshness. Our results show that timing, i.e., how long a node waits to receive data from its children (downstream nodes in respect to the information sink) before forwarding data onto the next hop (toward the sink) plays a crucial role in the performance of aggregation algorithms in the context of periodic data generation. By carefully selecting when to aggregate and forward data, we achieved considerable energy savings (as much as 6 times less traffic) while maintaining data freshness and accuracy.

References

[1] 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.

[2] 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.

[3] J. Zhao, R. Govindan and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *Proceedings of the IEEE Wilress Communications and Networking Conference (WCNC'02)*, March 2002, pp. 17–21.

[4] W. Heinzelman, J. Kulik and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedins of the International Conference on Mobile Computing and Networking (MobiCom)*, August 1999.

[5] L. Schwiebert V. Annamalai, S.K.S Gupta, "On tree-based convergecasting in wireless sensor networks," in *Proceedings of the IEEE Wilress Communications and Networking Conference (WCNC'03)*, March 2003.

[6] C. Florens and R. McEliece, "Packet distribution algorithms for sensor networks," in *Proceedings of IEEE INFOCOM 2003*, April 2003.

[7] A. Boulis, S. Ganeriwal, and M.B. Srivastava, "Aggregation in sensor networks: An energy-accuracy trade-off," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[8] J. Elson, L. Girod and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.

[9] 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.

[10] VINT, "The Network Simulator NS-2," http://www.isi.edu/nsnam, November 2001.

[11] I. Solis and K. Obraczka, "FLIP: A flexible interconnection protocol for heterogeneous internetworking," *To appear ACM/Kluwer Mobile Networking and Applications (MONET) Special on Integration of Heterogeneous Wireless Technologies*, 2004.