Aggregating Sensor Data from Overlapping Multi-Hop Network Neighborhoods: Push or Pull?

Kay Römer
ETH Zurich, Switzerland
roemer@inf.ethz.ch

Bernd-Christian Renner
TU Hamburg-Harburg, Germany
christian.renner@tuhh.de

***Abstract*— Network neighborhoods are a key communication abstraction in sensor networks, allowing sensor nodes to collect and aggregate sensor data from nearby other nodes. In many applications, multi-hop neighborhoods of several nodes overlap, such that nodes participate in many neighborhoods, having to contribute their data items to all containing neighborhoods. We consider two orthogonal approaches to efficiently support this data aggregation problem. A push-based approach, where each node floods its data item in a multi-hop neighborhood, and a pull-based approach, where each node collects data from nodes in a multi-hop network neighborhood using a spanning tree. Our goal is to identify situations where one approach outperforms the other. For this, we implement these protocols in TOSSIM, study overhead and yield as a function of the fraction of nodes in the network that perform data aggregation over a multi-hop neighborhood, and report our findings.**

***Keywords*— sensor networks; in-network data aggregation; multiple sinks.**

I. Introduction

Network neighborhoods are a key communication abstraction in sensor networks that allows a sensor node to communicate with other, nearby sensor nodes in a well-defined multi-hop neighborhood. Here, network neighborhoods are defined by an anchor node A, a hop-radius R, and optionally a predicate p. All sensor nodes N with a distance of at most R hops from node A for which $p(N)$ holds belong to the multi-hop neighborhood of node A. The anchor node A may then perform operations on its neighborhood, such as computing an aggregate over nodes in the neighborhood (e.g., computing an average temperature over all nodes in a neighborhood).

The importance of network neighborhoods is related to the fact that phenomena observed by sensor networks are often spatially correlated, that is, nearby sensor nodes produce correlated sensor output. This can be exploited, among others, for sensor calibration [2], outlier detection and removal [6], event correlation analysis [11], or node tasking [5]. The significance of network neighborhoods is underlined by the fact that several middleware services have been developed that support the notion of network neighborhoods, including Hood [14], Abstract Regions [13], and Logical Neighborhoods [8].

In many applications of network neighborhoods, all or many nodes in a sensor network maintain a network neighborhood with the same radius and perform the same operations on the neighborhood. In this case, network neighborhoods of different anchor nodes overlap, that is, a node participates in the neighborhoods of many anchors. When collecting and aggregating data from such overlapping network neighborhoods, a node has to contribute its data items to all neighborhoods it participates in.

This paper is concerned with protocols for data collection from such overlapping multi-hop network neighborhoods. In particular, we investigate two orthogonal approaches to this problem: *Push*, where each node floods its data item to all nodes in a network neighborhood, and *Pull*, where an anchor collects data items from nodes in a neighborhood using a spanning tree. The performance of these protocols heavily depends on the *anchor density*, i.e., the fraction of nodes in the network that need to collect data from a neighborhood. While Push can be expected to work better for higher anchor densities, Pull can be expected to work better for lower anchor densities. However, in general it is not clear which approach should be selected for a given anchor density. To investigate this issue, we implement Push and Pull protocols in TOSSIM and study yield and overhead as a function of anchor density.

We present the data collection problem in Sect. II and introduce the two data collection approaches in Sect. III. Experimental results are presented and discussed in Sect. IV.

II. Data Collection Problem

We consider the following data aggregation problem in a multi-hop sensor network:

- A fraction of all nodes act as anchors and maintain a multi-hop network neighborhood with hop-radius R. All anchors use the same radius.
- All anchor nodes simultaneously perform an identical aggregation operation over their neighborhoods. That is, the performed operation is the same on all neighborhoods (i.e., computing an average) and the operation refers to the same data item (i.e., temperature) on all nodes in all neighborhoods.

This data aggregation problem is motivated, among others, by ongoing work of the same authors [11], which is concerned with finding correlations between the output of a sensor and the output of other nearby sensors. In an equipment monitoring application (e.g., [1]), for example, one is interested in understanding if an abnormal behavior of one part of a large machine is correlated with abnormal behavior of nearby other parts of the machine. The work in [11] sets out to automatically discover frequently occurring patterns of the form "an abnormal temperature reading of machine part A was preceded by abnormal vibration signatures of nearby parts B and C in 70% of all cases."

To implement the above application, machine parts would be equipped with sensor nodes that monitor specific parameters of that part and generate events whenever an abnormal

[0]The work presented in this paper was partially supported by the Swiss National Science Foundation under grant number 5005-67322 (NCCR-MICS).

behavior is detected. Each sensor node would act as an anchor of a network neighborhood to monitor the occurrence of events in its neighborhood. In particular, each anchor node would perform an aggregation operation over its neighborhood to count the occurrences of different types of events in the neighborhood. This operation is repeated at regular intervals and the resulting information is mined for frequent patterns.

Besides this specific motivating application, many other applications follow the above pattern. In [13], a tracking application is described where sensor nodes can detect the proximity of a tracked object. To estimate the target's position, each node maintains a neighborhood and computes the average of the positions of those nodes in the neighborhood that currently detect the target.

III. Data Collection Protocols

Let us consider an instance of the data collection problem described in Sect. II where *each* node in a sensor network acts as an anchor of a network neighborhood with radius R. There are two orthogonal approaches to solve the data collection problem:

- Push: Each anchor node broadcasts its data item to all nodes in its neighborhood with radius R.
- Pull: Each anchor node collects data items from all nodes in its neighborhood with radius R.

In an ideal network, both approaches ensure that each anchor node receives the data items from all nodes in its neighborhood. This is true because of the symmetry of the setup, i.e., all nodes perform at the same time the same operation on the same data item on network neighborhoods with the same radius. Note that the Push approach also works if only a fraction of the nodes are anchors that need to collect data from its neighborhood. However, *every* node would have to push its data item to nodes within a radius R. Obviously, this results in increasing overhead as the fraction of anchor nodes decreases. The above two approaches have very different characteristics:

- Push can be implemented with *broadcast* communication and without any explicit routing structures by means of scoped flooding. Pull is typically implemented with *unicast* communication using an explicit routing structure, for example a spanning tree of the neighborhood with the anchor at the root, allowing nodes in a neighborhood to route data items to the anchor. Establishing and maintaining these routing structures generates overhead, while the actual routing of data items is efficient. In contrast, scoped flooding does not incur overhead for setting up and maintaining routing structures, but the actual flooding of data items incurs overhead as each data item may be transmitted redundantly along multiple paths.
- Push has lowest overhead if all nodes are anchor nodes, because then every pushed data item is actually needed by all nodes that receive it. Pull has highest overhead if all nodes are anchor nodes, because then each node is a member of the neighborhoods of many anchor nodes and has to route its data item to all of them.
- Pull is suitable for in-network data aggregation, where each node in the neighborhood first receives data items from its children in a spanning tree, aggregates (e.g., averages) these data items and its own data item, and forwards the result to its parent in the spanning tree. In contrast, Push is not suitable for this type of aggregation, because a pushed data item needs to be aggregated with a different set of other data items for each receiving anchor.

The above discussion suggests that Push is most suitable for high anchor densities, while Pull is most suitable for low anchor densities. However, it is less obvious which of the two strategies should be preferred if the anchor density is somewhere in between these two extremes.

To study the performance of these two approaches as a function of the anchor density, we consider specific implementations of the Push and Pull strategies. These are described in the following sections.

A. Push: Scoped Flooding with Packet Aggregation

This approach is based on scoped flooding, where every node floods its data item within a network neighborhood of radius R. To reduce communication overhead, a node tries to combine data items received from other nodes into a single message by randomly waiting before sending a message and including the data items that have been received in the meantime into the outgoing message.

In more detail, protocol messages contain a list of data items, each of which consists of the address and location of the node that generated the data item, a timestamp, a hop counter, and the actual payload data. A node first generates a data item containing its address, a hop counter with value R, and its sensor data. Then, the node waits for a random amount of time (10 ... 1200 ms in our experiments[1]). From messages arriving in the meantime, data items are extracted and filtered to remove any items that have been forwarded previously. Then, the hop counter in each item is decreased and items with a value of zero are dropped. The remaining items and the locally generated data item are then combined into a single broadcast message. A single message can aggregate up to four data items due to constrained message size.

B. Pull: Tree Routing with Data Aggregation

Every anchor first builds a spanning tree of its network neighborhood with radius R. Nodes send data items along the edges of the spanning tree(s) towards the anchor(s). We perform in-network data aggregation to aggregate sensor readings (e.g., compute an average) rather than just aggregating packets as for the Push protocol described above. We investigate two variants of this approach. In the *reactive* variant, the anchor sends a request to all nodes in the tree to collect data. In the *proactive* variant, nodes proactively send data items periodically.

In more detail, the protocol works as follows. The basic approach to build a spanning tree of a network neighborhood with radius R is similar to tree routing protocols such as MintRoute [15]. All nodes send beacon messages at regular intervals containing their address and a sequence number. Using the fraction of received beacons, each node computes a reception link quality for each network neighbor. The resulting measurements are then filtered with an exponentially weighted

[1]The protocol parameters given in this and the subsequent section have been empirically tuned based on experiments with TOSSIM to optimize protocol performance.

moving average (EWMA) filter to remove outliers (we use a filter parameter of 0.1 in our experiments). The resulting values are then exchanged among neighbors to compute a bi-directional link quality for each neighbor. Unidirectional and low quality links are ignored when building the spanning tree. To establish the spanning tree, a distance vector approach is used to compute the "shortest" path (in terms of the product of link qualities along the path) from each node to the anchor. For this, each node broadcasts a message containing its own address and its current distance to the anchor. Exchange of link qualities and tree establishment are combined into a single tree building message. Note that tree establishment executes concurrently for all anchors and each node may participate in many spanning trees. Due to constrained memory, in our implementation each node can handle up to 11 network neighbors, up to 6 tree children, and each node can participate in up to 10 spanning trees (i.e., network neighborhoods).

To reduce the probability of packet collision, all messages are randomly delayed (5 ... 600 ms in our experiments) before transmission. Nonetheless we noted during initial experiments that packet loss results in many nodes not participating in the tree. We therefore sent three copies of each tree building message in our experiments. Note that as the latter is a broadcast message, schemes based on acknowledgments and retransmissions won't work here.

Once every anchor has established a spanning tree, data collection is executed periodically. A node first generates its sensory data and then waits for messages from its tree children. Then, it aggregates the received values and its own values (e.g., computes an average). Finally, the node sends a message to its tree parent containing the aggregated value. Receipt of these messages is acknowledged and unacknowledged messages are retransmitted after a timeout (we allow up to two retransmissions in our experiments).

In the *proactive* variant, every node periodically generates and sends data items. In the *reactive* variant, anchors send a request message along the spanning tree to trigger data collection. This variant is useful if data collection is only performed irregularly. In our experiments, two copies of these broadcast messages are sent to increase the chance of nodes receiving a request.

IV. Results

The goal of our experiments is the identification of criteria for the selection of a data collection protocol (i.e., Push or Pull). The key metrics we are interested in are yield (the fraction of generated data items that are actually received by anchors)[2] and overhead (the number of messages/bytes needed to collect data). Key parameters are anchor density, the radius of network neighborhoods, and network density. We also investigated the impact of different protocol parameters (e.g., maximum number of retransmissions, backoff delays) to tune protocol performance. The actual choices used during the experiments are reported in Sect. III.

Results were obtained from TOSSIM using the empirical radio model (which has been generated from real-world link quality traces) and BMAC [9], the standard TinyOS MAC protocol. Due to constrained space, we focus on a single set of experiments for fixed network density and neighborhood radius. Other parameter choices result in a similar qualitative behavior, which is described in more detail in a separate report [10].

We consider 40 nodes that are uniformly distributed in a 30m by 30m area. Our main parameter *anchor density* is defined as the fraction of anchor nodes among all nodes. Each anchor collects data from a 4-hop neighborhood considering only nodes that are at most 15m apart. Data collection is repeated every 60 seconds. Payload data consists of counters that report the frequency of different types of events during the last 60 seconds (see Sect. II). The total size of the payload data generated by each node per round is 4 bytes.

Fig. 1 shows the results we obtained for different anchor densities. Subfigure (a) shows the yield, i.e., the fraction of data items actually received by anchors among data items anchors should ideally receive. Subfigures (b) and (c) show the number of packets and bytes sent by the protocols per received data item. The values for the tree protocols do *not* include overhead for tree construction, assuming a *best-case* network environment where tree construction is a one-time overhead and maintenance costs can be ignored.

With respect to overhead, trees perform better for lower anchor densities as expected. However, the break-even point is at a rather low anchor density < 0.2. In very dynamic networks requiring frequent updates of the spanning trees, even smaller values must be expected for the break-even point. Hence, trees should be only considered for very small anchor densities. This came out as a surprise, as we expected trees to perform much better due to the lack of redundant transmissions and due to in-network data aggregation.

With respect to yield, all protocols performed remarkably bad. Even in the best case, the tree protocols show a loss of 20%, while scoped flooding exhibits a loss of as much as 55% (note that scoped flooding is independent of the number of anchors). One could expect that increasing the number of retransmissions for the tree protocols would make things better, but in fact the opposite is true. The reason for this is that almost all packet loss is caused by MAC-layer collisions. Increasing the number of retransmissions only resulted in even more collisions and loss.

The reasons for this are two-fold. Firstly, our data collection protocols exhibit a specific traffic pattern, where longer periods of idle time are followed by traffic bursts, where all nodes transmit data almost concurrently. Still, the payload data in our experiments is small (only few bytes per node per 60 seconds), such that one could expect that the underlying MAC can schedule these transmissions without generating too much collisions. However, BMAC [9] requires the transmission of lengthly preambles before each packet, resulting in a large channel utilization even for small payloads. Further, BMAC does not deal with the hidden terminal problem, resulting in a large number of collisions even for a moderate channel utilization. While scheduled protocols such as LMAC [12] eliminate this problem, they have a high overhead in idle situations. In fact, this observation led us to design a MAC protocol that is more suitable for bursty traffic patterns by combining low overhead in idle situation with collision-free transmissions. We expect that this protocol, once available,

[2]Packet loss is a major problem in most multi-hop data collection sensor networks. Data yields of 58% for Great Duck Island, 40% for Redwood monitoring, 68% for Volcano monitoring [3] have been reported.

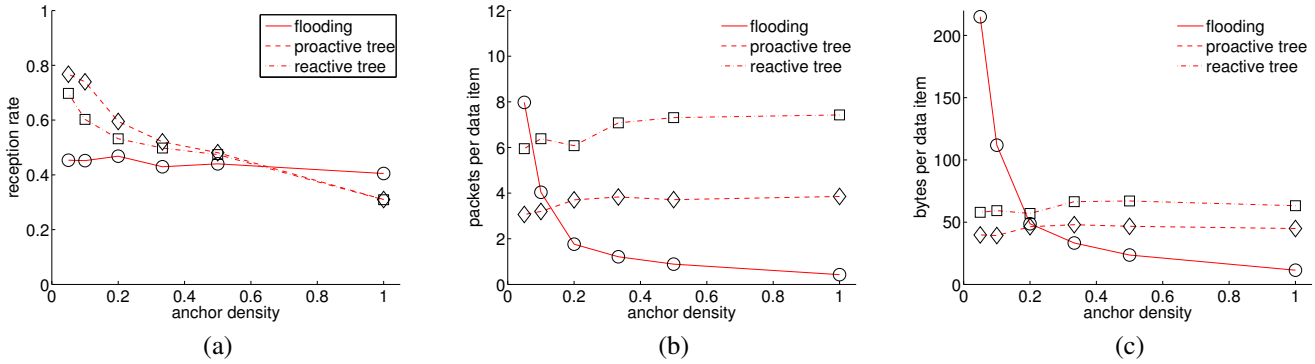

Fig. 1. Simulation results as a function of anchor density: (a) yield (b) packet overhead (c) byte overhead.

will have a significant impact on the performance of our data collection protocols.

A further observation is that with the tree approach, some nodes are not included in a spanning tree (even though they should be included) due to packet loss during tree construction. This is especially bad for yield, as these nodes will never contribute data items. When considering only nodes that are actually included in the spanning tree for the calculation of the yield, we obtain a maximum yield of about 80% for small anchor densities and a minimum yield of about 60% for high anchor densities for the proactive variant. Comparing these values with those reported in Fig. 1 (a), we conclude that failure to include all required nodes in a spanning tree substantially contributes to low yield for high anchor densities.

Finally, the reactive tree variant has a significantly larger overhead than the proactive variant due to the request messages and results in a lower yield because a lost request message prevents nodes from participating in data collection.

V. Related Work

In summary, the present paper is the first effort to compare Push and Pull strategies for data aggregation over overlapping *multi-hop* network neighborhoods. In [7], a similar comparison is performed for *single-hop* network neighborhoods, a much simpler problem as direct communication between producers and consumers of data items is always possible, eliminating the need for in-network aggregation. Similarly, Hood [14] supports data collection only from single-hop neighborhoods. In [4], a protocol is studied for routing data from multiple sources to multiple sinks across multiple hops. However, while we consider network neighborhoods, [4] assumes that sources and sinks are globally distributed over the network and in-network data aggregation is not performed. Finally, Abstract Regions [13] and Logical Neighborhoods [8] both support data aggregation over multi-hop neighborhoods, but the authors do not compare pull and push approaches.

VI. Conclusions

We studied the problem of data aggregation from overlapping multi-hop network neighborhoods, where nodes have to contribute its data items to multiple network neighborhoods for aggregation. We considered two orthogonal approaches to this problem, a Push approach based on scoped flooding, and a Pull approach based on spanning trees. We implemented these protocols in TOSSIM to study yield and overhead as a function of anchor density, i.e., the fraction of nodes in the network that perform data aggregation over a multi-hop network neighborhood. We find that the Push approach has lower overhead than the Pull approach unless anchor density is very low. However, both approaches have a surprisingly low yield due to the characteristics of the underlying MAC protocol, motivating the need for MAC protocols with better support for bursty traffic patterns.

References

[1] R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, L. Krishnamurthy, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and Deployment of Industrial Sensor Networks: Experiences from the North Sea and a Semiconductor Plant. In *Sensys 2005*, San Diego, USA, November 2005.

[2] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. A Collaborative Approach to In-Place Sensor Calibration. In *IPSN*, Palo Alto, USA, April 2003.

[3] J. Choi, J. Lee, M. Wachs, and P. Levis. Opening the sensornet black box. Technical Report SING-06-03, Stanford Information Networks Group, 2006.

[4] P. Ciciriello, L. Mottola, and G. P. Picco. Efficient routing from multiple sources to multiple sinks in wireless sensor networks. In *EWSN 2007*, Delft, The Netherlands, January 2007.

[5] C. Frank and K. Römer. Algorithms for generic role assignment in wireless sensor networks. In *Sensys 2005*, San Diego, USA, November 2005.

[6] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative Support for Sensor Data Cleaning. In *Pervasive 2006*, Dublin, Ireland, May 2006.

[7] A. Lachenmann, P. J. Marrón, D. Minder, O. Saukh, M. Gauger, and K. Rothermel. Versatile support for efficient neighborhood data sharing. In *EWSN 2007*, Delft, The Netherlands, January 2007.

[8] L. Mottola and G. P. Picco. Logical neighborhoods: A programming abstraction for wireless sensor networks. In *DCOSS 2006*, San Francisco, USA, June 2006.

[9] J. Polastre, J. L. Hill, and D. E. Culler. Versatile low power media access for wireless sensor networks. In *SenSys 2004*, Baltimore, USA, November 2004.

[10] C. Renner. Local data aggregation for sensor networks with multiple sinks. Technical report, TU Hamburg-Harburg, April 2007.

[11] K. Römer. Discovery of Frequent Distributed Event Patterns in Sensor Networks. In *EWSN 2008*, Bologna, Italy, January 2008.

[12] L. F. W. van Hoesel and P. J. M. Havinga. A Lightweight Medium Access Protocol (LMAC) for Wireless Sensor Networks. In *INSS*, Tokyo, Japan, June 2004.

[13] M. Welsh and G. Mainland. Programming Sensor Networks Using Abstract Regions. In *NSDI 2004*, Boston, USA, March 2004.

[14] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *MobiSys*, Boston, USA, June 2004.

[15] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys 2003*, 2003.