Cushion: Autonomically Adaptive Data Fusion in Wireless Sensor Networks

Jungmin So Jintae Kim Indranil Gupta

Department of Computer Science,
University of Illinois at Urbana-Champaign
{jso1, kim28, indy}@cs.uiuc.edu

Technical Report
August 2005

Abstract

Existing protocols for in-network fusion of data, in wireless sensor networks, are either single-path (i.e., tree-based) or multi-path. Tree-based fusion protocols have small message overhead but low reliability under node and link failures, while multi-path protocols have good reliability, but potentially higher overhead. This paper presents a suite of protocols, called Cushion, which automatically adapts message overhead in order to maintain a desired level of reliability. As a part of Cushion, we present (1) a simple adaptive aggregation protocol as well as (2) a node-aware adaptive aggregation protocol. Both these protocols use special control messages that enable nodes to decide when to transition between using single path and multi-path fusion approaches. The protocols span a continuous spectrum between these two approaches and can further increase reliability over multi-path approach using redundant transmissions. We present experimental results quantifying the benefits and drawbacks of using a Cushion-based approach versus both tree-based and multi-path approaches.

1. Introduction

Data fusion is an important service for sensor network applications. The goal of data fusion (or aggregation) is to aggregate data from independent sensors to compute useful information, such as the average of all the sensor readings, the maximum value among the sensor readings, or the number of sensors that detect an event. In data aggregation, reliability is often measured by the participation ratio, i.e., the number of participating nodes over the total number of nodes.

The data aggregation challenge addressed in this paper is: how does one achieve a desired level of reliability at all times, with low message overhead, in spite of time-varying unreliability in the underlying networks? In other words, can we design an *autonomically* and *automatically* adaptive protocol that maintains a high reliability, even as nodes fail and recover, and as the lossiness of links varies over time, without the need to configure parameters online or the need for human intervention? In addition, energy conservation requires the adaptive protocol to have low control overhead and actual aggregation message overhead.

In a typical in-network aggregation problem, data originates from multiple *source nodes*, and moves towards single *sink node* or root node. Along the way, i.e., *inside* the network, the data may be partially aggregated, thus reducing message overhead. Two well-known classes of solutions to this problem are: tree-based aggregation and multi-path aggregation. Our autonomically adaptive data fusion solution, called *Cushion*, contains two protocols that span a continuous spectrum between these two design points and can further increase reliability over multi-path approach using redundant transmissions. We motivate the design of Cushion by first discussing the complementary advantages and disadvantages of the two design points.

Tree-Based Aggregation: In tree-based approaches, e.g., [9, 12], a tree is constructed and maintained, with the sink node as the root. Data converges from the source nodes towards the sink along the edges of the tree. However, unreliable channels, energy depletion, and node failures, can result in momentary loss of entire subtrees of readings, while the tree is reorganized. Failures closer to the root affect the reliability very drastically. Finally, if the rate of failures is comparable to the rate of tree reorganization, a *thrashing-like* behavior could produce a consistently

low reliability all the time.

Multi-Path Aggregation: On the other hand, multi-path data fusion protocols take advantage of broadcast communication in the sensor networks [3, 13]. Reliability can be improved by broadcast communication because each node can send data to multiple parent nodes without incurring additional overhead. While this approach inevitably accompanies the problem of detecting duplicate messages from a node, Flajolet and Martin's counting algorithm [4] provides the framework to solve this duplicate problem. When all nodes in the network (except the sink) are sources, then the overhead of multi-path aggregation is same as tree-based aggregation (each node transmits once). However, if only a subset of nodes are sources, then multi-path aggregation may have larger overhead compared to tree-based aggregation, since nodes that are not part of the tree may participate in forwarding packets.

New Approach in Cushion: In this paper, we present two new aggregation protocols that adopt a *probabilistic approach* to realize autonomic and automatic adaptivity. Initially, a spanning tree rooted at the sink node is established in the network. All nodes except the sink are classified into two types: nodes along the spanning tree whose leaf is a source, or nodes outside the spanning tree. The most important parameter, for achieving adaptivity, is the *redundancy level* $p(> 0)$. When $p = 0$, the Cushion protocols operate as a tree-based protocol. In this case, node that are outside the spanning tree do not forward the packets from the source. (Nodes that are on the spanning tree forward packets from the source, regardless of p.) If the sink node does not obtain the desired reliability due to failures in the network, the sink attempts to increase the redundancy levels of the sources in the system. On the other hand, if reliability rises above the target reliability, e.g., due to node recoveries, the sink reduces the redundancy level. As a result of this reduction, the system might converge to a tree-like configuration, when the channel condition allows the sink to achieve the target reliability. As a result of this, the emergent behavior throughout the system is a minimization of the message overhead while maintaining a certain level of reliability, even as nodes fail and links become lossy.

If the redundancy level at each source in the system is increased to $p = 1$, the Cushion protocols reduce to a multi-path protocol, where multiple parents forward packets from the source to increase reliability. When $0 < p < 1$, nodes outside of the spanning tree forwards a packet with probability p, and thus message overhead is adjusted between the tree-based and multi-path approach in order to satisfy the required reliability at a necessary cost. If the desired level of reliability is not met even when $p = 1$, then p can further be increased over 1. When $p > 1$, it translates into *redundant transmission*. For example, if $p = 3$, then each intermediate node (including nodes on the spanning tree) transmits a packet three times, to further increase reliability. The adaptive behavior of Cushion is illustrated in Figure 1.

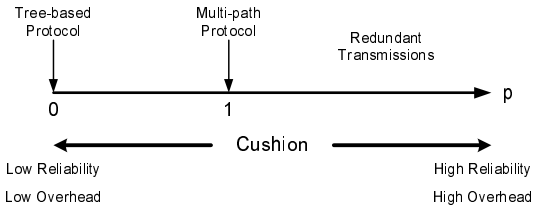

Figure 1. Illustration of adaptive behavior of Cushion. The Cushion protocols achieve desired reliability level by controlling message overhead using the redundancy level p.

Depending on whether the sink node adjusts the redundancy level on a system-wide basis (i.e., at all sources) or on a per-source basis, we obtain two flavors of adaptive aggregation protocols. We present both flavors, and then experimentally compare them to pure tree-based and multi-path-based approaches. The details of the protocols will be described in Section 4.

The remainder of the paper is structured as follows. In Section 2, we briefly introduce the related work. In Section 3, we present the basic schemes used in designing our protocols. In Section 4, we describe the adaptive behavior of our proposed Cushion protocols. In Section 5, we discuss the experimental results regarding the overhead and reliability in various settings. Finally, we conclude with directions for future work in Section 6.

2. Related Work

Directed diffusion [9] is a data communication mechanism proposed for sensor networks. Although the main goal of directed diffusion is to establish efficient communication between sources and sinks, it performs data aggregation by using the spanning tree built for communication. As an extension to this work, an algorithm for selecting the aggregation points is proposed, which tries to reduce energy consumption by using low-cost paths [8]. Another algorithm for placing aggregation points is proposed in [11], where each node can transfer the aggregation role to a neighbor if the neighbor is a "better" point of aggregation according to a cost function. Intelligent placement of aggregation points can improve energy efficiency.

There have been previous studies on increasing reliability. Karl et al. [10] suggested using different level of

forward error correction (FEC) according to the amount of aggregation in a message. Other approaches are mainly concerned with (1) utilizing multiple paths to increase reliability, and (2) handling redundant messages caused by multiple paths. In TAG [12], multiple paths are used to increase reliability, so that a node can divide the duplicate-sensitive aggregates, e.g. COUNT, into multiple values and send each of them to the multiple parents. While TAG uses unicast as a communication method, even more reliability can be achieved by exploiting broadcast communication. Broadcast communication provides redundancy without incurring additional overhead by allowing each node to send data to multiple "parent" nodes. Considine et al. [3] utilized Flajolet and Martin's counting algorithm [4] to construct messages that can detect duplicates and successfully aggregate the data. The algorithm for computing SUM is presented in [3]. Synopsis diffusion [13] extends Considine's work and use the order and duplicate insensitive (ODI) synopses to compute various aggregates.

There are also other data aggregation schemes that are not based on tree structures, yet try to achieve good fault-tolerance. [17, 2] proposed a scheme for aggregating duplicate-insensitive data such as MAX or MIN, where each node periodically communicates with neighbors to update its data until the global agreement reaches. A random walk technique for data aggregation has been proposed in [1]. Gupta et al. [6] proposed a topology-aware data aggregation protocol for large process groups by building a hierarchy and using gossiping on each level of the hierarchy.

Finally, there are several adaptive approaches proposed to improve reliability. In synopsis diffusion [13], the "level" of a node can be changed in order to increase or decrease redundancy (the Adaptive Ring topology [13]). Rodrigues et al. [14] proposed a dynamic feedback mechanism to improve reliability in a gossip-based broadcast algorithm. Gupta et al. [5] proposed protocols for gossip-based multicast dissemination. In their approach, multicast is integrated with gossiping, which provides good probabilistic reliability with additional copies of the original multicast. Although our primary concern is the trade-off between reliability and message overhead, it is noteworthy that adaptive protocols such as SPIN [7] and T-MAC [15] have been proposed to reduce energy consumption by reducing message overhead.

3. Designing the Cushion Protocols

The goal of the Cushion protocols is to maintain a desired level of reliability (participation ratio), regardless of dynamically changing network conditions, by controlling the amount of message overhead. Moreover, the amount of message overhead should not be more than needed for achieving the desired reliability level.

In Section 1, we briefly described two classes (tree-based and multi-path) of aggregation protocols. The Cushion protocols move between the two protocols based on channel condition, by controlling the message overhead. Also, the Cushion protocols can achieve a reliability higher than that of a multi-path protocol using redundant transmissions. For a tree-based protocol, the spanning tree rooted at the sink node has to be established and maintained, and the routing and aggregation strategy needs to be defined. For a multi-path protocol, the multi-path routing scheme must be designed, and also duplicate messages need to be detected and removed for duplicate-sensitive aggregates (e.g. COUNT, AVG). We describe these schemes briefly in the following, which will be the components of Cushion protocols. The terms defined here will also be used in Section 4, where we focus more on the adaptive algorithms in Cushion.

Tree Establishment: The sink node initializes the tree establishment phase by broadcasting a QUERY message. A QUERY message includes the sender's id, its parent's id, and the number of hops from the sink which is called *depth*. (The sink node does not have any parent, so it has a null value for the parent's id.) When a node receives a QUERY message, it selects the first sender as its parent, and rebroadcast the packet, with the *depth* incremented by 1. When the node rebroadcasts the packet, its parent node overhears the packet and remembers the node as its child by recording in its *child-list*. So at the end of the tree establishment phase, each node knows its parent node, its depth in the tree, and its child nodes in the tree. The ANSWER messages (messages including data) from the sources will be sent along the established tree.

Routing and Aggregation: We use the routing and aggregation strategy similar to TAG [12]. Time is divided into *epochs*, where a single data aggregation process (answers from sources are collected at the sink) takes place. An epoch is again divided into *rounds*, where the number of rounds in each epoch is larger than the maximum depth d of the tree. At the start of ith round, nodes with depth $d-i$ transmit their ANSWER packets. At the end of each epoch, the answers from the sources are collected at the sink node.

Multi-path Routing: The multi-path routing in Cushion uses similar ideas with synopsis diffusion [13]. After the tree establishment phase, all nodes in the network know their hop distance from the sink (*depth*). The idea of multi-path routing is to have a node aggregate and forward all packets coming from nodes with larger depth. For example, if the source node has the depth l, then when it sends

the ANSWER packet, all nodes that have depth $l-1$ who receives the packet, forwards it towards the sink.

Tree Maintenance: Due to failures or mobility, a path from a source to the sink may break. If this causes the reliability to drop, the Cushion will start using multi-path routing to increase the reliability. Since a node can detect whether its parent node is forwarding its packets or not, it can choose to switch the parent, if the current parent node is not reliably forwarding its packets.

Suppose node A has depth $l-1$, and node B has depth l in the spanning tree. Node A is selected as B's parent. When B sends its packet to A, A aggregates packets from its child nodes and sends a packet to its parent node. B can overhear this packet and check whether A has received its packet. If multi-path routing is used, B can detect who has received and forwarded its packet, and who has not. If the link between B and A breaks, B will notice that A is not receiving and forwarding its packet. If A does not receive B's packet for τ epochs, B chooses to re-select a parent among its neighboring nodes with depth $l-1$. Suppose node B has selected node C as its new parent. Then, this information is piggybacked in the ANSWER packet, so that node A can remove B from A's child-list, and C can include B in C's child-list.

Duplicate Detection: When packets are forwarded via multiple routes, an answer can be counted multiple times. For duplicate-sensitive aggregates (e.g. COUNT, AVG, SUM), the duplicates need to be detected and removed. For duplicate detection, we apply the Order- and duplicate-insensitive (ODI) synopsis technique used in [13].

4. Cushion: Adaptive Data Fusion Protocols

In the previous section, we have described the basic building blocks for designing the Cushion protocols. Here we focus on how the two protocols of Cushion adapt to the network condition in order to maintain the desired reliability.

The main idea is to have the sink node measure the reliability level, and control the message overhead by sending out CONTROL messages to the network. In particular, the sink node controls the *redundancy level* $p(>0)$ of the sources in order to control the message overhead.

According to the redundancy level p, the protocols behave as the following.

- $p = 0$: The protocols behave similar to a tree-based protocol. Only the nodes in the spanning tree forward the packets, in its corresponding round that matches its depth. (A *round* is defined in the previous section.) Other nodes do not forward packets.
- $0 < p < 1$: Nodes that are not on the spanning tree forwards the packet with the probability p in its corresponding round. The nodes in the spanning tree still forward with 100% probability.
- $p = 1$: All nodes that receive packets in the previous round forward the packet in the corresponding round. The behavior of the protocols becomes similar to [13].
- $p > 1$: Nodes forward packets more than once. That is, each node transmits the packet with 100% probability in $\lfloor p \rfloor$ rounds, and transmits with probability $p - \lfloor p \rfloor$ in the following round. For example, if the redundancy level is 2.5, then each node transmits the aggregated message in its corresponding round, transmits once more in the next round, and transmits with 50% probability in the following round.

Figure 2 shows a simple example of how the Cushion protocols behave according to the redundancy level. In the figure, the redundancy level is set to 0.5. When node S sends a packet, B forwards the packet since it is the parent node of S in the spanning tree. Other nodes, A and C, forward the packet with 50% probability.

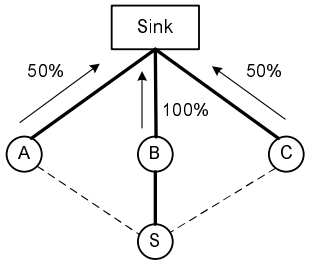

Figure 2. An example scenario illustrating the adaptive behavior of Cushion, when the redundancy level is 0.5. The solid lines are the edges the spanning tree.

The reliability and overhead are both increased when the redundancy level increases, and drops down as the redundancy level is decreased. So redundancy level is the key parameter in controlling overhead to maintain desired reliability level in Cushion. Depending on how the redundancy level is assigned to sources, there are two flavors of Cushion protocols: Simple adaptive protocol and Node-aware adaptive protocol.

4.1. Simple Adaptive Fusion in Cushion

The first protocol of Cushion is called the Simple Adaptive protocol. In the Simple Adaptive protocol, a single redundancy level p is assigned for all nodes. Whenever the

sink node decides to change the redundancy level, it runs the algorithm in Figure 3 and broadcasts the new redundancy level in a CONTROL message. The reliability measure used in our protocols is the *participation ratio*, which is the percentage of nodes that have successfully sent their messages to the sink node.

Given a target participation ratio, the sink node tries to find the optimal redundancy level. To do this, the sink node monitors the reliability level while tuning the redundancy level. Initially, the redundancy level p is set to 1, meaning that the protocol operates as a multi-path protocol. If the sink node achieves the target reliability, it might be possible that the message overhead is higher than what is needed. In this case, the sink node reduces p to reduce the overhead. When the sink keeps reducing p, p eventually becomes zero, and the protocol converges to having a tree-like behavior. On the other hand, if the sink node does not achieve the target participation ratio, it increases p to increase the participation ratio. p is increased until the target reliability level is obtained.

The protocol needs to decide how fast it should increase and decrease the redundancy level. The requirements depend on applications, but we choose to be conservative and prefer to achieve the target reliability quickly when the reliability level is lower than the target. Thus, we use a Multiplicative Increase Additive Decrease (MIAD) approach, where p is increased multiplicatively and decreased additively.

The algorithm for the Simple Adaptive protocol is described in Figure 3. We assume that the sink node knows the number of sources in the network. (The sink node can count the sources that have recently sent a message to the sink and regard the number as the total number of nodes.) The granularity of changing redundancy level is a protocol parameter, which is denoted as c. (In our simulations, we use $c = 0.25$.) When the sink node does not obtain the target reliability, it computes the new p as $p_{new} = p + fc$, where initially $f = 1$. If the target reliability is still not achieved in the next epoch (an *epoch* is defined in Section 3), then f is doubled, which results in multiplicative increase. Once the desired reliability is achieved, f is reset to 1.

Now when the desired reliability is achieved, the overhead might be unnecessarily high. So the sink node tries reducing the redundancy level slowly. Unlike the method of increasing the redundancy level, the sink decreases the redundancy level as $p_{new} = p - c$, which results in additive decrease. The redundancy level is decreased until the reliability goes below the desired level, and is increased again. Note that in Figure 3, p and f are global variables.

Once the sink node decides to change the redundancy level, it broadcasts a CONTROL message at the beginning of an epoch, including the new redundancy level. If the

Algorithm *AdaptiveSingleEpoch*
Input: desired reliability level r_0, granularity of redundancy level c
(∗ Our first adaptive protocol in each epoch ∗)
1. $r \leftarrow$ most recently measured reliability
2. **if** $r = r_0$ **then stop**
3. **if** $r < r_0$
4. **then** $p \leftarrow p + fc$
5. $f \leftarrow 2f$
6. $CONTROL \leftarrow$GenerateControlMessage(p)
7. Broadcast $CONTROL$
8. **else** $p \leftarrow p - c$
9. $f \leftarrow 1$
10. $CONTROL \leftarrow$GenerateControlMessage(p)
11. Broadcast $CONTROL$

Algorithm *Adaptive*
Input: desired reliability level r_0, granularity of redundancy level c
(∗ The overall first adaptive protocol ∗)
1. $f \leftarrow 1$
2. **repeat**
3. AdaptiveSingleEpoch(r_0, c)
4. **until** $r = r_0$

Figure 3. Cushion's Simple Adaptive Protocol: Every source has the same redundancy level. The sink node runs this algorithm and broadcasts the new redundancy level in a control message.

sink node changes redundancy level at the end of every epoch, **a CONTROL message will be broadcasted in every epoch, which may incur a significant overhead**. To reduce the CONTROL message overhead, we use the following scheme. Suppose the sink increases the redundancy level with $f = 1$. If the desired reliability is achieved in the next epoch, it means that the redundancy level is likely to be fine-tuned to the network condition. In this case, the sink node does not send CONTROL messages for h epochs, if the reliability does not fall below the desired level during those epochs. If the reliability goes below the desired level, the sink immediately starts increasing the redundancy level. Using this scheme, **the CONTROL messages are sent frequently when the network is dynamically changing, and rarely sent when the network is static**, after the redundancy level is fine-tuned to current network condition. It is typical that sensor networks have static topology, and so the network condition does not change drastically all the time. Thus, we expect that the CONTROL messages are sent infrequently.

4.2. Node-Aware Adaptive Fusion in Cushion

The second protocol of Cushion, called Node-Aware Adaptive protocol, adds some complexity to the Simple Adaptive protocol to improve efficiency. In the Simple Adaptive protocol, a single redundancy level p was maintained for all sources. However, if the network condition is not uniform across the sensor networks, it may be more efficient to assign different redundancy level for each source. For example, suppose only a specific region has higher packet loss rate, due to some reasons. To obtain participation from nodes in this area, the Simple Adaptive protocol may unnecessarily increase the message overhead for sources outside the failure-prone area. In this case, it is more efficient to assign high redundancy level to sources in the failure-prone area, and low redundancy level to other sources.

In Node-Aware Adaptive protocol, a separate redundancy level is assigned for each node, by the sink. When sending a packet, the source piggybacks the redundancy level in the packet, so that the intermediate nodes can figure out the forwarding strategy for the packet. An intermediate node may receive packets from multiple sources and aggregate them before forwarding towards the sink. In this case, the redundancy level for the particular packet is chosen as the *maximum* redundancy level among the sources participated in the aggregated packet.

The sink node maintains the redundancy level for each source, and also the *history* of aggregation for the last w epochs. The parameter w is called *history window*. (We use $w = 20$ in our simulations.) The history records whether a source has participated in the data aggregation or not, in each epoch. Using the history, the sink node can find out the participation rate of each source in the recent past.

The algorithm for Node-Aware Adaptive protocol is described in Figure 4. When the target reliability is not obtained, the sink node increases the redundancy level to increase the reliability. In Node-Aware protocol, instead of having all nodes assigned a new redundancy level, only m nodes with the worst participation rate are assigned an increased redundancy level. To inform the sources, the sink node includes in the CONTROL message the set of nodes that are assigned new redundancy level, and the redundancy levels for each of those nodes. On the other hand, when the desired reliability level is achieved, the sink node starts reducing the redundancy level. In Node-Aware Adaptive protocol, the sink node selects m nodes with the highest redundancy levels, and decreases their redundancy level. The reason behind this strategy is that when the reliability increases, it might be due to improved network conditions in the failure-prone area. However, this strategy may not be efficient if the reliability is improved because the network condition is further improved in the areas other than

Algorithm *NodeAdaptiveSingleEpoch*
Input: desired reliability level r_0, granularity of redundancy level c
1. $r \leftarrow$ most recently measured reliability
2. **if** $r = r_0$ **then stop**
3. **if** $r < r_0$
4. **then** n_{r_i} = the node with ith worst reliability.
5. p_{r_i} = the redundancy level of n_{r_i}.
6. $target\_nodes \leftarrow \{n_{r_i} | 1 \leq i \leq m\}$
7. $p_{r_i} \leftarrow p_{r_i} + fc$
8. $f \leftarrow 2f$
9. $CONTROL \leftarrow$GenerateControlMessage((n_{r_1}, p_{r_1}), $\cdots$, (n_{r_m}, p_{r_m}))
10. Broadcast $CONTROL$
11. **else** p_i = the redundancy level of a node n_i.
12. $p = \max_{1 \leq i \leq N} p_i$
13. $target\_nodes \leftarrow \{n_i | p_i = p\}$
14. $k \leftarrow |target\_nodes|$
15. $p \leftarrow p - c$
16. $f \leftarrow 1$
17. $CONTROL \leftarrow$GenerateControlMessage((n_1, p), $\cdots$, (n_k, p))
18. Broadcast $CONTROL$

Algorithm *NodeAwareAdaptive*
Input: desired reliability level r_0, granularity of redundancy level c
($*$ The overall second adaptive protocol $*$)
1. $f \leftarrow 1$
2. **repeat**
3. NodeAdaptiveSingleEpoch(r_0, c)
4. **until** $r = r_0$

Figure 4. Cushion's Node-Aware Adaptive Protocol: The sink node assigns a different redundancy level for each source. Similar to the Simple Adaptive Protocol, the sink broadcasts the new assignment in a control message.

the failure-prone region. Since the decreasing strategy always chooses the nodes with highest redundancy levels, it may not be able to further reduce the redundancy level of areas with better network condition. This issue is further discussed in Section 5, where we compare the performance of Simple Adaptive and Node-Aware Adaptive protocols.

5. Performance Evaluation of Cushion

We have evaluated the performance of the proposed Cushion protocols using simulations. In this section, we describe our simulation setup and discuss the results.

5.1. Simulation Setup

We have used the ns-2 simulator [16] with wireless extensions for our simulations. The sensor nodes have transmission range of 10m, and they are randomly placed in a 50m $\times$ 50m area (the number of nodes are varied in the simulations). A sink node is placed in the center, which broadcasts a QUERY message every 10 seconds to gather information from the source nodes. The source nodes reply to the QUERY message by sending ANSWER messages at each *epoch*. Each epoch is 1 second long, and the maximum network diameter D is set to 20. The duration of each *round* in an epoch is calculated as T_{epoch}/D, so the duration of a round in our simulations is 50ms. (The epochs and rounds are defined in Section 3.)

The type of aggregate we use in our simulations is COUNT, which calculates the number of sources. Since COUNT is a duplicate-sensitive aggregate, our protocols can be applied to other types of aggregate as well. For medium access, CSMA (Carrier Sense with Multiple Access) protocol is used.

Under this simulation setup, we have evaluated our proposed protocols, ADAPTIVE and N-ADAPTIVE. ADAPTIVE refers to the Simple Adaptive protocol described in 4.1, and N-ADAPTIVE refers to the Node-Aware Adaptive protocol described in 4.2. For the purpose of comparison, we have evaluated other four aggregation protocols, described in the following.

- TREE: The TREE protocol is similar to [12]. In the TREE protocol, the sink node floods the QUERY message to the network to establish an aggregation tree in the network. Periodically, data from each source is forwarded along the tree, and is aggregated at the intermediate nodes to reduce overhead. To recover from failure, each node may maintain multiple backup parents in the aggregation tree, so that if the parent node does not receive the data packet for a certain amount of time, the node switches its parent to one of its backup parents. However, each node only sends its data to a single parent node.
- SYNOPSIS: The SYNOPSIS protocol is the basic protocol (which does not use adaptive rings) in [13]. For simplicity, we did not use the synopsis generation techniques described in [13], but concatenated all information when aggregating messages. So the error from generating and aggregating synopsis is not modeled here.
- A-SYNOPSIS: The A-SYNOPSIS protocol is the protocol that uses adaptive rings in [13].
- FLOOD: FLOOD protocol is also described in [13]. In FLOOD protocol, there is no tree structure in the network. In each round, every node broadcasts its message to its neighbors. If the new data is received, the node transmits the aggregated message once again in the next round. So every node usually transmits multiple times in the FLOOD protocol.

For TREE, ADAPTIVE and N-ADAPTIVE, the number of transmission failure before a node switches to a different parent is set to 4. For A-SYNOPSIS, the probability of switching levels is set to 0.5. For ADAPTIVE and N-ADAPTIVE, the number of epochs before reducing the overhead (h) is set to 10. Also, the granularity in changing redundancy (c) is set to 0.25, unless otherwise specified. Finally, the target reliability for ADAPTIVE and N-ADAPTIVE is set to 0.9 (90%). For additional parameters of TREE, SYNOPSIS and A-SYNOPSIS protocols, refer to [12] and [13] respectively. The parameters for the ADAPTIVE and N-ADAPTIVE are defined in Section 4.

In order to evaluate reliability and overhead, we use the following two metrics:

- Average participation ratio: For each epoch, we measure the participation ratio as the number of participated sources (sources that the sink received data from) divided by the total number of sources. Average participation ratio is the participation ratio averaged over all epochs during the simulation time. This is a measure of reliability.
- Average number of transmissions in each epoch: We count the number of packet transmissions in each epoch to measure the overhead. For our proposed Cushion protocols, control messages are also counted in this metric.

Finally, total simulation time for each simulation is 100 seconds, and each data point in the graphs is a result of 10 runs with different random seeds.

5.2. Results

Impact of Packet Loss: In the first simulation, 100 nodes were randomly placed in the simulation area, and 25

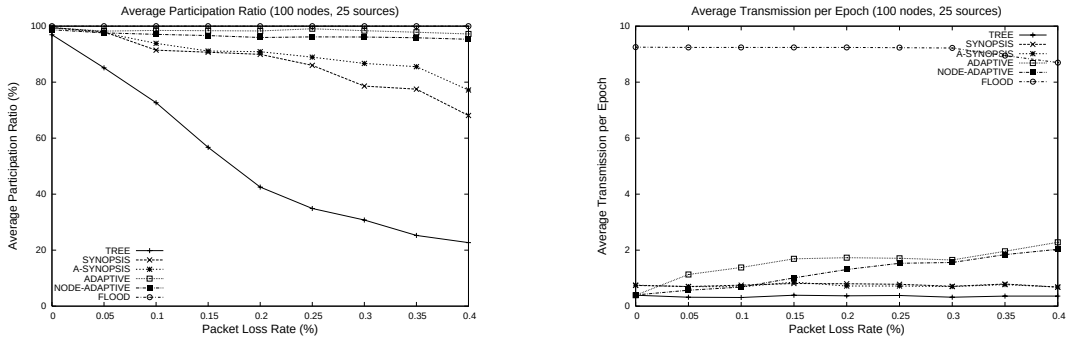

Figure 5. Average participation ratio (reliability) and average transmissions per epoch (overhead) of protocols for the scenario with 100 nodes and 25 sources.

sources were randomly picked among the nodes. We have varied the *packet loss rate* on each link to see the impact of packet loss on the performance of protocols. In this experiment, the packet loss rate is uniformly applied to all links in the network. For example, if the packet loss rate is q, then when a packet is transmitted, the probability that the receiver will correctly receive the packet is maximum $1-q$ for each transmission on any link. Since the packets might collide, the actual packet reception rate may be lower. For each protocol, we have measured the average participation ratio and the average transmissions per epoch. The results are shown in Figure 5.

Figure 5(a) shows the average participation ratio, which is a measure of reliability. FLOOD, ADAPTIVE, and N-ADAPTIVE maintain participation ratio over 90%, even when the packet loss rate increases up to 40%. The reliability of the TREE protocol drops down at the fastest rate, followed by SYNOPSIS and A-SYNOPSIS. For TREE, if the number of hops from the source to the sink is k, then the probability that the sink will successfully receive the packet from the source is $(1-q)^k$, which decreases rapidly as q increases. SYNOPSIS and A-SYNOPSIS use multipath forwarding to increase reliability, but they fail to maintain the desired reliability when the packet loss rate becomes high. Even when SYNOPSIS and A-SYNOPSIS fall below the desired reliability, ADAPTIVE and N-ADAPTIVE still maintain high reliability using redundant transmissions. This behavior of ADAPTIVE and N-ADAPTIVE is reflected in Figure 5(b), where the overhead of the protocols is shown. TREE, SYNOPSIS, A-SYNOPSIS and FLOOD show no change in the overhead as the packet loss rate changes. For ADAPTIVE and N-ADAPTIVE, when the packet loss rate is low so that even TREE can achieve the desired reliability, ADAPTIVE and N-ADAPTIVE behave similar to TREE and thus use similar overhead as shown in the figure. When the packet loss rate increases, the protocols increase the redundancy by having multiple intermediate nodes aggregate and forward packets from sources. So up to a certain point of packet loss rate, ADAPTIVE and N-ADAPTIVE shift from a TREE-like protocol to a SYNOPSIS-like protocol as the packet loss rate increases. If the loss rate increases even higher, ADAPTIVE and N-ADAPTIVE shift from SYNOPSIS-like protocol towards a FLOOD-like protocol by having each node send the same packet multiple times. However as seen in Figure 5(b), ADAPTIVE and N-ADAPTIVE use much less overhead than FLOOD while maintaining similar level of reliability as FLOOD, which means that after some point, more redundancy does not help towards increasing reliability.

Figure 5 also shows the difference between ADAPTIVE and N-ADAPTIVE. While ADAPTIVE and N-ADAPTIVE show similar reliability, N-ADAPTIVE requires less overhead than ADAPTIVE. N-ADAPTIVE can achieve comparable reliability as ADAPTIVE while using less overhead because it can increase redundancy of only the sources that have higher loss rate. The performance difference between ADAPTIVE and N-ADAPTIVE will be examined again later when we assign packet loss rate on links according to a non-uniform spatial distribution.

Impact of Node Density: In the next simulation, we have changed the node density to study its impact on protocol performance. 50 nodes were placed in the simulation area and 12 sources were randomly picked among the nodes. The results are shown in Figure 6.

The shape of the curves looks similar to Figure 5, but several differences can be found. In Figure 6(a), the performance degradation of the SYNOPSIS and A-SYNOPSIS is much quicker than Figure 5. This is intuitive, because SYNOPSIS and A-SYNOPSIS benefit from having multiple paths from sources to the sink. When the network is sparse, SYNOPSIS and A-SYNOPSIS cannot benefit

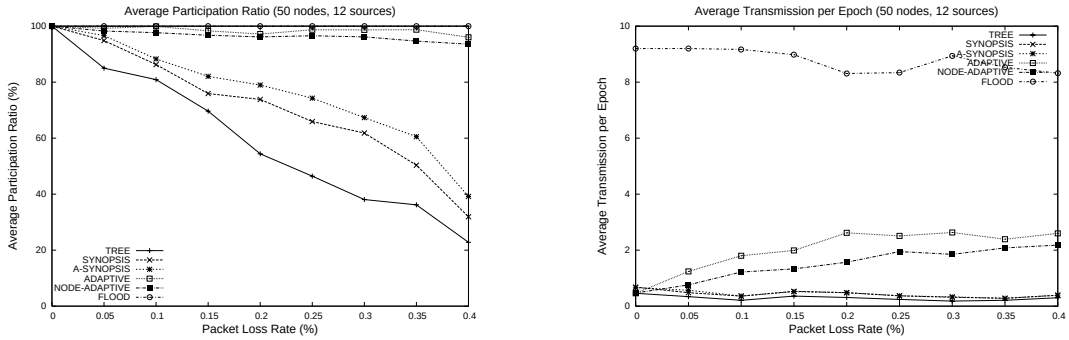

Figure 6. Average participation ratio (reliability) and average transmissions per epoch (overhead) for the scenario with 50 nodes and 12 sources.

much from redundancy, and thus their performance degrades quickly. However, ADAPTIVE and N-ADAPTIVE maintain reliability even in the sparse network scenario by paying a higher cost, as shown in Figure 6. Thus, the overhead of ADAPTIVE and N-ADAPTIVE are higher than the scenario in Figure 5.

Impact of Spatial Distribution in Link Quality: In the third simulation, instead of setting the packet loss rate fixed for all links, we only applied the packet loss rate to the upper-right quadrant of the area. Links in other areas have zero packet loss rate. The purpose was to see the difference between ADAPTIVE and N-ADAPTIVE, because ADAPTIVE does not consider per-source reliability to control redundancy of each source, whereas N-ADAPTIVE keeps track of reliability measure for each source and controls redundancy at per-source level. The result is shown in Figure 7.

In Figure 7(a), we can see that the participation ratio of SYNOPSIS, A-SYNOPSIS, ADAPTIVE and N-ADAPTIVE are all similar. However, ADAPTIVE and N-ADAPTIVE use less overhead than SYNOPSIS and A-SYNOPSIS while achieving comparable reliability, as shown in Figure 7(b). N-ADAPTIVE uses less overhead than ADAPTIVE, but the difference is not so significant around 10-15%. This is mainly due to the behavior of N-ADAPTIVE when *decreasing redundancy level* (details explained in Section 4). When N-ADAPTIVE decreases redundancy, the first sources to decrease redundancy level are the ones with the highest redundancy levels. However, in this scenario, the redundancy level of the sources in the upper-right quadrant region must be maintained, although it is higher than other regions. The redundancy level of the sources in the areas other than the upper-right quadrant can decrease their redundancy without losing reliability. This result suggests that it may be better to start decreasing redundancy from the sources that had reliably sent their data for the longest duration in the recent path. We are planning to investigate this design choice in the near future.

Temporal Behavior of the Protocols: Finally, to study the adaptive behavior of ADAPTIVE and N-ADAPTIVE, we have simulated a scenario where at some point of time the packet loss rate increases dramatically, and some time later the network is recovered and the packet loss rate drops down again. 100 nodes were placed in the simulation area with 25 sources chosen randomly among nodes. The simulation is run for 100 seconds. At the start, the packet loss rate is zero at every link. At 30 seconds, the packet loss rate (of every link) increases to 50%. This goes on until 70 seconds, when the packet loss rate becomes zero again. Figure 8 shows the temporal behavior of protocols in this scenario.

In Figure 8(a), it is shown that at 30 seconds, the reliability of all protocols drops down quickly. But after a short while, ADAPTIVE and N-ADAPTIVE quickly recover and maintain the desired reliability, where as other protocols suffer from low participation ratio during the entire period of high packet loss. Figure 8(b) shows the changing overhead of the protocols. Both ADAPTIVE and N-ADAPTIVE increase their overhead to tolerate packet losses, but the increase in ADAPTIVE is more dramatic, because it increases redundancy level for all the sources, instead of only a subset of the sources. N-ADAPTIVE shows more graceful increase in overhead compared to ADAPTIVE, because it chooses a subset of nodes with least reliability and starts increasing only their redundancy levels.

In conclusion, the proposed protocols ADAPTIVE and N-ADAPTIVE in the Cushion framework efficiently adapt to the network conditions to maintain desired reliability at the cost of no more than necessary overhead. N-ADAPTIVE (Node-aware Adaptive protocol) uses less

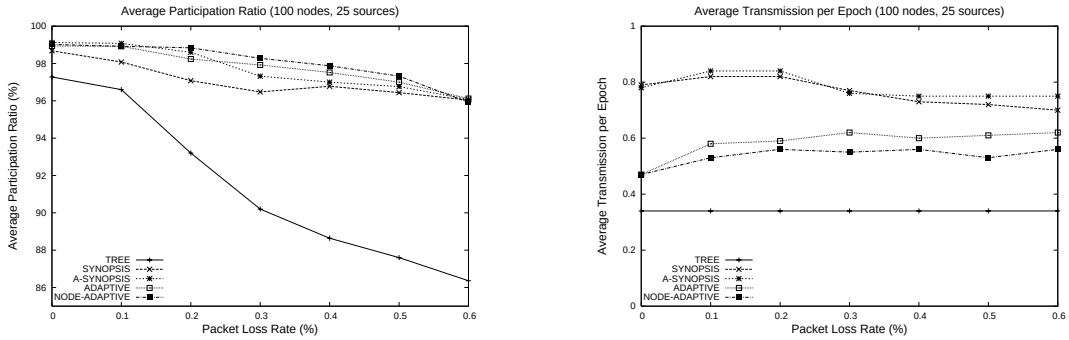

Figure 7. Average participation ratio (reliability) and average transmissions per epoch (overhead). The packet error rate is only applied to the upper right quadrant of the simulation area.

overhead compared to ADAPTIVE (Simple Adaptive protocol) using per-source control of redundancy level.

6. Conclusion

In this paper, we have proposed data aggregation protocols for wireless sensor networks that autonomically and adaptively control redundancy (and thus overhead) in order to maintain a desired level of reliability, in spite of node failure and lossiness of links. In our approach, called Cushion, the sink node adjusts the level of redundant transmissions at the source and intermediate nodes to achieve desired reliability with overhead no more than needed. Simulation results indicate that our protocols adapt to the changing network conditions very well. Compared to tree-based protocols, we showed that our protocols provide substantially high reliability. Compared to the multi-path protocols, our protocols incur substantially less message overhead while achieving comparable reliability.

The biggest advantage of the Cushion approach is that the sink node is capable of self-configuration. Once the desired reliability is set, the sink node automatically takes care of the process for achieving the reliability by monitoring the reliability level and adjusting the redundancy level accordingly. Considering the large-scale nature of sensor networks, it is very important for the protocol to have a self-configuring ability over the changing network conditions. Although some previous works have reactive mechanisms for recovery, we believe that our protocols are the first ones that proactively monitor the status and automatically control parameters in sensor data aggregation.

Future Work: One of the future directions is to consider the significance of each node, which has not been taken into account in our protocols. Nodes in the network have different level of significance. For example, nodes that are closer to the sink are more significant than nodes far away from the sink, because data is highly aggregated at the closer nodes. Also, nodes with less number of neighbors can be more significant than nodes with more neighbors, because packets sent from low-degree nodes benefit less from redundancy. Considering topology and location of each node may lead to more efficient aggregation protocols.

References

[1] C. Avin and C. Brito. Efficient and robust query processing in dynamic environments using random walk techniques. In *Information Processing in Sensor Networks (IPSN)*, 2004.

[2] A. Boulis, S. Ganeriwal, and M. B. Srivastava. Aggregation in sensor networks: An energy-accuracy trade-off. In *IEEE SNPA*, 2003.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximation aggregation techniques for sensor databases. In *20th International Conference on Data Engineering*, March 2004.

[4] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for database applications. *Journal of Computer and System Sciences*, 1985.

[5] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *SRDS*, 2002.

[6] I. Gupta, R. van Ranesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *DSN*, 2001.

[7] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *ACM MOBICOM*, 1999.

[8] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[9] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *ACM MOBICOM*, 2000.

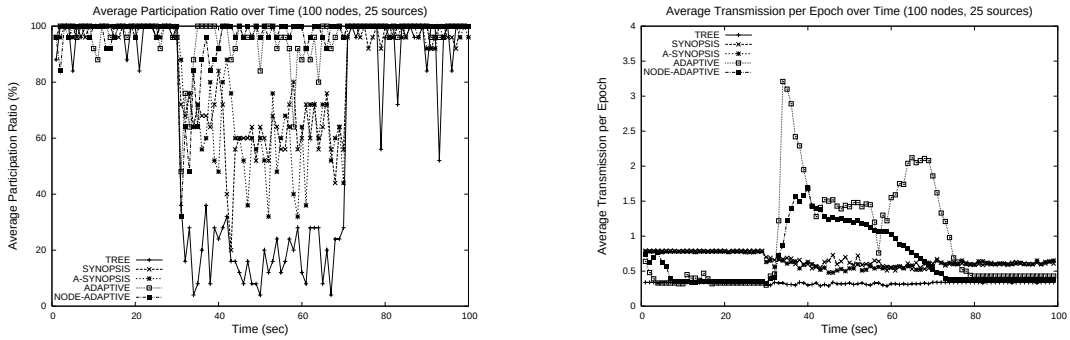

Figure 8. Average participation ratio and average transmissions per epoch versus time. At the start, every link has packet loss rate of 0%. At 30 seconds, the packet loss rate increases to 50%. At 70 seconds, the packet loss rate drops down to 0%.

[10] H. Karl, M. Lobbers, and T. Nieberg. A data aggregation framework for wireless sensor networks. In *Dutch Technology Foundation ProRISC Workshop on Circuits, Systems and Signal Processing*, November 2003.

[11] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran. Dfuse: A framework for distributed data fusion. In *ACM Sensys*, 2003.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad hoc sensor networks. In *OSDI*, 2002.

[13] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *ACM SenSys*, 2004.

[14] L. Rodrigues, S. Handurukande, J. Pereira, R. Guerraoui, and A. Kermarrec. Adaptive gossip-based broadcast. In *DSN*, 2003.

[15] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *ACM SenSys*, 2003.

[16] VINT Group. UCB/LBNL/VINT network simulator ns (version 2).

[17] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *SNPA*, 2003.