

Reliable Actuation in Sensor Networks

Sean Rooney and Luis Garcés-Erice
 IBM Research, Zurich Research Laboratory
 Säumerstrasse 4. CH-8803 Rüschlikon, Switzerland
 Email: {sro, lga}@zurich.ibm.com

Abstract—We present a protocol that uses a publish/subscribe approach to perform reliable but efficient actuation over a sensor network whose topology may change. Actuation on a given group of devices in the sensor network is achieved through a publish operation on the topic the devices in that group are subscribed to. The publication message contains the necessary data to successfully perform the actuation. We make the case for our design showing that: a) suitable data distribution techniques and cross-layer optimizations can reduce transmissions within the messaging layer of the sensor-network b) a soft-state approach can help with the frequent topology changes in wireless sensor networks caused by the transmission medium. We describe our protocol and compare its features and robustness to those of epidemic protocols through simulation. Our protocol is more efficient when the actuation is performed on selected groups of devices within the sensor network. In the general case, the efficiency of our proposal is similar to that of an epidemic model, plus feedback is given to the initiator of the actuation. Robustness remains close to the epidemic approach, even for moderate bit error rates.

Index Terms—Wireless, Sensor Network, Actuation

I. INTRODUCTION

Actuators are devices capable of changing the state of an underlying system in response to control messages received from another entity. Sensors, on the other hand, sense the state of some underlying system and transmit information about that state to some other entity. Descriptions of wireless actuators are less prevalent in the research literature, possibly due to the fact that current efforts have concentrated more on environmental monitoring applications where actuators play no role.

As processors have become cheaper, it is now possible to package the sensor/actuator hardware with a micro processor and an RF transceiver on a single, small, inexpensive board. Sophisticated logic can be programmed on the sensor/actuator unit. Low end sensor/actuator units are battery powered, allowing them to operate in environments in which cabling would be impossible or prohibitively expensive. As “sensor/actuator unit” is rather verbose we shall hereafter simply call such units *sensors*. The MicaZ from Crossbow is a typical example of a current generation device featuring an 8 MHz processor with 128 kbytes of Program Flash Memory, at least

4 kbytes of SRAM and 512 kbytes of flash memory for measurements.

Even after significant advancements in the design of these devices, battery life is still the main concern when considering the utility of sensor networks. The predominant power draw in a sensor node is the radio transceiver even when it is idle [1]. In consequence, the lifetime of the network can be significantly extended by reducing the number of transmissions the sensors are required to perform. One possible technique to achieve that is cross-layer optimization, where the standard layered-architecture is subverted such that information at higher layers can be used to optimize the behavior of lower layers. We consider cross-layer optimization later in Section II.

The second important issue that conditioned our design is the inherent unreliability of the wireless medium (and the devices themselves). Later in Section II-B we present an experiment showing how the medium affects packet transmission. The publish/subscribe (pub/sub) mechanism enables a loose form of communication suitable for data-centric networks [2], which is a good match for sensor networks where devices may not be reachable all the time. In a pub/sub system, participating entities *announce* an interest in receiving messages on a given topic (i.e., they subscribe to a topic) and their intention to transmit on a given topic (i.e., they publish on a topic).

Sensor applications can be broadly characterized as either actuating or monitoring. In actuation systems commands are sent from control applications in the fixed network (i.e., outside the sensor network) to change the state of the devices. Actuating devices can be implemented as a set of subscribers on topics to which the applications on the fixed network are publishers. Conversely, in sensing applications data is gathered from the sensors and transmitted to the fixed network where it can be stored and analyzed. Sensors within a monitoring system can be implemented as a set of publishers on topics to which the applications on the fixed network are subscribers. Considering this, we propose a pub/sub protocol for sensors in which the *announcements* produced by these devices are used within the networking layer to determine the path a message takes through the sensor network. As we show below in Section III, using the knowledge available at the application layer (i.e., the announcements), we create an overlay network to drive the messages to their destinations, attempting to minimize the total number of transmissions.

Although we have introduced the basics for a full

This paper is based on “Messo & Preso: Practical Sensor-Network Messaging Protocols,” by Sean Rooney and Luis Garcés-Erice, which appeared in the Proceedings of the 4th European Conference on Universal Multiservice Networks (ECUMN), Toulouse, France, February 2007. © 2007 IEEE.

monitoring/actuation mechanism on a sensor network using pub/sub (as we previously described in [3]), in this paper we focus on the actuation part of the scheme. We consider an actuation protocol where, as it is often the case, it is important for the sender to know whether a message is delivered, and the target for the actuation is not the whole sensor network, but a well-defined subset of the deployed sensors. For example, labels with electronically erasable displays and equipped with radio transceivers enable the dynamic updating of prices within a supermarket. Logically, not all labels are interested in all updates and the sender has a clear interest in knowing whether all price updates were successful.

The paper is organized as follows: In Section II we present the two aspects of wireless sensor networks that most importantly influenced the design of the protocol subsequently described in Section III. Then in Section IV we offer a study on the performance of our actuation protocol, comparing it to a simpler solution implemented using a dissemination protocol. We review comparable approaches in the literature in Section V, before we conclude in Section VI.

II. ISSUES ADDRESSED IN THE PROTOCOL DESIGN

As already stated, there are two main issues affecting our design decisions: 1) reducing energy consumption by attempting to minimize the number of packets exchanged, and 2) gaining robustness against variations in the transmission medium, that may change the reachability of devices. In order to assess the impact of the first, we briefly describe the characteristics of different approaches to reliably disseminate data in a (sensor) network. We choose the dissemination technique with the lowest total packet count. We also present experimental evidence of the low quality of the transmission path under realistic conditions: this further justifies our decision for a soft-state approach.

A. Minimizing Radio Usage

Most current sensor networking stacks organize the devices in a tree, such that the root node is a bridge to the fixed network. This is the case in the most widely accepted standards such as the network libraries of TinyOS [4] and ZigBee [5]. Levis *et al.* [6] in their overview of network abstractions for TinyOS classify tree based routing as a general abstraction which any OS for sensors should support directly. Fortunately, much work has been done in the area of optimal data dissemination over tree structures (spanning tree protocols [7], Multicast [8], etc.). However, we need to consider the option that best suits our scenario: messages are sent from the fixed network down the tree and delivered *reliably* to a well-known set of subscribers which make up some *sub-set* of the nodes in the sensor network.

Because of their simplicity, two main approaches have been used for reliable distribution in sensor networks:

Flooding: A message is flooded to all nodes and all those that are subscribed acknowledge. If not all

acknowledgments have been received within some time the process is repeated. *Addressing:* Each message is sent to individual addresses and then acknowledged. When an acknowledgment is not received from a given address, the message is resent for that address. This assumes some routing infrastructure to reach an address.

Those two techniques may be appropriate when a message has to reach all nodes in a sensor network, but they are wasteful if we are targeting a sub-set of the nodes. A third possibility is what we have named *Directed Broadcast*, which is a variation of reliable Multicast modified for the characteristics of wireless sensor networks:

Directed Broadcast: Each parent node sends a message of a given type (topic) to its immediate children using broadcast, only if at least one child (or descendant) has shown an *interest* in that type of message. A parent node sends an acknowledgment back when all interested children have acknowledged a received message. The main difference with reliable Multicast in a wired network is that in Directed Broadcast the parent can send a single message that reaches all its immediate children due to the wireless broadcast medium. In Directed Broadcast, the children need to announce their interest to their parent. Ideally, a parent interested in one topic would have all its children interested in the same topic, such that no node has to forward messages for which it has no interest. Thus Directed Broadcast requires that effectively an overlay is built over the nodes interested in receiving the same type of messages: the overlay is implemented by the relation between parent and children nodes interested in the same topic. Thus cross-layer optimization is needed to have sensors choose as parent another sensor interested in the same topic, as we see below.

The density of subscribers plays an important role in the suitability for a given scenario of Directed Broadcast, Flooding and Addressing. In order to compare these, we perform a simple analysis of the number of packets exchanged by each of these approaches in order to get a message reliably delivered to a given fraction of the sensors (and the deliver acknowledged back to the sender, too). We assume that the probability of a node being subscribed to a topic is independent of all other nodes' subscriptions and we denote it p . As we just want to provide an idea of why we choose Directed Broadcast over other approaches, we further assume a simple tree topology of depth N , such that every non-leaf node has R children.

For the reader interested in the mathematical analysis, we refer to [3] and the extensive literature on algorithm performance on trees. Here we present in Figure 1 the number of packets exchanged by the three approaches (Flooding, Addressing and Directed Broadcast) to reliably deliver one message to a fraction p of the devices.

Specifically, Figure 1 shows the total number of radio transmissions that are used by the three protocols for a tree having depth $N = 5$ and number of children $R = 2$. The packet count of Flooding and Addressing both increase linearly with subscription probability while

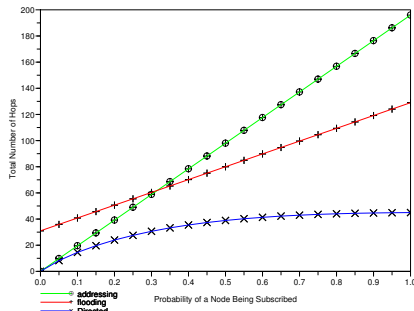


Fig. 1. The number of packets exchanged on the same network by flooding, addressing and directed broadcast.

directed broadcast increases less than linearly. This is mainly due to the fact that Directed Broadcast aggregates acknowledgment packets by having each parent wait for the acknowledgment of the children before acknowledging itself. Addressing is better than Flooding when targeting a small part of the sensor network, but as p increases Flooding becomes a better strategy. Recalling that our goal is to actuate on fractions of the sensor network, and save energy in the process, the choice of directed broadcast is clearly justified. We revisit Figure 1 in Section IV, where we compare these curves with simulation results. Even if this model is very simplified, we see how results fit very well with the expected behavior described here.

The use of Directed Broadcast brings a clear advantage, however there is also a drawback: the need to maintain the overlay over the sensors. This is not needed by the Flooding and Addressing methods, which do not require a distributed logic infrastructure to be maintained (at least at the “application” level, because Addressing needs a routing infrastructure to be maintained at the network layer). Thus, the implementation of Directed Broadcast is somewhat more complex than the other two, and it may be less robust against topology changes. We discuss how we try to overcome this last issue in the following.

B. Dealing with Unreliable Network Behavior

The previous discussion ignored the dynamic aspects of the sensor network. In particular, the topology of the network may change over time. First we need to understand how the networking layer constructs the tree in order to understand how and why it changes. We use the TinyOS standard network libraries to motivate this discussion but the ZigBee standard does something similar. At the network layer, all sensors periodically broadcast a routing message containing the set of neighbors that they have heard, a measure of the quality of the connectivity with each of those neighbors and the current depth in the tree of the sensor. Each sensor chooses as parent the neighbor with the lowest depth (except newcomers, which have initially a depth of -1) in the tree with acceptable connectivity. Connectivity is measured using the number of routing messages the sensor has heard from the neighbor over some time window. As in each routing

message the neighbor provides a measure of goodness of the connectivity with the sensor, the device can make an estimate of the goodness of the bidirectional connectivity with each of its neighbors. Sensors apply this parent-selection algorithm after the reception of each routing message.

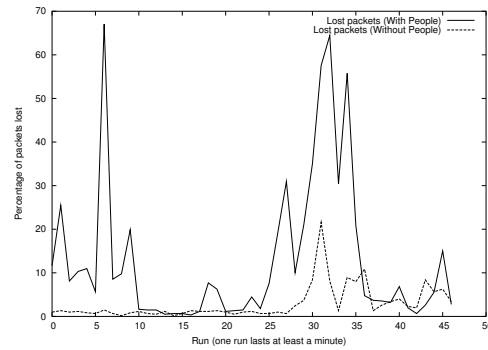


Fig. 2. Example effect of the environment (people) on packet loss.

Network topologies may change because the nodes change location or because the characteristics of the link between the nodes change. Our concern is the latter, which is more difficult to predict because it depends on the environment in which the nodes execute rather than on the nodes themselves. Zhao *et al.* [9] explore the effect of sensor placement on loss in different contexts. They conclude that there is a wide gray area of distances between good connectivity and no connectivity at which it is difficult to predict the loss rate. They believe it is due to multi-path effect meaning that small changes in the environment can have large consequences for connectivity. As an experiment we measured the loss rate across 20 meters between two MicaZ motes in our canteen when it was empty and at peak time. Figure 2 shows the results. The sender and receiver were both at one meter height. Each measurement consisted in determining how many packets had to be sent to receive 600 packets. As the sensing rate was 10 packets per second, if there were no loss then each run would last 1 minute. As many measurements were performed as would fit into one hour. The influence of the environment (in this particular case, people) on loss rate is dramatic.

The messaging layer must thus assume that the network layer will reconfigure itself. This leads us to adopt a soft-state approach: optimizing across layers, the interest announcements are periodically refreshed on intervals in-phase with the rate at which the networking layer reconfigures itself. The flexibility of the soft-state approach allows us to take advantage of the energy-saving properties of Directed Broadcast, by permitting the construction of the necessary infrastructure, while providing the means to make it robust against topology changes. We develop these points in the description of the protocol below.

III. A PUB/SUB PROTOCOL FOR SENSOR NETWORKS

We have seen how energy can be saved by using the right distribution protocol, and how cross-layer optimiza-

tions could help implementing it. We have also motivated the need for the messaging layer to use a soft-state approach to take into account frequent topology changes. Now we describe a protocol, called *Preso*, for having publications delivered to sensor networks, allowing actuation on a given subset of the devices. The *Preso* subscriber protocol is implemented in NesC and runs on TinyOS on the MicaZ sensor. In order to integrate the sensor network in a whole enterprise pub/sub infrastructure, adapters to a messaging broker running the SCADA MQTT [10] protocol have been written in the Java language. The broker itself runs on a dedicated machine connected across a serial line to the sensor at the root of the sensor network's routing tree. This setup allows messages to be published on the broker by a conventional MQTT client application on the fixed network and have those messages propagated to the appropriate subscribed sensors. Thus we are able to control the actuation on the sensors through an external application running on a PC, for example. In this paper we concentrate on the sensor network part of the infrastructure.

A. *Preso* Protocol Description

Preso is a protocol designed for distributing a message on a given topic to all devices subscribed to that topic such that the sender knows whether the message was delivered to the recipients. *Preso* assumes that the underlying network layers organize the nodes of the sensor network into a routing tree with the broker running at the root of that tree. As we have seen in Section II-A, this is a reasonable assumption. In the following, when we need to discuss implementation details, we use our *Preso* reference implementation on TinyOS .

1) *Topic overlay*: Over the basic routing tree provided by the TinyOS networking layer, *Preso* nodes form a per-topic overlay tree such that each node in a given overlay is subscribed to the corresponding topic. An overlay is formed by children nodes announcing to their parent on the routing tree the topic they are interested in, say t (see Figure 3). If the parent itself is interested in the same topic t , a link is created between the parent and each interested child on the overlay for topic t . If the parent is not interested, then it forwards the interest announcement to its own parent, until a device interested in t is found. Note that the link on the overlay is purely logical and may span multiple sensor network hops through a number of devices. However, energy is wasted if a device has to forward messages it has no interest in. Thus, whenever possible, a child should select as parent a device interested in the same topic. Because the announcements are sent periodically, the overlay is able to adapt to the changes in the underlying routing tree by creating links between children and parent nodes depending on the available connectivity.

In order to announce to its parent the topics it is interested in, a sensor sends a *Request-to-subscribe* packet addressed to the broker (located at the root of the tree) containing a summary of all the topics the device wants

to subscribe to. This summary contains the identity of the topic, the address of the device and the *Preso* sequence number of the last message it has received on that topic. A node learns which topics the nodes in the sub-tree below it are subscribed to. This is done by receiving (intercepting) the *Request-to-subscribe* packets on their way to the root. If the node is itself subscribed to any of the topics in the intercepted *Request-to-subscribe* packet, the node adds the child to its list of children for that topic (creating a link on that topic's overlay) and removes the topic from the *Request-to-subscribe* before forwarding it on. If there are no more topics in the *Request-to-subscribe* the packet is dropped.

Task `ProcessPacket (ActiveMessage m)`

Require:

```

    m = incoming packet
    children = set of children for this node
    t = topic this node subscribes to
    topicsDescendants = topics of interest to descendants
1: if m is Request-to-Subscribe then
2:   topicDescendants.add(m.getTopics())
3:   if m.contains(t) then
4:     if !children.contains(m.sender) then
5:       /* new child, create overlay association */
6:       children.add(m.sender)
7:     end if
8:     m.removeTopic(t)
9:     if m.hasTopics() then
10:      /* topics other than t in Request-to-Subscribe */
11:      forwardToParent(m)
12:    end if
13:  end if
14: else
15:  if m is ActuationCommand then
16:    /* see details in Figure 8 */
17:    if m.contains(t) then
18:      performActuation(m.command)
19:    end if
20:    if topicsDescendants.contains(m.getTopic()) then
21:      /* descendants are interested in m */
22:      broadcastToChildren(m)
23:    end if
24:  end if
25: end if

```

Fig. 3. Handling incoming *Request-to-Subscribe* and *Message* (containing an *Actuation Command*) packets.

The overlay is congruent with an invariant routing tree: if a node n is a direct child of another node n' in the overlay then n is also a descendant of n' in the routing tree and there is no node n'' subscribed to that topic such that n'' is both a descendant of n' and an ancestor of n . Informally, this is proved by noting that, given how the *Request-to-subscribe* are propagated up the routing tree, if n'' is interested in the topic and is a parent of n in the routing tree, n'' must intercept the *Request-to-subscribe* from n before it reaches n' . As the overlay is formed using a packet exchange that is distinct from that used in the formation of the routing tree (described in Section II-B) there may be times when the above no longer holds but it will converge to this state, if the routing tree itself converges to a DAG. We enhance the congruence of the routing tree with the overlay by updating the overlay tree immediately after each routing-tree reconfiguration.

Nodes that participate in an overlay for topic t forward a message of type t down the routing tree if they have children in the overlay. Using TinyOS, a message is sent down the routing tree by broadcasting, reaching all its immediate children with a single message sent. A node that is not in the overlay for t forwards all messages in t down the tree. A node acknowledges the reception of a message when all its overlay children have acknowledged. If an acknowledgment from one or more children is not received within some time, then the parent broadcasts the message again (for its children to receive it). Loss between nodes in wireless networks is likely to be strongly correlated, i.e. it is likely that if one child did not receive a message then other children did not receive it either: this increases the usefulness of the rebroadcast message, given that all the children receive it.

Preso is a stop-and-go protocol in that at most one message can be waiting acknowledgment at any moment. That is, a second message is not injected through the root node into the sensor network until the first has been acknowledged by the children of the root node. Allowing only one message to be in-flight per topic reduces the total amount of per topic state, which is necessary if we are to allow multiple topics on the sensor network and we have the memory restrictions described above for these devices.

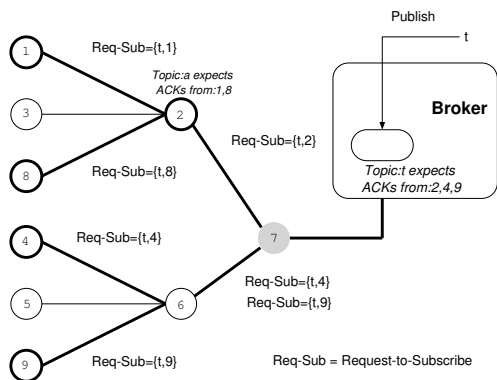


Fig. 4. The per-topic overlay in Preso.

Figure 4 shows a sensor network with a well-formed routing tree running Preso, where nodes 1, 2, 4, 8 and 9 are all subscribed to topic t . These nodes form the overlay for topic t (denoted in Figure 4 by the thicker links). The root node, 7, sends the initial broadcast message, which is repeated by nodes 2 and 6, but not by any other node as they have no descendant subscribed to topic t . Node 2 expects to receive acknowledgments from 1 and 8, while 7 expects to receive acknowledgments from 2, 4 and 9.

2) *Reliable delivery*: Each Preso publish message has a sequence number and nodes report the highest sequence number they have received on a topic in their Request-to-subscribe packets. Each parent node in the overlay puts in its Request-to-subscribe packets the minimum of all the Preso sequences numbers reported from its children as well as its own. If all Request-to-subscribe packets for a topic received at the root contain a Preso sequence

number that corresponds to the last publication message sent on that topic, then all *currently connected* nodes are up to date. It is also possible to know how many nodes actually receive the message: each node includes in the acknowledgment it sends the number of descendants that acknowledged the message. This number is summed at each node recursively up the tree so that at the root, the total number of nodes that acknowledged is obtained.

If a node n loses connectivity with its parent before the parent can deliver a message to n , and n later rejoins the routing tree at another location, then the root receiving the out-of-date sequence number in the Request-to-subscribe packet knows that a node to which a message was not delivered (in this case, n) has rejoined. The root then resends the message with the appropriate Preso sequence number, which will follow the direct path through the corresponding topic overlay to the recently reconnected node, following the path of nodes announcing out-of-date sequence numbers (which is the inverse of the path that the Request-to-subscribe packets followed from n to the root).

A parent will not resend a message if all children that have not acknowledged have reported sequence numbers in their Request-to-subscribe messages equal to or greater than the sequence number of the message (meaning that the acknowledgment must have been sent, but was lost). If after N attempts to send a message a parent node still has not received an acknowledgment from a child then it sends a *Publish-failed* packet to its own parent indicating the addresses of the children that have not acknowledged. The reception of a Publish-failed packet at the broker (via the root node) indicates that a given message was not received by all recipients. The message is retained for possible retransmission later. Messages that are resent from the broker because of topology changes are recognized by other nodes as such because the Preso sequence number is smaller than the highest sequence number they have acknowledged. Messages in such cases are not acknowledged.

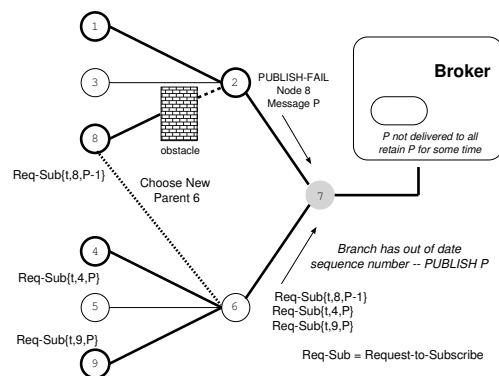


Fig. 5. Topology change with incomplete delivery in Preso.

Figure 5 shows a Preso sensor network deployment and an overlay formed for a topic t . A node in the overlay, 8, has lost connectivity with its parent 2, before this last could deliver to 8 a message with Preso sequence number

P . Node 2 tells the root that the transmission could not be performed. Node 8 subsequently attaches itself to node 6. The last message on the corresponding topic that node 8 received is $P - 1$, which is the Preso sequence number announced in the next Request-to-subscribe packet. The packet is carried to the root, which reacts by sending the message with sequence number P . Node 6 does not participate in the overlay for topic t so the message is merely forwarded on. Node 2 will see that all its children have reported Preso sequence numbers in their periodic Request-to-subscribe at least as big as P , so there is no need to rebroadcast message P . Nodes 2, 4, 8 and 9 receive the message but only 8 does not ignore it.

In summary, Preso uses two separate independent means of determining whether a message has been delivered. The first is to require children to acknowledge messages and have parents resend a message if the acknowledgment does not arrive within a given time. This increases the possibility of a message being received over a lossy link, but does not allow for the case that parent and child simply can no longer communicate, e.g. a large obstacle has been placed between them. For this case there is a second mechanism to verify delivery: the parent sends a negative acknowledgment (a Publish-failed packet) back to the root allowing the broker to retain the message knowing that some nodes failed to receive it. The routing tree invariant ensures that the child connects to the network at another location if such a possibility exists. The last received Preso sequence number included in the periodically sent packets forming the overlay (Request-to-subscribe packets) allows the identification of a path to the out-of-date node at its new location.

B. Preso implementation

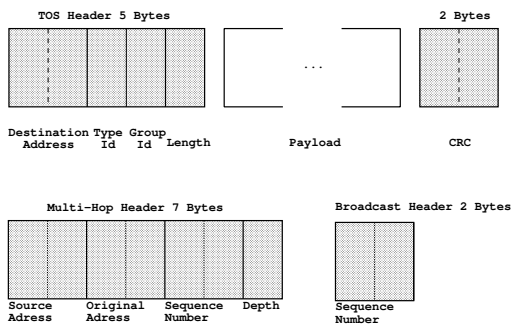


Fig. 6. TOS Active Message format.

TinyOS offers an *active message*¹ abstraction [11]. An active message is one whose handler is identified in the message itself. Active messages provide a stateless form of communication adapted to a network whose topology may be constantly changing, and whose communication

¹This use of “message” has nothing to do with the message in a pub/sub system. Throughout the paper, we clearly distinguish between a message (a publication) and a packet (for the internal protocol semantics, e.g. an acknowledgment). In this section we are however forced to follow the TinyOS convention.

needs are simple. The format of the TinyOS active message is shown in Figure 6. The header is five bytes long, the destination address is encoded in two — each sensor is assumed to have a unique address. The single byte *Type ID* allows the purpose of the message to be identified, and the single-byte *Group ID* allows multiple different applications to share the same radio frequency.

TinyOS messages that are carried up the routing tree have an additional seven-byte header. This contains the source address, the originating address, a two-byte sequence number, and the depth in the tree of the originating node encoded in a single byte. The originating address is the address of the sensor that originally sent the message, whereas the source address is the address of the sensor that forwarded the message to the receiver. Messages from the root are broadcast down the routing tree. The first time a sensor hears a broadcast message, the sensor repeats it, effectively propagating the message further away from the root. A broadcast message contains a sequence number that allows the device to determine whether it has seen the message before.

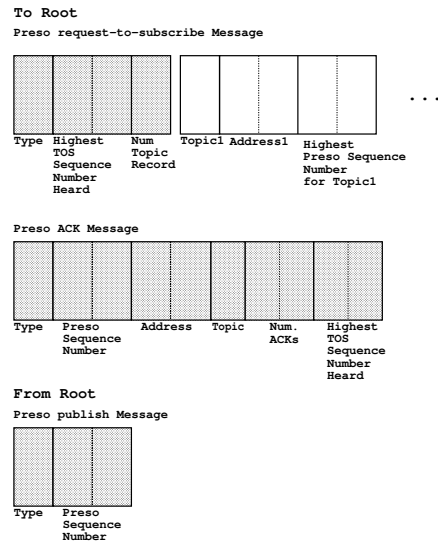


Fig. 7. Summary of the Preso messages.

Preso uses the *TOS multihop message* for carrying packets up the tree and the *TOS broadcast message* for carrying them down. The exact structure of the Preso Request-to-subscribe and acknowledgment packets are shown in Figure 7.

Figure 8 summarizes the action at a node when a publish message is received. All messages going down the tree have the same TOS type. Normally, in TinyOS messages are rebroadcast at the network layer without the intervention of the application, but an application can request that messages be redirected to it instead. Each broadcast message is then treated by Preso following the flow diagram displayed in Figure 8. Note that the actual situation is more complex because, following the soft-state approach, all retained state is removed if a timer expires before it is refreshed, so for example the state about a child from whom the parent is awaiting an

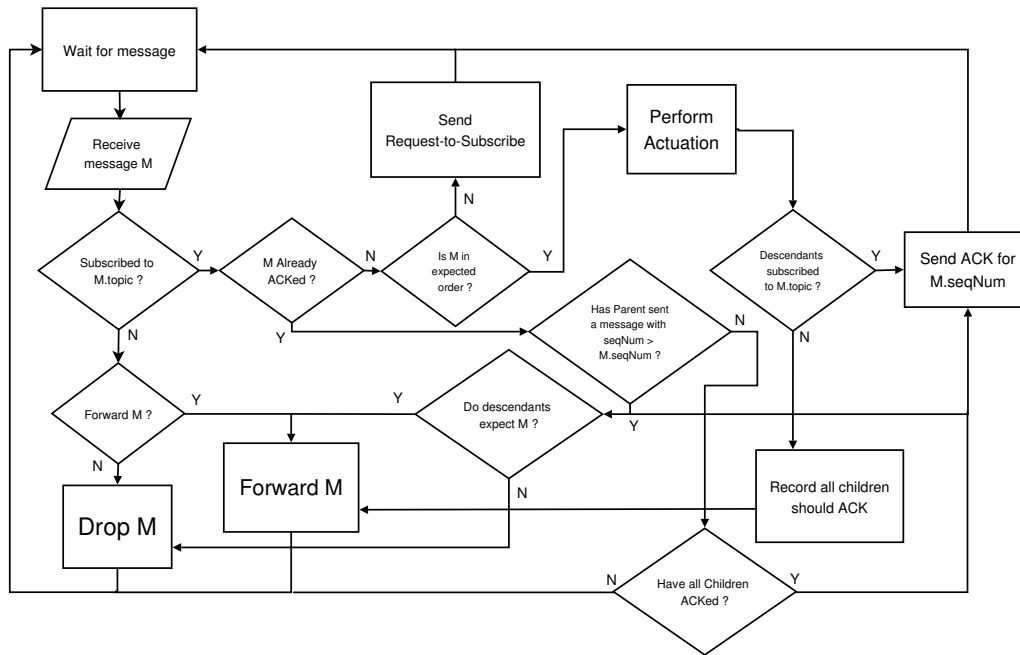


Fig. 8. Reception of a publish message in Preso.

acknowledgment may be removed before the acknowledgment is received. The fact that the state has expired means that connectivity has been lost with the child. Thus, the diagram summarizes the processing of a received publication message in a stable network.

For each topic a Preso node is subscribed to, the node keeps: 1) the set of children on the overlay for that topic and 2) for each child in that set the highest Preso sequence number that it has received. A parent node needs thus to keep for each child the address (2 bytes) and the received Preso sequence number (2 bytes), totaling 4 bytes. As an extreme example, assume a sensor network uses 30 topics and each node may have 30 children for each topic. In the worst case, the parent node may need to use up to $30 \times 30 \times 4 = 3,600$ bytes, close to the total RAM (4 kbytes) available on the devices we use (RAM size increases in newer generations of the devices). In consequence, in realistic deployments we have to trade number of children against numbers of topics to arrive at an acceptable amount of state.

IV. EXPERIMENTAL RESULTS

The protocol performance comparison in Section II-A does not take into account many factors which influence performance in an actual deployment: channel and medium characteristics, radio propagation, obstacles, hidden senders, etc. In order to determine whether the actual behavior of Preso corresponds to that of Directed Broadcast, we measure the efficiency of Preso using the TinyOS TOSSIM [12] simulator and compare it with the *Drip* protocol that comes with the TinyOS release. *Drip* is an epidemic dissemination protocol based on the Trickle [13] algorithm. Trickle uses a form of “polite gossip” whereby metadata including a version number is assigned to data types. A node will only broadcast its metadata if it has not

heard another node transmit the same thing over a time window. A node that hears another one transmitting out-of-date metadata will broadcast the newer version leading to the first node to request the transmission of the newer data from the second. In this way every connected node in the system will eventually get the most recent version of the data. *Drip* is a more intelligent flooding protocol than that described in Section II-A.

We deploy the sensors in a rectangular grid of equally spaced nodes. The sensors are organized in rows of 5. We use such a configuration because we believe it better reflects real potential deployments on, for example, warehouses, bridges, transport vehicles or structure perimeters, than a random location would do. Also, extending the grid with the increasing number of sensors by adding rows as described ensures that deeper trees are created when we use larger sensor populations. In addition, a predefined regular structure makes it easier to analyze what is going on in the network. Admittedly, some situations cannot be easily (or at all) fitted to our setup, but we think it is a good compromise between generality and ease of analysis. Each sensor in the simulator is at a distance of 5 units from the adjacent sensors.

We study the effect of different ranges for the radio on the sensors by using the *disc* model available in TOSSIM for controlling radio propagation. In this model a node can send and receive messages to all nodes which are less than a distance M units from that node. By changing the value of M we can vary the number of sensors that are in range of each other. The disc model does not introduce any progressive degradation with distance, it is just a cut-off model. We are interested in radio models only inasmuch as they allow us to force the sensors to organize themselves into different configurations.

In all scenarios we consider that the information to be

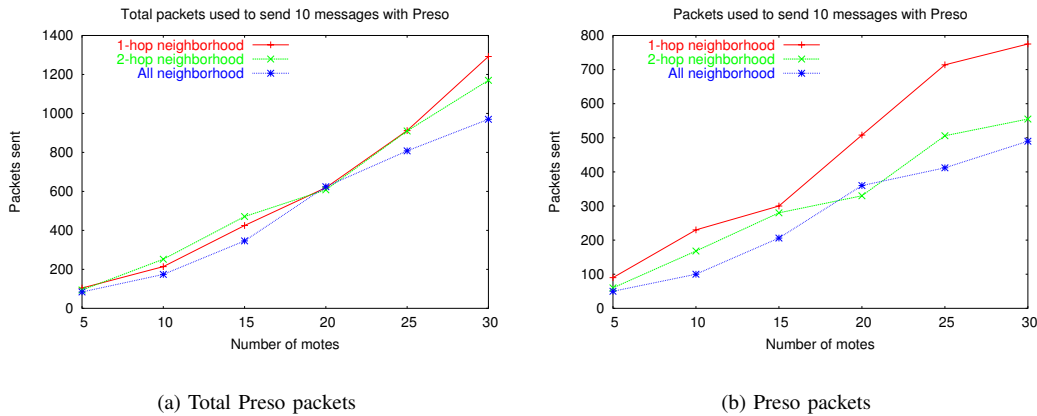


Fig. 9. Packets used to send 10 messages to all sensors in a Preso network.

sent to the sensors, whatever its purpose, is organized in topics. This is just a way to define the interest of the sensors on a given piece of information. A sensor performs the actuation only when it receives data on the appropriate topic. For Preso, this is achieved by sensors subscribing to the corresponding topic, and the command being published on said topic. Using Drip, the discrimination of the messages has to be done at the application level, because there is no mechanism that allows information to be separated. One approximation would be to use the ID on the Drip messages to identify a topic, but a Drip node only propagates messages if they are of the ID the sensor is interested in. Thus a message for a given ID can only reach a sensor interested in it if there is a connected path of sensors to the root (base station or point of entrance to the sensor network), where all sensors in the path are also interested in the same ID. Clearly this is not true in general, and when it holds, it is a degenerate case where the result is *de facto* separate sensor networks, each of them propagating messages of a different ID.

A. Comparison Drip/Preso with Different Disc Models

We compare the number of packets needed to deliver 10 message on a given topic to a sensor network where all sensors are interested in the topic. We study the effect of increasing the number of sensors and having different radio ranges as determined by the disc model. The network is grown from 5 to 30 sensors following the rules stated above, and the radio ranges correspond to “1-hop neighborhood” (each sensor can send and receive packets from its immediate neighbors on the grid), “2-hop neighborhood” (a sensor can communicate with its immediate neighbors and its neighbors’ neighbors) and “all in range” (all sensors can send and receive to all others). The results comparing Preso and Drip can be seen in Figures 9 and 10.

Comparing Figures 9(a) and 10, it can be observed that when all sensors are within range of each other, Drip uses far fewer packets than Preso. This is because while both protocols take advantage of broadcast,

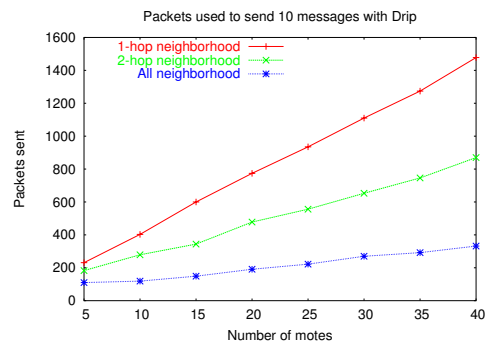


Fig. 10. Packets used to send 10 messages to all sensors in a Drip network.

where one message can reach directly all sensors with a broadcast packet, Preso sends back acknowledgments using multihop routing. Moreover, to reduce the per-node state the multihop routing protocol limits the number of children that one sensor may have in the routing tree, thus creating unnecessary packet transmissions (every sensor could acknowledge a packet directly to the root of the tree). In Preso, the root is sure that a message has been delivered. On the other hand, when the sensors are not all in range, Drip messages have to go through more hops in order to reach their destinations (see Figure 10). The number of Preso packets needed remains stable for the three different radio ranges plotted in Figure 9(a).

Figure 9(a) includes all the packets used by the whole Preso stack, i.e. including those used to form the routing tree. The frequency, with which the routing tree is refreshed depends on the expected rate at which link quality varies, it is hence configurable. Figure 9(b) show the number of packets used by Preso excluding those of the routing tree formation. As can be observed, Preso uses fewer packets than Drip for the cases where the sensors are not all in range of each other. These are also the situations where the use of Preso makes more sense, due to the nature of the underlying routing algorithm. Appropriately tuning the network maintenance protocols would add in most deployments a small amount of packets to those used by the actuation itself.

B. Comparison Drip/Preso with Heterogeneous Interests

We measure the packets needed to make a message arrive at a given proportion of interested sensors. We simulate a network of 50 sensors where each sensor can communicate only with its immediate neighbors (1-hop neighborhood). In Figure 11 we compare Preso with Drip. The proportion p of sensors is the probability of a mote being interested in the topic. For a network of size N , and a probability p of a sensor being interested in the topic of the message being sent, we choose randomly $N \times p$ sensors before each experiment. Those $N \times p$ motes subscribe to the topic when using Preso. We present the 95% confidence intervals for 5 experiments on each data point in the plot.

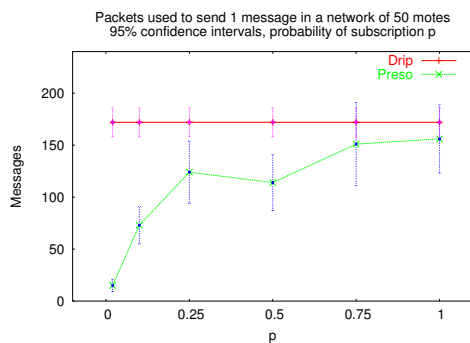


Fig. 11. Number of packets needed for a message to arrive to a given portion of the sensors in the network.

In Figure 11 can be seen how Drip and Preso behave when a given portion of the sensors is targeted by a message. As Drip can make no distinction on which sensors are interested in a topic, the amount of packets needed to distribute a message is the same regardless of the number of sensors expecting the message. The message is distributed to all sensors in any case. Preso needs an approximately equivalent number of packets to distribute one message to all sensors. The messages used in tree formation are not included in the figure for Preso as they are independent of p and can be configured to any arbitrary value depending on how frequently topology changes are to be expected. Including them would cause the graph for Preso to shift upward by a constant.

When most sensors are interested in a message ($p=0.75$), the number of packets required to distribute the message is reduced, but not significantly. This is due to the fact that, even if some motes show no interest in the message, statistically there is a high probability that they are on the forwarding path to a sensor that is interested. The non-interested sensor would have to forward the message to its descendants in the routing tree, and the acknowledgments from these toward its parent. Thus it makes little difference if one is interested in a message or not. The reduction in packets comes from the cases where the non-interested sensors are on the outside boundaries of the network, thus avoiding the routing paths from the interested sensors to the root.

On the other side of the plot we see the case where only

one sensor is interested on the topic (for our case with $N=50$ sensors, that is $p=0.02$). Preso builds a dedicated path from the root to the sensor, and the number of packets needed for the message to reach its destination is directly related to the depth of the routing tree. Only the minimum necessary sensors have to actively forward packets, while a non-discriminating protocol uses an order of magnitude more packets, wasting energy in the process. The same can be said for $p=0.1$.

Interestingly, it can be seen that the average number of packets needed to send a message increases for $p=0.25$, when compared with $p=0.50$. However, the tendency of the curve shows that Preso uses fewer and fewer packets to send a message the fewer sensors that are interested in said message. We explain this anomaly by the fact that having fewer sensors on a topic makes the overlay for that topic more “brittle”. Every hop over a single overlay-link is forwarded by more than one sensor, and each time the packet is forwarded the probability of a collision (packet loss) increases. When an overlay link maps directly to two neighboring motes, a packet loss needs only retransmit one packet. However, when the packet is forwarded over multiple non-interested sensors a lost packet has to be resent by the last sensor belonging to the overlay. The packet can be potentially (depending on where in the network it is lost) forwarded by all the sensors in between the sensor resending the packet and its children in the overlay. The same argument can be applied when an acknowledgment is lost, triggering a packet to be resent.

In summary, Preso does best when it delivers a message to a selected number of sensors interested in the corresponding topic. The benefit of using Preso increases as the interests of nodes for topics in the sensor network become more heterogeneous. The experimental results give a good fit of the behavior predicted for the directed broadcast in Figure 1.

C. Comparison Drip/Preso in a Severely Lossy Medium

We compare the performance of Drip and Preso when the medium introduces packet losses. The comparison is performed by simulation of a network of 25 sensors, following the same spatial distribution presented before. Sensors can only communicate with their immediate neighbors. Losses are introduced by setting a non-zero bit error probability on the links.

A bit error probability model (flipping each bit on a packet with a given probability) is not the best fit for simulating a wireless medium. In this kind of scenario, normally a burst of bits is altered (due to fading, interference, etc.), which would completely destroy a single packet instead of changing individual bits across a number of packets². However, for a given bit error probability e and a fixed packet size, a derived packet

²In TOSSIM, there is no way to set a packet loss probability when using the disc radio model. There are mechanisms allowing this when using the “empirical” model, but then we lose control on how sensors communicate with each other.

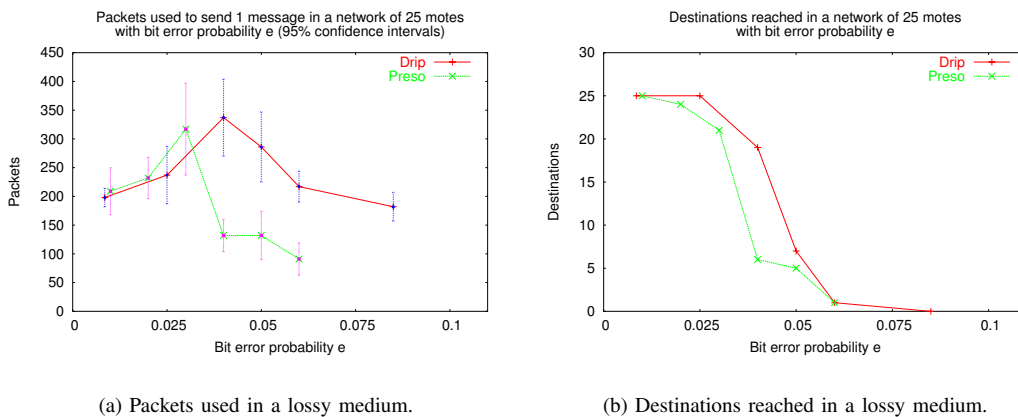


Fig. 12. Comparative performance of Drip and Preso for different bit error probability.

loss rate can be obtained. Drip and Preso messages have in general different sizes, thus the same bit error rate translates into different packet loss probabilities. In order to partially overcome this issue, we set Drip packets to be the same size as the packets used by Preso to distribute the actuation data. This is achieved by adding two bytes to the default payload of a Drip message. We justify the choice of the reference packet size by considering that 1) the data distribution packets are the most important to get the actuation done, and 2) those packets are the part of the Preso protocol that most resembles the mechanism used by Drip (broadcast packets), so the effect of bit error probability is similar for both protocols. In the case of Preso, we let the simulation run, and only when the overlay is formed we introduce error in the links. The Preso subscription announcements are issued every 20 seconds, and a parent disconnects a child after a minute without renewing the subscription. A data packet is resent after 15 seconds if an acknowledgment is not received. For both protocols, we let them run until the number of sensors performing the actuation does not increase for two minutes.

In Figure 12(a) can be observed the number of packets used by Drip and Preso depending on the bit error probability modeling the medium. In this case (as opposed to the previous section) we are counting the subscription announcement packets used to maintain the overlay during the experiment. We do this because in the presence of abundant errors the number of overlay maintenance packets is not something that can be tuned down to negligible values. In Figure 12(b) the actual number of sensors reached by the actuation packets is plotted against the same bit error probability. Thus for a given bit error probability e , one can compare the number of packets used for an effective number of sensors actually performing the actuation. The packets used are around 13 bytes long, which for a bit error probability of 0.01 gives already a 65% packet loss. Thus we are testing the protocols under very severe conditions.

We can observe 3 regions in Figure 12(a). In the first one, for the lower bit error probability the number of

packets used by both protocols increases with the rising error probability, but both remain very similar. A second region occurs for higher bit error probabilities where both protocols present a peak number of packets sent. The peak occurs earlier for Preso ($e \approx 0.03$) than for Drip ($e \approx 0.04$). As can be seen in Figure 12(b), the peak happens in both cases around the point where the number of reached destinations starts to abruptly decrease (although it has already slightly decreased before). This peak is the point at which it is still possible for the respective protocol to deliver a message to all nodes within the network. In the third region after the peak not all nodes receive the message within the time bound and the total number of messages sent decreases.

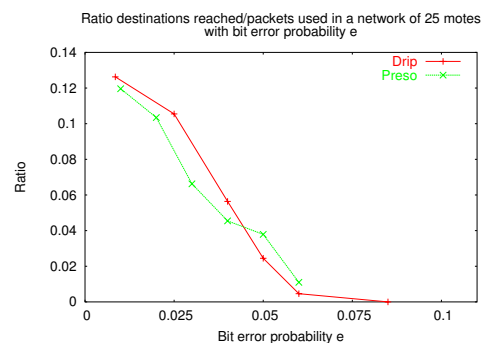


Fig. 13. Ratio of the number of sensors reached by the message to the number of packets used in the process.

Although the number of actuators reached are effectively the same for both protocols for $e \geq 0.05$, Preso wastes less energy by sending fewer packets. In Preso the number of packets is reduced because the overlay cannot be properly maintained anymore: due to packet losses, some sensors are not able to refresh the subscriptions at their parents, and the parents do not send them the message being propagated. Recall that the subscription announcement messages are negatively affected by the error model used in TOSSIM, which only allows for bit error probability, so they suffer a higher packet loss. If the packet loss were uniform for all packets, the overlay

would stay in place longer and allow for more destinations to be reached. The overlay breakdown also explains the low number of sensors reached by the message: children get disconnected from their parents and only a few sensors around the root of the tree are able to get the message. Thus fewer sensors are reached, but fewer packets are sent, too. In Figure 13 we show the ratio of the number of sensors reached by a message to the packets used in the process. Ideally, one would have a maximum of sensors reached with a minimum number of packets used. Preso remains very close to Drip. It can be observed that when Preso reaches a low number of destinations, it is because few packets are sent (otherwise the ratio would tend much faster to 0, as Drip does), due to the overlay breakdown. In our experiments the loss rate on all links is identical and invariant. It is more likely that losses on links is highly variable (see Figure 2), both Preso and Drip guarantee that a message will be delivered eventually to all node that are connected to the network.

In summary, Preso in the presence of errors uses a similar number of packets as a much simpler approach (except for pathological losses), provides feedback to the external application, and sends fewer packets when the losses are such that destinations cannot be reached anyway, saving energy in the process.

V. RELATED WORK

In this section we describe some alternative approaches to implementing monitoring and actuating applications using sensor-networks.

TinyDB [14] describes a means by which data can be extracted from a sensor network in a data-centric way. *TinyDB* treats the entire sensor network as a database that can be queried using SQL-type requests. In *TinyDB* the sensors organize themselves into a tree such that requests are received from the parent and propagated to the children, whereas replies are returned back up the tree. Nodes within the tree combine data received both from their children and from themselves to reduce the total amount of data that needs to be transmitted. The root of the tree is the connection to the fixed network from which requests originate.

The idea of interacting with the sensor network as a whole rather than with the individual devices simplifies the view of the network to the external devices shielding them from low-level information such as activation schedules, device identity, topology information, etc. In this sense it resembles the work described here in that pub/sub abstraction hides the details from the fixed network. Considering a sensor network as a database is appropriate if there is a rich sensor ecology supporting sensors of multiple different types and if the data that is of interest is not known before the deployment of the network. However, *TinyDB* queries are stateful in order to apply grouping functions from results from children, for example the average value from a set. This means that changes in the routing tree require the entire distributed application to reorganize itself. As the results of a query

are necessarily sent to one location, there is no simple way of distributing the results to multiple base stations. In large commercial applications, we may wish to connect a device network to the enterprise network at several places to provide fault tolerance and to reduce the overall number of hops between a device and the closest base station. This is possible in a protocol like Preso as described in [3].

TinyDB does not match well with reliable actuation: for example, a supermarket in which electronic labels are updated dynamically across the device network. Although a *SELECT* statement can be associated with an action, when performing a control function we are interested in knowing if it could *not* be performed on a given device, for example, because we have lost connectivity. In such cases the identity of the device is important, which is in contradiction with the data-centric approach.

Directed diffusion [15] is a form of pub/sub in which a spanning-network is formed from publishers to subscribers. The spanning-network is obtained by flooding the subscription request through the network, assigning weights to each network edge as a measure of how appropriate it is as a return path from publisher to subscriber. The means by which these weights are assigned is decided by each node locally. Data is then propagated from publishers to subscribers using one or more of the possible paths; each node decides locally along which paths it should forward the message. Data messages are carried using unicast. A node will not repeat a message it has already forwarded and will not repeat a message on a topic for which it has not seen a subscription message. In directed-diffusion a node only knows about its immediate neighbors making end-to-end reliable delivery, as required by Preso, difficult to implement.

We described the *Drip* and *Trickle* algorithms in Section IV. *Trickle* was originally designed for updating code within a sensor network. It is appropriate for the dissemination of data where the timeliness of the arrival is not important. While some attempts have been made to add naming within *Drip* such that only certain nodes within the network receive a new message [16], *Drip* is inappropriate for reliable actuation as delivery to unconnected sensor nodes can fail silently. As the percentage of nodes interested in a topic decreases the probability of a given node having an immediate neighbor interested in the same topic also decreases, meaning that simple epidemics protocols become less effective.

Some work has proposed *reliable transport protocols* over the sensor network, either using new protocols, for example RMST [17], PSFQ [18] or extensions to TCP [19]. These protocols follow the same model: intermediate nodes between source and destination cache resent data segments and then perform the retransmission if a negative acknowledgment or in the case of TCP a selective acknowledgment is received. The aim of avoiding end-to-end retransmissions over a lossy multi-hop network is similar to that of Preso. In Preso reliability is performed at the messaging rather than the transport layer allowing a hierarchical approach in which nodes delegate

to their descendants the responsibility for the reliable delivery of messages.

VI. CONCLUSION

We have argued that most practical sensor network application can be classified as either monitoring, in which information is drawn from the wireless sensor network to the fixed network, or actuating, in which control messages are sent from the fixed network across the sensor network. Current research has concentrated more on monitoring rather than actuating applications. Actuating applications have different, more challenging requirements: in particular, actuation requires reliable delivery and reports of failure when this cannot be achieved.

We have proposed and justified two main features that our actuation protocol should have: 1) save energy, by reducing the number of packets used to transmit messages reliably (using Directed Broadcast and cross-layer optimizations), and 2) have the protocol distributed structure be resilient to changes in the topology of the routing tree due to the changing connectivity between wireless nodes (using a soft-state approach). We have described a protocol that follows those guidelines, allowing for reliable but efficient actuation across a sensor network. This involves using a per-topic overlay across the routing tree to limit the amount of nodes which have to forward messages. We achieve reliable delivery over a network whose topology is changing by being able to identify which subtrees contain nodes that have not received all messages, and having messages that are resent directed automatically to these subtrees (avoiding those not interested in receiving the resent message). We have extensively described the protocol fundamentals and measured its performance, comparing our proposal to an epidemic protocol. We conclude that while epidemic protocols are more resilient than ours, our overlay approach has a similar behavior for a moderate error probability, and its features make it more suitable when all nodes are not interested in the same topics and reliable delivery is required. We have quantified how reliability is traded-off against increased cost of delivery.

REFERENCES

- [1] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of IEEE INFOCOM*, vol. 3, June 2002, pp. 1567–1576.
- [2] J. M. Hellerstein, W. Hong, and S. R. Madden, "The sensor spectrum: technology, trends, and requirements," *SIGMOD Records*, vol. 32, no. 4, pp. 22–27, 2003.
- [3] S. Rooney and L. Garcés-Erice, "Messo and Preso, practical sensor-network messaging protocols," in *Proceedings of ECUMN'2007*, Toulouse, France, February 2007, pp. 364–376.
- [4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.
- [5] ZigBee Alliance, "ZigBee Document 02130r10, Network Specification Revision 1.0, Version 1.00," ZigBee Alliance, 2400 Camino Ramon, Suite 375, San Ramon CA 94583, USA, Tech. Rep., 2004.
- [6] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in TinyOS," in *First Symposium on networked system design and implementation (NSDI04)*, San Francisco, California, USA, March 2004, pp. 1–14.
- [7] C. Gui and P. Mohapatra, "Efficient overlay multicast for mobile ad-hoc networks," in *Proceedings of IEEE WCNC'03*, vol. 2, New Orleans, LA, March 2003, pp. 1118–1123.
- [8] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *ACM Computer Communications Review*, vol. 25, no. 4, pp. 342–356, Aug. 1995.
- [9] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2003, pp. 1–13.
- [10] IBM, "Telemetry integration." [Online]. Available: <http://www-306.ibm.com/software/integration/mqfamily/integrator/telemetry/>
- [11] B. Phillip, H. Jason, and C. David, "Active message communication for tiny networked sensors," 2001 [Online]. Available: <http://www.tinyos.net/papers/ammote.pdf>
- [12] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*. New York, NY, USA: ACM Press, 2003, pp. 126–137.
- [13] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *In First Symposium on Network Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, March 2004, pp. 15–28.
- [14] S. Madden, "The design and evaluation of a query processing architecture for sensor networks," Ph.D. dissertation, University of California, Berkeley, 2003.
- [15] C. Intanagowiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of Mobile Computing and Networking*, Boston, MA, USA, August 2000, pp. 56–67.
- [16] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Proceedings of EWSN 2005*, Istanbul, Turkey, 2005.
- [17] F. Stann and J. Heidemann, "RMST: Reliable data transport in sensor networks," in *Proceedings of the First International Workshop on Sensor Network Protocols and Applications*, Anchorage, Alaska, USA, April 2003, pp. 102–112.
- [18] C. Wan, A. Campbell, and L. Krishnamurthy, "PSFQ: A reliable transport protocol for wireless sensor networks," in *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, New York, NY, USA, 2002, pp. 1–11.
- [19] T. Braun, T. Voigt, and A. Dunkels, "Energy-efficient TCP operation in wireless sensor networks," *PIK Journal Special Issue on Sensor Networks*, vol. 28, no. 2, 2005.

Sean Rooney was awarded a PhD from the University of Cambridge in 1998 for work on the open control of ATM networks. Since then he has been a Research Staff Member at IBM's Zurich Research Laboratory. His current research interests include self-controlling distributed systems, sensor networks and software models for exploiting multiprocessor architectures.

Luis Garcés-Erice obtained his MSc in Telecommunication Engineering (EE+CS) from the Public University of Navarra (Spain) in 2001. He obtained his PhD in 2004 from Télécom Paris University (France) working at Institut EURECOM. Currently he is at IBM Research Zürich (Switzerland) as a Post-Doc. His research interests are mainly in P2P and distributed systems, network protocols and more recently middleware and sensor networks. Luis Garcés-Erice is a professional member of the IEEE Communications society, Computer society and the ACM.