Benefits of Inter-Tree Optimizations for Content based Publish-Subscribe in Sensor Networks

Praveen Jayachandran, Raghu K. Ganti, Indranil Gupta, and Tarek F. Abdelzaher
Department of Computer Science
University of Illinois, Urbana-Champaign
email: {pjayach2, rganti2, indy, zaher} @uiuc.edu

Abstract

Sensor networks pose the challenge of distribution of content generated within the network to the origins of requests for this content in an efficient and timely manner. In this paper, we formulate this multiple-source multiple-sink data dissemination problem as a content-based publish-subscribe problem. While previous research concentrated on optimizing each flow independently for energy consumption, we propose inter-flow optimizations for network-wide energy savings. We propose a methodology to construct multiple multicast trees, one for each publisher, so as to increase the extent of aggregation across multicast trees at intermediate nodes. We describe how inter-tree optimization, through aggregation of multiple publisher-subscriber flows at intermediate nodes, can be performed. Further, in the presence of application specified delay constraints, we extend our scheme to maximize aggregation while ensuring that the delay constraints are met.

Index Terms

Sensor networks, Publish-Subscribe, Data dissemination, Energy optimization, Aggregation, Multicast

I. Introduction

Wireless sensor networks have found wide use in a variety of applications, such as tracking enemy objects in battle-fields, monitoring animal habitats, and industrial applications [1]. A variety of sensor nodes exist, most prominent of which are the Berkeley motes [2], which have limited processing, sensing, energy, and communication capabilities.

There has been substantial work in data aggregation and query processing in sensor networks, such as Directed Diffusion [3], TAG [4], and TinyDB [5]. These approaches can be classified as "data-centric routing", where routing is not between end-hosts with unique identifiers, but for named data. In these approaches, the query is flooded to sensor nodes that potentially have relevant data, and these data are then aggregated in-network, and sent to the source of the request. If the rate of queries is high, then the expense of flooding the query can be substantial.

To avoid flooding of queries, solutions such as DIFS [6] and GHT [7] adopt the "data-centric storage" approach. Events are named and stored in specific nodes in the network, based on the name and geographical location of the

event. Nodes have unique identifiers through which they can be addressed and queried. Storing events by name provides a logical mechanism by which a query can be directed only to the nodes that hold the required data and need not be flooded. The above approaches concentrate on efficiently answering a single query through in-network processing.

Supporting publish-subscribe in sensor networks provides a natural solution to the query processing and data dissemination problem, as well as propagating code updates. A publish-subscribe mechanism supports multiple simultaneous flows from publishers to subscribers, allowing for efficient inter-flow optimization. This inter-flow optimization provides a globally efficient solution, rather than addressing queries independently. The data published and consumed could be either raw sensor data, or named data as in DIFS or GHT. There are two kinds of publish-subscribe systems:

- *Group-based:* Events are classified into one of a fixed set of groups. Subscribers subscribe to all the events within a group.
- *Content-based:* Data pertaining to an event are classified into a number of *information spaces*, each associated with an *event schema* that defines the type of information contained in each event. An event is represented as a set of *(attribute, value)* pairs.

In this paper, we address the problem of multiple-source multiple-sink data dissemination in wireless sensor networks, by formulating it as a content-based publish subscribe problem, with multiple publishers and subscribers. Each subscription is for specific content rather than for events within a group.

Our contributions in this paper are two-fold. First, we propose a methodology for *inter-tree* energy optimizations of content-based multicast trees in sensor networks. While previous research concentrated on aggregation within a single flow (typically a tree), our solution is novel in that we address inter-tree optimizations, where aggregation is performed across multiple publisher-subscriber flows. We efficiently construct multiple multicast trees, one for each publisher, so as to increase the extent of in-network aggregation. We discuss how intermediate nodes perform inter-tree aggregation leading to considerable energy savings. Second, we address the problem of real-time publish-subscribe, where we optimize the basic solutions to the publish-subscribe problem based on delay constraints (in terms of hop count) specified by the application.

To motivate the relevance of a generalized content-based publish-subscribe scheme, we provide a few example applications. Consider an application scenario, where a set of heterogeneous sensor nodes (assume that each node is equipped with one or more of temperature, humidity, and pressure sensors) are deployed to monitor a large ecological environment, for example the Great Duck island deployment [8]. Various users can present range queries to this network to obtain different results. For example, a query could ask for temperature readings within a particular range, and pressure readings within another range. Typically, there are multiple sensor nodes generating data, and

multiple users consuming this data, which is essentially a content-based publish-subscribe system with multiple publishers and subscribers.

Another interesting, but futuristic application scenario is in a forest or sanctuary, where events are to be recorded. For example, consider the event of a predator attacking its prey, which needs to be captured on a camera. Currently, this is done manually, through long hours of waiting for the event to happen. Imagine an array of sensors being deployed in the forest, with the capability of detecting such an event, and a set of mobile cameras placed at strategic locations. In this scenario, the publishers are the sensors which generate events and the subscribers are the cameras (actuators which are waiting for the event to happen). When the event does occur, the data needs to be transmitted from the sensors to the cameras within certain delay constraints, so that the camera that is nearest to the event can move to the location of the event and capture it. This introduces a new dimension to our problem: performing real-time publish subscribe in sensor networks.

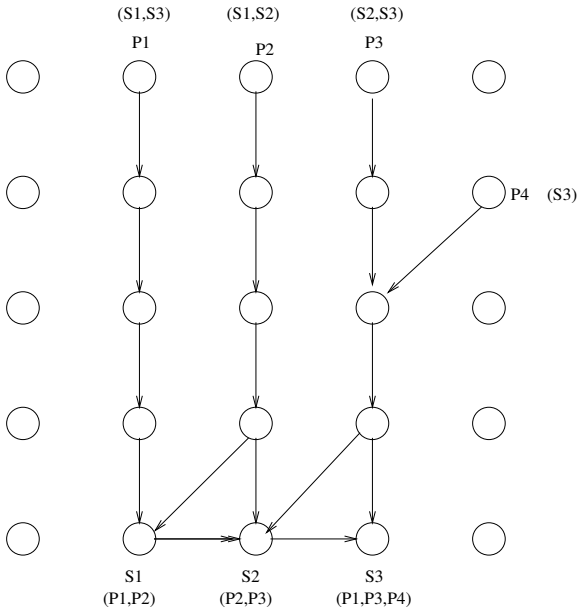

Fig. 1. Example illustrating publish-subscribe without attempting to aggregate across flows

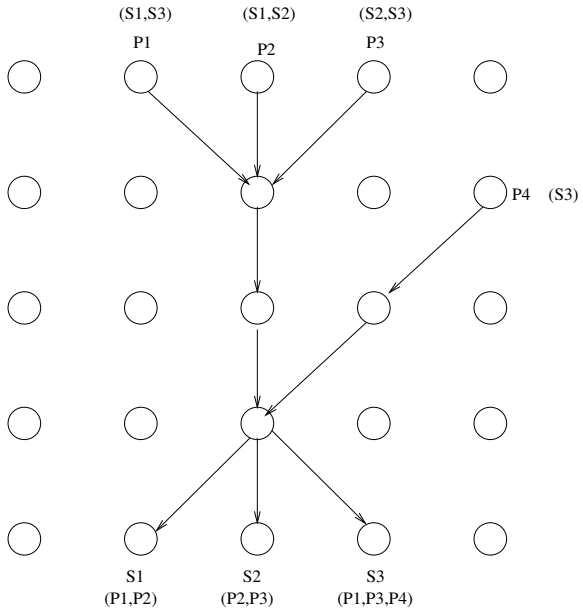

Fig. 2. Example illustrating publish-subscribe, intentionally constructing multicast trees so as to increase aggregation across publishers

To illustrate, Figures 1 and 2 show a sensor network with publishers generating data and subscribers consuming them. $P1, P2, P3$, and $P4$ are publishers of content, and $S1, S2$, and $S3$ are subscribers. The clauses shown below each subscriber refer to the publishers queried by this subscriber, and correspondingly the clauses shown above the publishers refer to the subscribers that query this publisher. Figure 1 shows a scenario where publishers construct multicast trees to subscribers who have requested the data, independently of other publishers. Contrast this with the scenario shown in Figure 2, where each publisher intelligently constructs its multicast tree so as to maximize the aggregation across the multicast trees constructed by different publishers. The second scenario involves only

8 broadcasts, while the first scenario requires 15 broadcasts, assuming that the flows are synchronized in time to allow aggregation. This aggregation across several multicast trees results in significant energy savings, and is the primary motivation of our work.

The rest of the paper is organized into six sections. Section II discusses work related to publish-subscribe schemes in both distributed systems and sensor networks. The basic publish-subscribe problem and its extension to incorporate hop count constraints are defined and solutions to these problems are presented in Section III. We present a simple mathematical analysis in Section IV. We implement the algorithms proposed on a TinyOS simulator and discuss the implementation details in Section V. We do extensive evaluation of these algorithms in Section VI, and conclude in Section VII.

II. Related Work

We divide the related work section into two parts. The first summarizes previous work on publish-subscribe mechanisms in traditional distributed systems. The second part provides a brief survey of publish-subscribe systems in sensor networks, and presents some related sensor network research.

A. Publish-Subscribe in Distributed Systems

In [9], [10], to efficiently implement content-based publish-subscribe systems, two key problems are identified:

- Efficiently *match* events to subscriptions.
- Efficiently *multicast* events within a network of brokers (intermediate nodes of multicast trees).

Matching can be solved by a naive algorithm that tests all subscriptions against each event. However, this algorithm does not scale with increasing number of publishers and subscribers. The *Gryphon* project [11] uses a tree-matching algorithm to solve the matching problem in sub-linear time on an average [9]. The main idea of this algorithm is to organize the subscriptions into a Parallel Search Tree (PST) data structure, where each node corresponds to a test and each subscription is a path from the root to a leaf. In this paper, we do not concern ourselves with the matching problem and only concentrate on the data dissemination problem.

A link matching algorithm is used to solve the multicasting problem in the Gryphon project [10]. The PST is augmented with vectors of *trits*, where the value of each trit is either “Yes”, “No”, or “Maybe”. A trit corresponds to whether a subscriber can be reached from a given broker using a particular link. This algorithm works in three stages, the first annotates the PST with trits, the second computes an *initialization mask* at each broker, which is used for routing. The final stage refines the initialization mask until all values in the mask are “Yes” or “No”.

Recent work [12] has looked at optimization across multiple overlays in the Internet. This inter-overlay optimization scheme, Anysee, is a peer-to-peer live streaming system that joins resources across multiple overlays, so

as to better utilize these resources. Our inter-tree energy optimization is in some sense similar to the inter-overlay optimization. However, our inter-tree optimization is in terms of aggregation, in order to reduce network-wide energy consumption.

FeedTree [13] proposes a multicast protocol for RSS news-feeds in the Internet to reduce the load on news-servers. Splitstream [14] addresses this problem by constructing a forest of interior-node-disjoint multicast trees to distribute load among several peers. An epidemic style multicast protocol is described in [15]. The above mentioned algorithms are not well suited for sensor networks due to the energy constrained nature of the sensor network.

B. Publish-Subscribe in Sensor Networks

An SQL type query-based language has been presented in [5], where users view the sensor network as a single database and query this network database using SQL type queries. Each query is considered independently and no aggregation is performed across multiple queries. In contrast, we attempt to efficiently construct multicast trees so as to maximize the amount of aggregation possible across multiple publisher-subscriber flows.

A publish-subscribe based solution for data placement and asynchronous multicasting which aims at power conservation is presented in [16]. Middleware services, which determine the data cache placement and the updation of these caches in order to minimize the communication overhead and power consumption of data transfer, have been developed. The publish-subscribe paradigm used in [16] is a group-based paradigm, as opposed to the content-based paradigm used in our work.

A publish-subscribe middleware for sensor networks, Mires, has been proposed in [17]. Mires provides a high-level publish-subscribe service to the node application. It is responsible for advertising the topics (subscriptions) provided by the local application and publishing information pertaining to the advertised topics. It assumes a single sink node and does not account for overlap of information requested by multiple sinks, and aggregation across multiple publisher-subscriber flows.

A semi-probabilistic routing scheme for publish-subscribe systems in sensor networks is described in [18]. Message subscriptions are broadcast deterministically in the immediate vicinity of the subscriber. When a message is published, it is routed using this deterministic information, if available. In the absence of known subscriptions, the message is broadcast probabilistically. However, this solution is not targeted towards a system with multiple publishers and subscribers.

A publish-subscribe routing algorithm for ad hoc sensor networks has been presented in [19]. The routing algorithm relies on a distributed broker-tree infrastructure, which supports data-aggregation, loop-free routing, and fault tolerance. To overcome the drawback of scalability, an interval-routing approach is used. However, this requires additional services such as neighbor detection, leader election, and support for fault detection.

Two implementations of the publish-subscribe paradigm have been discussed in [20]. In the first implementation, a publish-subscribe broker efficiently filters messages on behalf of the subscribers. The second implementation addresses the issue of scalability by distributing the brokers throughout the network. The design of efficient routing protocols and message dissemination schemes are of primary concern in this case. Centralized and distributed architectures for publish-subscribe systems in mobile environments have been proposed in [21], although no implementations have been developed.

In contrast to these approaches, we concentrate on aggregating data across different publish-subscribe flows to conserve energy. We propose a methodology by which multicast trees can be efficiently constructed to increase the in-network aggregation possible. We also tackle the problem of data dissemination in the presence of application specified delay-constraints.

Our solution methodology can be adopted together with solutions such as DIFS [6] and GHT [7]. Events are named and stored in specific nodes in the network, based on the name and geographical location of the event. Using such a logical mechanism for storing data, DIFS can efficiently handle range queries on high-level events by querying only the relevant rendezvous points (and not all sensor nodes publishing data). Content-based subscriptions are typically range queries, which makes the use of DIFS very appealing. Further, the spatially distributed index maintained by DIFS provides efficient load balancing in terms of average search and storage requirements. The only minor modification we need to make to DIFS is to store not only high-level events, but also low-level raw sensor data, whenever the subscription queries are for raw data.

III. Problem Definition and System Design

The following section describes the problem in the first subsection and develops solutions for this problem in the latter subsection.

A. Problem Definition

The content-based publish-subscribe problem in sensor networks can be decomposed into two sub-problems:

- Matching relevant published data to subscriptions
- Dissemination of the published content to the subscribers

The subscribers are the users of the network who request data through queries. The published data can either be raw sensor readings, or aggregated named data as in DIFS or GHT. In DIFS and GHT-based solutions, the network maintains all published data as *(name, value)* pairs. In this paper, we do not concentrate on how subscriptions are matched to relevant published data.

The dissemination of published content to the subscribers is essentially a multicast problem, with the difference that there are now multiple sources (nodes with the required published information). The dissemination problem, thus requires construction of multiple multicast trees, one for each source. The optimization criteria for the multicast tree construction could be either *power consumption*, when node lifetime is of primary concern, or *delay*, when data needs to be propagated to the subscribers within specified delay constraints. In the power constrained problem, the objective we consider, is to minimize the total bytes transmitted in the network over all publish-subscribe streams. In the delay constrained problem, we assume that the application specifies constraints on the hop-count between every *(publisher, subscriber)* pair. In the presence of these constraints, the objective is again to minimize the total bytes transmitted in the network.

B. System Design

In this section, we describe a solution to the dissemination problem described in Section III-A. We propose a methodology to construct multicast trees so as to increase the amount of aggregation possible, and describe how inter-tree optimization can be performed. We explain how the proposed solutions can be adapted to target different optimization criteria, such as power consumption and delay. We make the following assumptions about the system:

1) Topology and node positions are static.
2) Each node has a unique identifier.
3) Nodes are aware of the topology, that is, they know the location and neighbor information of all other nodes in the network.

Sensor networks are typically deployed to study the characteristics of a particular environment and are quite static. Unique identifiers for each node can be easily achieved through efficient hashing techniques. Finally, our assumption of the knowledge of each node's location and neighbor information is not unrealistic, as efficient node localization schemes exist such as [22], that can determine the location of all the nodes in the network. Once the location of all nodes in the network is known, based on (possibly approximate) knowledge of radio range, the neighbor information can be obtained for all nodes.

There are two steps to the publish-subscribe process. A subscriber first initiates a query to a set of publishers. The publishers then route content relevant to the query back to the subscribers. Note that, it is possible for new queries to arrive while the published content for other queries are being delivered. We now briefly describe these two steps:

1) *Directing queries to relevant publishers and query processing:*

   Each subscription query potentially consists of multiple clauses, each querying for different sensor data (we shall use the term query to particularly refer to subscription requests in the rest of the paper). Each of these

individual clauses is directed to the relevant publishers that correspond to the sensor data requested in the clause. Since common clauses of multiple queries from different subscribers are targeted to the same publisher, these common clauses are processed only once. This reduces the computational overhead at the publishers, especially when the number of subscriptions is large and contain similar query clauses.

2) *Dissemination of query responses to subscribers:*

The problem of dissemination of query responses to subscribers can be formulated as a multiple multicast problem. A publisher could have data relevant to multiple subscriptions and this data needs to be disseminated to the different subscribers. Also, the response to each subscription will potentially arrive from multiple publishers. Hence, flows of different source-destination pairs can be merged through inter-flow aggregation, to reduce the overall communication cost.

The basic outline of our solution is as follows. Each subscriber independently constructs a multicast tree to all publishers it needs to query, and routes the query along this multicast tree. Thus, each publisher is aware of the multicast trees used by each of its subscribers. Based on this information, each publisher assigns costs to all nodes in the network, such that a node which is an ancestor of a larger number of publishers in the subscribers' multicast trees known to it, is assigned a smaller cost (the exact numerical cost is defined in Section III-B.3). The idea is that a node that is an ancestor of more publishers in the multicast trees constructed by the subscribers has a greater likelihood of being part of multiple flows from publishers to subscribers, and thus provides a greater potential for aggregation. Based on this cost assignment, a publisher constructs a multicast tree and routes data along this multicast tree. Each publisher that is queried by at least one subscriber, constructs such a multicast tree. An intermediate node that is a part of multiple flows, can possibly aggregate these flows (inter-tree aggregation) and save considerable energy. A compact representation of the various steps involved in our solution is as follows:

Step 1 *Each subscriber independently constructs a multicast tree to all publishers it requests data from. In constructing such a multicast tree, all nodes are assigned equal costs (Section III-B.2).*

Step 2 *Each publisher, based on the requests it received from subscribers, assigns costs to all nodes. A node that is an ancestor to a greater number of publishers, in the subscribers' multicast trees is assigned a lower cost (Section III-B.3).*

Step 3 *Each publisher constructs a multicast tree to the set of subscribers that requested data from it, based on the cost assigned in Step 2 (Section III-B.4).*

Step 4 *Intermediate nodes aggregate data of multiple multicast messages, if possible (Section III-B.5).*

In the following subsections, we describe each of these steps in greater detail.

1) Multicast Tree Construction: There have been several multicast tree construction heuristics proposed for ad hoc wireless and sensor networks. We develop schemes based on two such heuristics, which we call Node Removal Heuristic (NRH) and Minimum Distance Heuristic (MDH).

The first is a simple heuristic based on pruning the minimum spanning tree to obtain a multicast tree. Due to the broadcast nature of the wireless channel, in our heuristic we consider node costs instead of link costs. We start with the set of all nodes in the network, and iteratively remove the maximum cost nodes that will still leave all the destinations reachable from the source of the multicast tree. The algorithm terminates when no more nodes can be removed without disconnecting some destination from the source. We call this heuristic the Node Removal Heuristic (NRH).

The second heuristic that we consider is based on a heuristic proposed in [23]. This algorithm is easy to implement and has been theoretically analyzed to show that it minimizes the overall power consumption in the network. As power is a critical resource for sensor networks, we use this heuristic to construct multicast trees. The heuristic works as follows. The multicast tree is initialized to contain only the source. At each iteration, a path from the current tree to the 'closest' destination, is added to the tree. This continues until all destinations belong to the multicast tree. The closest destination can be identified by collapsing the tree to a single point and executing Dijkstra's algorithm for this point, based on any arbitrary cost metric. Here again we consider node costs instead of link costs. The total cost of the multicast tree is the sum of the costs of the individual nodes in the multicast tree. We call this heuristic, the Minimum Distance Heuristic (MDH).

The above mentioned heuristics are used both to construct the multicast trees from subscribers to publishers (to route the query) described in Section III-B.2, as well as to construct multicast trees from the publishers to their respective subscribers (to route the published data) described in Section III-B.4. Note that the heuristics presented here are only examples of a class of multicast tree construction algorithms that can be used with our proposed solution. Any multicast tree algorithm that uses the cost of nodes to construct the multicast tree belongs to this class.

2) Multicast Tree Construction by Subscribers: Each subscriber computes a multicast tree to all the publishers it needs to query, using any one of the heuristics mentioned in Section III-B.1. For this multicast tree construction, the cost of using a node as part of the tree is the same as that of any other node. The query is routed using this multicast tree to all the publishers.

3) Node Cost Assignment by Each Publisher: Upon reception of requests, each publisher becomes aware of the multicast trees used by its subscribers. Each publisher defines a value $publisher\_count_i$ for each node i, that denotes the number of publishers reachable from node i over all multicast trees known to this publisher. We use this as a measure of the extent of aggregation possible over flows from publishers to the subscribers. The main

idea is that a node that is an ancestor to a larger number of publishers, has a greater potential to be a point of aggregation. The cost of each node i is set to $max(MAX\_PUBLISHER\_COUNT - publisher\_count_i, 0)$, where $MAX\_PUBLISHER\_COUNT$ is a predefined constant. Such a cost assignment is then used to construct a low cost multicast tree (lower cost implies a greater potential for aggregation) to route data from the publisher to its subscribers. Note that the costs are assigned independently by each publisher based on the set of multicast trees of its subscribers. Hence, different publishers could assign different cost values to the same node. In our implementation, we use a $MAX\_PUBLISHER\_COUNT$ value of 20 (the default number of publishers and subscribers are 35 and 10, respectively).

4) Multicast Tree Construction by Publishers: Based on the cost assignment policy described in Section III-B.3, each publisher constructs a multicast tree to its subscribers using any one of the heuristics outlined in Section III-B.1, and routes data to the subscribers along this tree. Such a tree has a greater likelihood of sharing paths with flows from other publishers, leading to greater energy savings.

5) Aggregation at Intermediate Nodes: An intermediate node that is a part of multicast trees of multiple flows from publishers to subscribers, can perform inter-tree aggregation whenever possible. The extent of aggregation across multiple multicast trees is dependent on two factors, which are:

1) **Delay**: Aggregation is possible if the messages from multiple sources arrive at the intermediate node at nearly the same time.
2) **Number of multicast flows that can be aggregated**: If the destinations within the subtrees rooted at the intermediate node of two or more multicast flows are the same, then aggregation of the data is possible (assuming that the data of the different flows are of the same type and can be aggregated). However, whenever two multicast flows have a common intermediate node, the node can combine the two messages into a single packet regardless of the destination set, thereby saving on the header overhead.

Our solution assumes the existence of a time synchronization scheme, several of which have been developed for sensor networks [24], [25]. Further, we assume that each node waits for a specified time (δ) before it broadcasts the given message to the next hop in the multicast tree (in addition to any MAC layer delays). This tunable parameter δ affects the extent of aggregation. Increasing δ will allow more messages to be aggregated. But, increasing δ increases the delay of the message, which affects the real-time delay constraints, if any.

Note that, we do not assume any in-network processing within a multicast flow as done in [5], which will provide further savings, as it reduces the amount of data each publisher sends to the subscribers. In this paper, we define two types of aggregation schemes across multicast flows, namely: *header aggregation* and *payload aggregation*. Traditionally, sensor networks have aggregated data from multiple senders en route to a common destination, which we term as payload aggregation. Our intelligent multicast tree construction methodology described in the previous

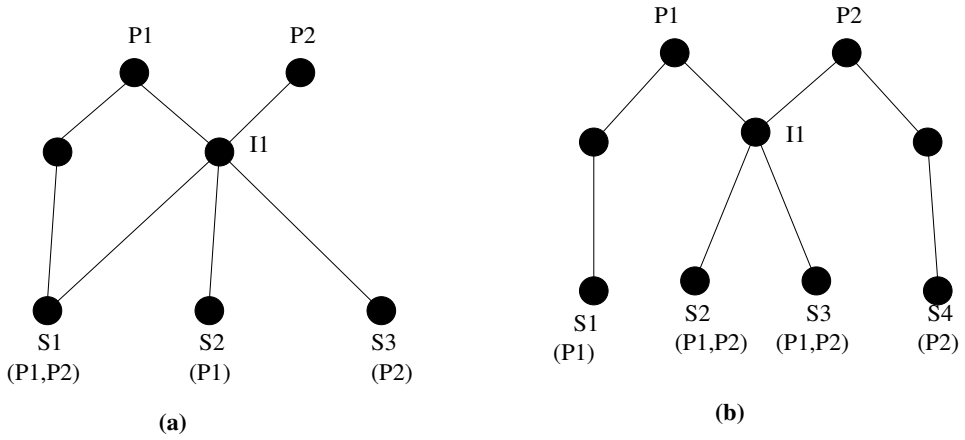

Fig. 3. Illustration for header and payload aggregation

sections, helps increase the extent of aggregation possible across multiple multicast trees.

Header aggregation is the simple aggregation of messages, where the data of two different messages are concatenated into a single message. A simple scheme such as this also results in considerable savings, because it reduces the number of header bytes transmitted and the number of messages sent. In most sensor networks, the amount of information (payload) sent in a single message is comparable to the header overhead. Such small payload sizes are also the reason behind choosing the default packet size in TinyOS [26], a widely used operating system for sensor networks, to be 29 bytes. Sensor networks are highly resource constrained, and the main consumption of energy is due to radio communication. Thus, reduction of the number of bytes and messages transmitted translates into greater energy savings. In this aggregation process, we assume that the length of the message from the source is small enough so that two or more messages can be aggregated into a single message. The maximum number of messages that can be aggregated within a single message is constrained only by hardware limitations. The details of the packet format and how header aggregation is exactly done are provided in Section V.

Payload aggregation is more aggressive in that it tries to reduce the number of bytes transmitted by aggregating the payload data of multiple multicast flows, when the data are of the same type. An intermediate node will incur savings only when the subscribers that are part of the subtree of one multicast tree rooted at this intermediate node, are the same as the subscribers that are part of the subtree of the other multicast tree rooted at the given intermediate node. Note that, we do not aggregate messages from the same publisher, but perform inter-flow aggregation.

We illustrate the header and payload aggregations in Figure 3. The publishers from which the data are required is shown below each subscriber. We observe from Figure 3(a) that the intermediate node (I_1) has to deliver data from P_1 and P_2 to S_2 and S_3, respectively, and both P_1, P_2 to S_1. As I_1 needs to transmit both P_1 and P_2 (as the destinations are different), it is useless for it to aggregate the payload. However, it can reduce the header overhead by performing header aggregation, and concatenating the data from both P_1 and P_2 and sending it in a single

message. On the other hand, in Figure 3(b), the intermediate node I_1 can perform payload aggregation and send out only one message, with reduced payload size (as opposed to two messages). The method of payload aggregation employed is beyond the scope of this paper.

6) Delay-Constrained Multicast Tree Construction: When the published content need to reach the subscribers within certain specified delay-constraints, we assume that the application provides these constraints in terms of the number of hops within which the published content must reach the subscriber from the publisher. We further assume that the delay-constraint imposed is not too stringent making it impossible to route the published content to a subscriber within the specified number of hops. In other words, the delay constraint in terms of maximum number of hops allowed, should be at least as large as the minimum distance between the publisher and the subscriber. In this case, our objective is to maximize the aggregation across multiple multicast trees, while ensuring that the number of hops between any *(publisher, subscriber)* pair is within its corresponding delay-constraint. Since we observed that MDH performed consistently better than NRH (shown in Section VI), we implement the delay-constrained scheme only using MDH, and modify it in the following manner. Instead of choosing the closest destination to the tree in each iteration, and adding the shortest path from the chosen destination to the multicast tree, we do the following. We choose the closest destination (W) to the tree, and choose a connection point (V) on the tree closest (in terms of cost as described in Section III-B.3) to the destination that ensures that the sum of the distance of the connection point from the source (U) within the tree and the distance of the destination from the connection point is less than the hop-count delay constraint for that source-destination pair. This constraint can be expressed as follows:

$$hop\_count(U,V) + hop\_count(V,W) \leq delay\_constraint(U,W)$$

As the tree is constructed, the distance of all points within the tree to the source is noted. When a destination is chosen to be added, we run Dijkstra's algorithm for the destination and calculate the distances from the destination to all tree nodes. Aided with this information, one can easily compute the closest connection point on the tree (in terms of cost) that satisfies the hop-count constraint.

IV. Mathematical Analysis of the System with Certain Simplifying Assumptions

In order to mathematically analyze the benefits of intelligent-tree construction and inter-tree aggregation, we consider a two-dimensional grid topology. We make the following assumptions about the system:

1) Each node can communicate only to its immediate up, down, left, and right neighbors.
2) The publishers are in a straight line at the left end of the grid, and the subscribers in a straight line at the right end of the grid. Both the publishers and the subscribers are centered vertically. In other words, there is at most one more publisher (or subscriber) above the central point in the grid, than below.

Grid size	K x K
No. of Publishers	P
No. of Subscribers	S
Packet header size	H bytes
Packet payload size	D bytes
Total bytes transmitted	E bytes

TABLE I
NOTATION USED IN THE ANALYSIS.

3) Each subscriber requests content from all the publishers.
4) All the published content are similar and perfect aggregation is possible. More concretely, two messages each with payload size D, can be aggregated into a single message with payload size D. The analysis for the case where the aggregation is not perfect (the payload size increases by a fraction A of the published content size D, each time a message is aggregated), although simple to derive, is more complex in terms of detail. We omit presenting the analysis for this case, in favor of simplicity.

For such a system, we analyze the energy consumption (E) in total number of bytes transmitted across all flows for the following cases:

1) No aggregation is performed.
2) Aggregation is only opportunistic. In other words, whenever two messages are to be sent from the same node, they can be aggregated. However, no special effort is undertaken to efficiently construct multicast trees to maximize aggregation. Inter-tree aggregation is performed without intelligent-tree construction.
3) Intelligently constructing multicast trees to maximize aggregation, together with inter-tree aggregation.

The notation used in the analysis is provided in Table I.

A. No Aggregation

As no aggregation is performed, each message is of size $H+D$. To calculate the total bytes transmitted, we need to calculate the total number of messages transmitted, and multiply it by $H+D$. We consider two cases, $P \geq S$ and $P < S$. Figure 4 shows the messages transmitted for these cases. The edges represent message transmissions, and the number on each edge represents the number of messages transmitted between the two nodes.

Case 1: $P \geq S$

The publishers in line with subscribers S_1 and S_S each require a total of $K+S-2$ messages (there are $K+S-1$ nodes in their multicast tree, all with degree one). The publishers in line with S_2 to S_{S-1} require a total of $K+S-3$ messages (there are $K+S-1$ nodes in the multicast tree, one node has degree two and all other nodes have degree

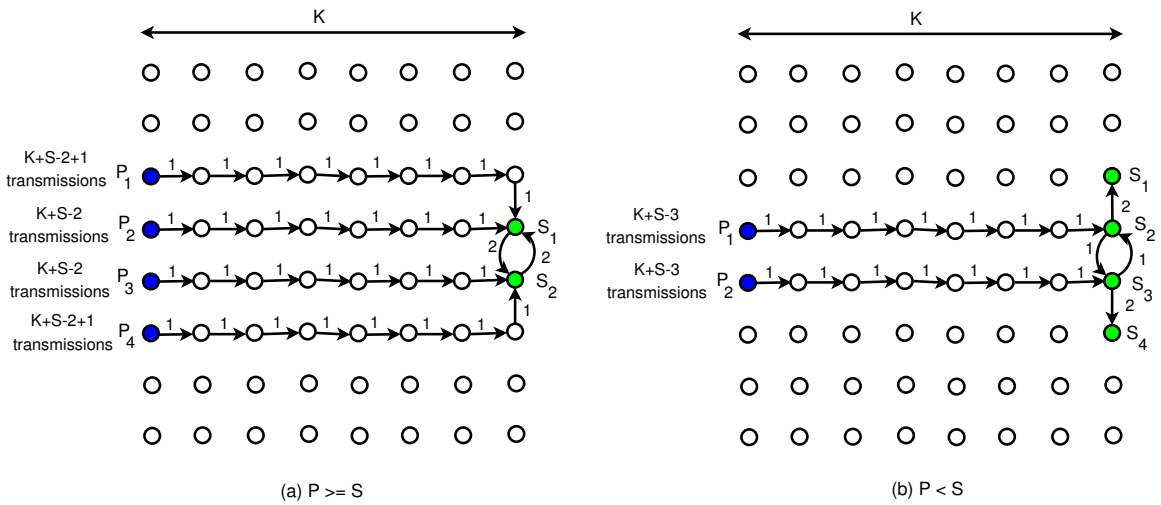

Fig. 4. Figure showing the messages transmitted for the no aggregation case (a) $P \geq S(P = 4, S = 2)$, (b) $P < S(P = 2, S = 4)$

one). We need to now account for the messages due to the $P - S$ other publishers, who are not in line with any subscriber.

Without loss of generality, we assume that there are $\lfloor \frac{P-S}{2} \rfloor$ publishers above S_1, and $\lceil \frac{P-S}{2} \rceil$ publishers below S_S. The number of messages sent by publishers above S_1 are $(K+S-2+1), (K+S-2+2), \ldots, (K+S-2+\lfloor \frac{P-S}{2} \rfloor)$. Therefore, the total number of messages sent by publishers above S_1's column is given by,

$$\left\lfloor \frac{P-S}{2} \right\rfloor (K+S-2) + \frac{\lfloor \frac{P-S}{2} \rfloor (\lfloor \frac{P-S}{2} \rfloor + 1)}{2}$$

Similarly, the total number of messages sent by publishers below S_S's column is given by,

$$\left\lceil \frac{P-S}{2} \right\rceil (K+S-2) + \frac{\lceil \frac{P-S}{2} \rceil (\lceil \frac{P-S}{2} \rceil + 1)}{2}$$

Hence, the total number of bytes transmitted in the entire network is given by,

$$\begin{aligned} E &= [2(K+S-2) + (S-2)(K+S-3) + (P-S)(K+S-2) + \frac{\lfloor \frac{P-S}{2} \rfloor (\lfloor \frac{P-S}{2} \rfloor + 1)}{2} \\ &\quad + \frac{\lceil \frac{P-S}{2} \rceil (\lceil \frac{P-S}{2} \rceil + 1)}{2}](H+D) \\ &= [(P-S+2)(K+S-2) + (S-2)(K+S-3) + \frac{\lfloor \frac{P-S}{2} \rfloor (\lfloor \frac{P-S}{2} \rfloor + 1)}{2} + \frac{\lceil \frac{P-S}{2} \rceil (\lceil \frac{P-S}{2} \rceil + 1)}{2}](H+D) \end{aligned}$$

Case 2: $P < S$

For each publisher, the multicast tree consists of $(K + S - 1)$ nodes, with one node of degree 2, and all other nodes of degree one. Therefore, the number of messages transmitted for each publisher is $(K + S - 3)$ (as shown in Figure 4(b)). Therefore,

$$E = P(K+S-3)(H+D)$$

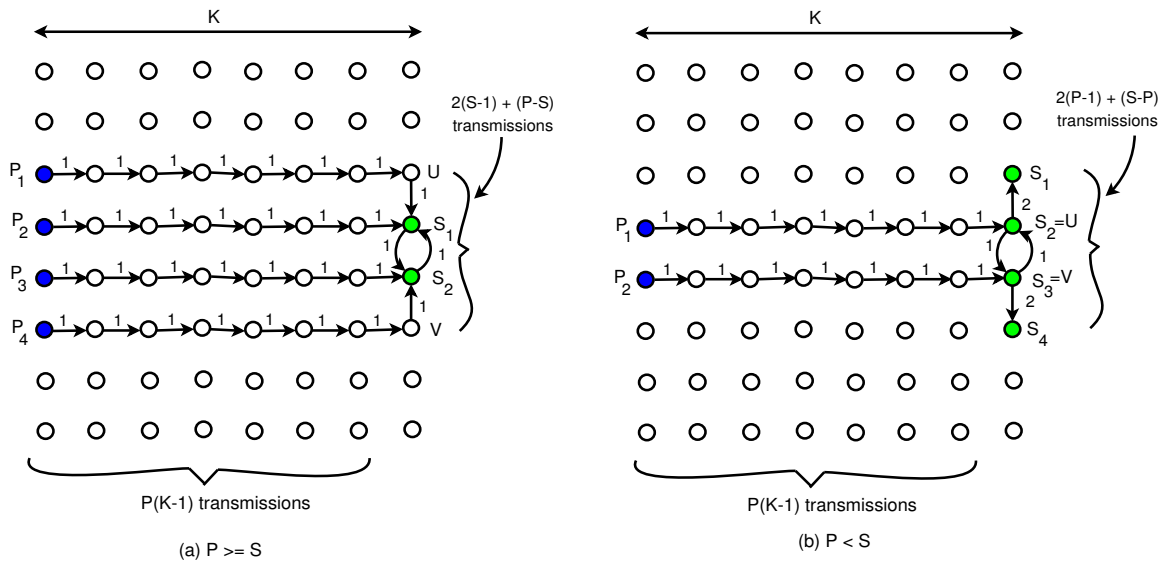

Fig. 5. Figure showing the messages transmitted when aggregation performed is opportunistic (a) $P \geq S (P = 4, S = 2)$, (b) $P < S (P = 2, S = 4)$

B. Opportunistic Aggregation

Figure 5 depicts this scenario. Each message transmitted is of size $H + D$, due to our perfect aggregation assumption, regardless of the number of publishers' content that have been aggregated within the message. In this case, each publisher has no knowledge of the other publisher-subscriber flows. We assume that our Minimum Distance Heuristic (MDH) is used to construct the multicast flows. According to this heuristic, at each iteration, the destination closest to the source is added to the tree. Notice that in this scenario, the heuristic in fact creates optimal multicast trees for each flow, taken independently. We consider two cases, $P \geq S$ and $P < S$. Figure 5 shows the multicast trees constructed and messages transmitted for these cases.

Case 1: $P \geq S$

We count the total number of messages transmitted in the following manner. Notice the grid of $P \times (K-1)$ nodes, each transmitting one message. At the end of these $P \times (K-1)$ transmissions, each publisher P_i's message has reached the node in the same row, but in the rightmost column. We now need to count the number of messages transmitted by nodes in the rightmost column, that is, the column in which the subscribers are present. Let the node in the rightmost column and in the row corresponding to P_1 be U, and the node in the rightmost column and in the row corresponding to P_P be V, as shown in Figure 5(a). It is easy to see that, the minimum number of message transmissions happen when messages start from U and propagate down towards S_S, aggregating messages due to publishers P_1, P_2, …, $P_{\lfloor \frac{P-S}{2} \rfloor + S - 1}$, along the path. Likewise, messages start from V and propagate up towards S_1, aggregating messages due to publishers P_P, P_{P-1}, …, $P_{P-(\lceil \frac{P-S}{2} \rceil + S - 2)}$, along the path. At the end of these messages, all subscribers would have received all the published content. The total number of messages transmitted by nodes in the subscriber's column is,

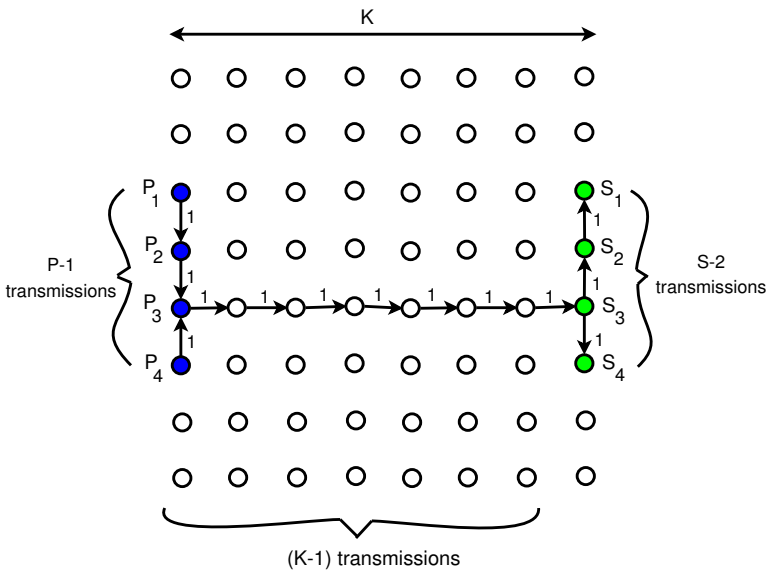

Fig. 6. Figure showing the messages transmitted when multicast trees are constructed so as to maximize aggregation (P=4,S=4).

$$\left\lfloor \frac{P-S}{2} \right\rfloor + S - 1 + \left\lceil \frac{P-S}{2} \right\rceil + S - 1 = 2(S-1) + (P-S)$$

Therefore,

$$E = (P(K-1) + 2(S-1) + P - S)(H+D)$$

Case 2: $P < S$

The number of messages transmitted can be calculated similar to the previous case. Here again, $P \times (K-1)$ transmissions enable each published content to reach the corresponding node in the rightmost column. We now need to calculate the messages transmitted by all the subscribers. Let the subscriber in the same row as P_1 be U, and the subscriber in the same row as P_P be V, as shown in Figure 5(b). The minimum number of messages transmitted would be when messages start from U and propagate down towards S_S, and similarly messages start from V and propagate up towards S_1. The number of transmissions by subscribers is given by $2(P-1)+(S-P)$, similar to case 1.

Therefore,

$$E = (P(K-1) + 2(P-1) + S - P)(H+D)$$

C. Intelligent Tree Construction

This section calculates the cost of the optimal inter-tree structure for the analyzed scenario. Our NRH and MDH heuristics are geared to achieve this optimality, but are not guaranteed to do so. Hence, this is not a performance

analysis of our algorithms themselves, rather it is an indication of the maximal benefits that can be obtained via *any* inter-tree optimization strategy. The multicast trees constructed and the aggregation performed for this case are shown in Figure 6. Comparing this to Figure 5 immediately reveals the drastic reduction in the number of transmissions required when multicast trees are intelligently constructed to facilitate inter-tree optimizations.

Due to our perfect aggregation assumption, each message transmitted is of size $H + D$ regardless of the number of publishers' content that have been aggregated within the message. Notice that a total of $P + (K - 2) + (S - 2)$ transmissions are required to deliver all the publisher's content to all the subscribers.

Therefore,

$$E = (P + K + S - 4)(H + D)$$

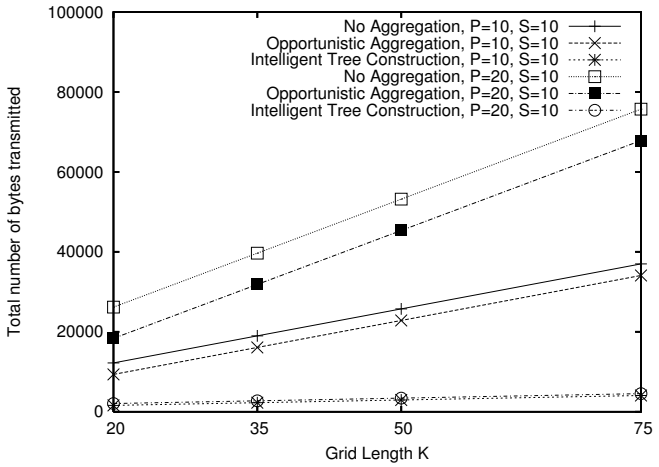

Fig. 7. Analytical comparison of the total number of bytes transmitted for the three schemes for varying grid lengths.

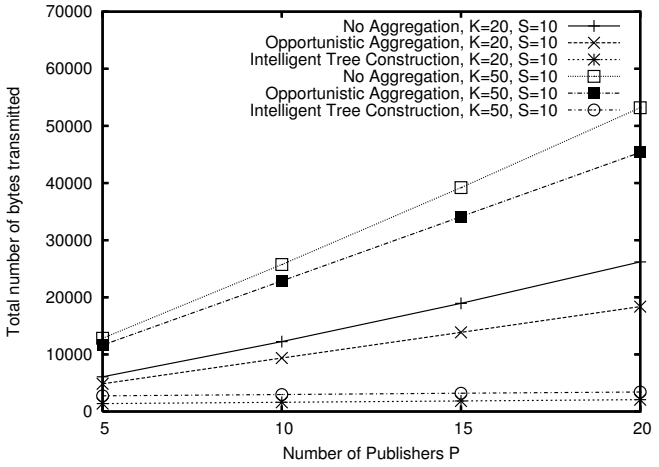

Fig. 8. Analytical comparison of the total number of bytes transmitted for the three schemes for varying number of publishers.

Using the expressions derived above for the total number of bytes transmitted, we compare the performance of the three schemes for different values of K, P, and S. Figure 7 compares the three schemes for varying grid lengths, while setting S to 10, and P to 10 and 20. The comparison for varying number of publishers is shown in Figure 8, setting S to 10, and K to 20 and 50. Figure 9 shows a similar plot for varying number of subscribers, while setting P to 10, and K to 20 and 50. All the curves are linear in K, P, and S. The curve for total bytes transmitted using the intelligent construction of multicast trees has a much lower slope than the other schemes, in all three figures. Although the scenario considered here is idealistic, this shows the potential for aggregation when construction of several publish-subscribe flows are performed together, in an intelligent manner.

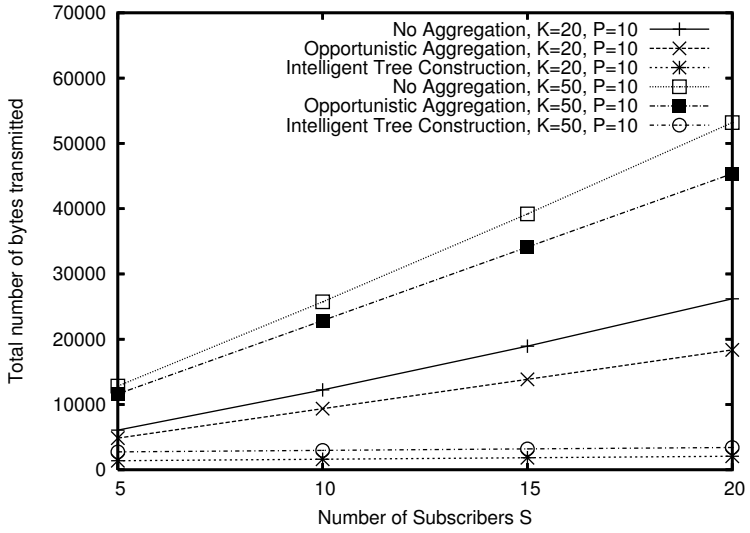

Fig. 9. Analytical comparison of the total number of bytes transmitted for the three schemes for varying number of subscribers.

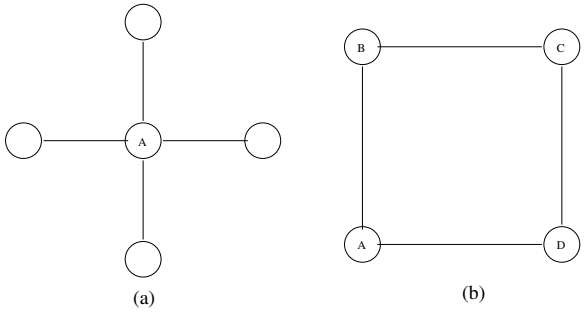

Fig. 10. A typical node and its neighbors

V. Implementation Details

As a proof of concept, we have implemented MDH on a sensor network simulator, TOSSIM [27]. TOSSIM is a discrete event simulator for TinyOS based sensor networks.

A. Communication Model

Our communication model assumed for the simulations on TOSSIM is a grid topology. A network of $n \times n$ nodes is considered, where each node can only communicate with its immediate left, right, top, and bottom neighbors (if they exist). This is an ideal communication scenario (where no interferences exist), our simulations are MAC-independent and we believe that the MAC layer problem is orthogonal to our inter-tree optimization problem. We do not study the interactions of various MAC layer protocols with our inter-tree optimization schemes, as it is beyond the scope of this paper. Figure 10(a) shows a typical node and its neighbors.

type (1 byte)	source (1 byte)	multicast id (1 byte)	sender (1 byte)	destination list (MAX_DESTS bytes)	tree node list (MAX_TREE_NODES bytes)

Fig. 11. Packet format of multicast tree message

B. Message Formats and Message Propagation

As described earlier in Section III-B, there are two stages at which multicast trees are constructed. In the first stage, the subscribers construct multicast trees, and route queries along this tree to their respective publishers (from which they request data). In the second stage, the publishers construct multicast trees to the corresponding subscribers based on the node cost assignments. Both of these multicast tree propagations follow the same format, as described below. We assume that each node has global network knowledge, that is, the knowledge of location and neighbors of all the nodes in the network. This can be achieved using a geographic hashing technique as in GHT. This knowledge is used to construct the multicast tree on the publisher's end. The multicast tree is constructed by the source of the data dissemination and is mapped on to a packet called the multicast tree message and propagated to the corresponding set of destinations. The packet format is shown in Figure 11. The packet size in TOSSIM is constrained to 128 bytes (emulating sensor hardware). This small packet size prevents us from storing explicit parent-child relationships within the multicast tree packet. The packet contains only the set of nodes within the multicast tree. For long lasting multicast flows, a globally unique multicast identifier is created by the source, and each intermediate node that belongs to the multicast tree stores the multicast identifier along with the set of destinations and tree nodes for that multicast.

The multicast tree message is composed of six fields, which are:

1) `Type`: A one byte field that indicates that the packet is a multicast tree packet
2) `Source`: A one byte field that indicates the source of the multicast tree.
3) `Multicast id`: A one byte globally unique identifier for this flow.
4) `Sender`: A one byte field that indicates the sender of the packet, which is explained in detail later.
5) `Destination list`: An array of size `MAX_DESTS`, terminated by -1, that indicates the destinations of this multicast tree.
6) `Tree node list`: An array of size `MAX_TREE_NODES`, terminated by -1, that indicates the set of all nodes in the multicast tree. `MAX_DESTS` and `MAX_TREE_NODES` are predefined constants based on the size of the network.

Any node that receives this packet will check the list of tree nodes to verify if it is a part of this list and if so, will rebroadcast the packet and store the multicast identifier, destination list and tree node list. If the current node receiving the packet has no children in the multicast tree, then it will not forward the packet any further. We explain

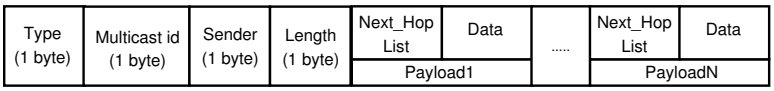

Fig. 12. Packet format of data message

how a node identifies its children in the next sub-section. The packet is not forwarded by nodes that are not part of the multicast tree.

Data packets are forwarded just like the multicast tree packet. When an intermediate node receives messages from multiple multicast trees, it can aggregate the data contained in these messages whenever possible (payload aggregation) or concatenate them into a single message when aggregation is not possible (header aggregation) as described in Section III-B. The method of aggregation employed is beyond the scope of this paper. The message format for the data packet is as follows (shown in Figure 12):

1) `Type`: A one byte field that indicates that the packet is a data packet or an aggregated data packet
2) `Multicast id`: A one byte globally unique identifier for this flow.
3) `Sender`: A one byte field that indicates the sender of the packet, this is explained in detail later.
4) `Length`: Indicates the length of the packet.
5) $Payload_1$ .. $Payload_N$: Any number of multicast messages can be composed into a single packet, packet length permitting. Each payload field includes the set of next hop nodes to which the message is intended, followed by the message itself, terminated by a delimiter. Each payload message can either be a single message, or an aggregated message.

When a node receives such a message, it extracts the set of messages that are intended for it and rebroadcasts these messages, if necessary.

C. Message Loss and Collision Avoidance

Due to the inherent lossy nature of the wireless channel and due to collisions, message losses are inevitable. We use an acknowledgment based scheme while constructing the multicast tree to recover from losses due to the wireless channel, and random back-off timers to avoid losses due to collisions. Whenever a node receives a multicast tree packet, it checks to see whether it is from a parent or its child. In this context, we assume that a node receiving a multicast tree packet for the first time (for a particular source) considers the sender of this packet as its parent and the rest of its neighbors in the multicast tree as its children. To ascertain whether a node sending a multicast tree packet is a parent or child, the `sender` field that is part of the packet is used. This field is set by each node to be its own address, when it broadcasts the multicast tree packet. If the multicast tree packet is received from the parent, an acknowledgment is sent to the parent, indicating the reception of the packet. The parent resends a

multicast tree packet if it fails to receive an acknowledgment from all its children within a certain timeout. To avoid unnecessary messages in the network, rebroadcasts of the multicast tree that occur due to loss of acknowledgment messages will not be propagated further.

Collisions occur due to the CSMA nature of the MAC layer, where hidden terminal problems are not solved. Consider a scenario as shown in Figure 10(b), where node A broadcasts a multicast packet, and nodes B and D receive it. If both of these nodes are part of the multicast tree, then they will broadcast this packet resulting in a collision at node C. As both nodes B and D sense that the channel is idle, their messages will always result in collisions at node C. To avoid these collisions, we introduce random back-off timers before a node broadcasts a multicast tree packet or an acknowledgment packet. Thus, in the scenario described above, nodes B and D will each wait for a random amount of time before broadcasting the multicast tree packet, thus resulting in very few collisions.

D. Handling Node Failures

Node failures are common in sensor networks and applications should gracefully handle node failures. After a threshold number of retries, if a parent is unable to receive an acknowledgment from a child node, it assumes that the child node has failed. Under such a scenario, the parent tries to find an alternate temporary multicast route to the subset of destinations in the child's subtree (obtained from the destination list and tree node list maintained for this multicast flow). Meanwhile, it sends this node failure information back to the source of the multicast tree, so that the source can reconstruct the multicast tree. As all nodes have location information of all other nodes in the network, the temporary multicast tree construction by the intermediate node can take place just like any other multicast tree construction.

VI. Evaluation

We evaluate the performance of our aggregation scheme through simulations. We conducted simulations on TOSSIM. We use two performance metrics for comparison: the total number of bytes transmitted by all nodes in the network and the number of intermediate nodes that perform payload aggregation. We compare the performance of our intelligent tree construction methodology and inter-flow optimizations (we call this 'intelligent tree construction') with a simple scheme that does not perform any aggregation across multicast trees. We use MDH as the default tree construction algorithm, as we show in Figures 13 and 14 that MDH performs better than NRH.

Unless otherwise specified we consider the following default setup: a grid of 50×50 nodes, each node connected to the four neighbors around it (the nodes immediately above, below, left and right). We assume that the nodes are aware of the locations of other nodes. We consider a default of 10 subscribers and 35 publishers.

In our simulations, we generate the set of publishers and their corresponding subscribers as follows. Each subscriber selects each publisher with probability 0.5, thus the average number of feeds each subscriber get is 17.5, with a Gaussian distribution around this average (when the number of publishers is 35). The error bars in each plot show the standard deviation over ten runs (some points are slightly perturbed, so as to make the error bars discernible).

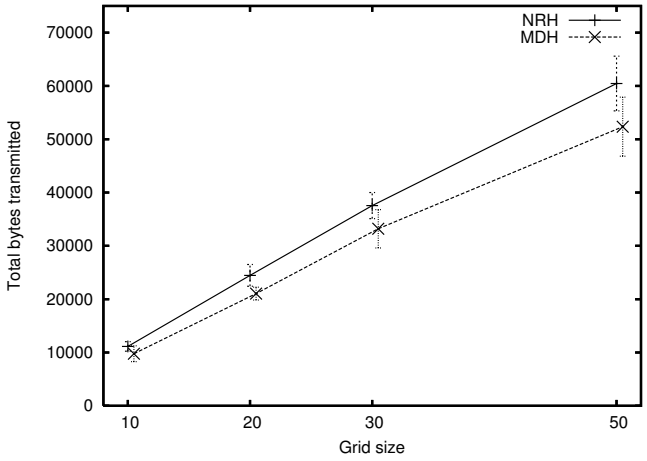

Fig. 13. Total number of bytes transmitted vs. Grid size for NRH and MDH

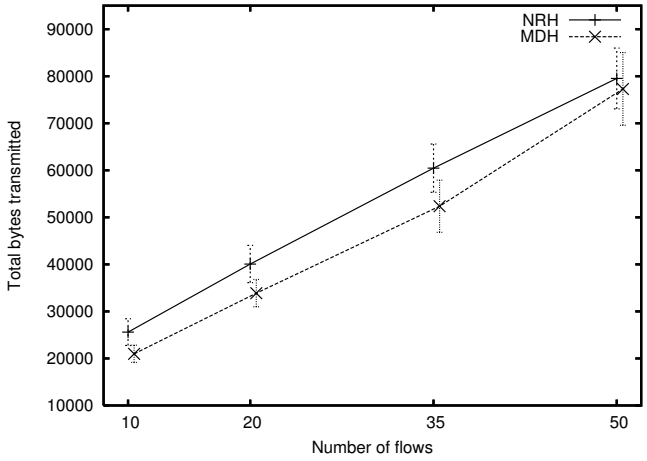

Fig. 14. Total number of bytes transmitted vs. Number of flows for NRH and MDH

a) Comparison of bytes transmitted for NRH and MDH when grid size and number of flows are varied: We first compare the two multicast tree construction heuristics, NRH and MDH, for different grid sizes and number of flows (multicast trees), and we plot this in Figures 13 and 14, respectively. This experiment was performed using our tree construction methodology. We observe that MDH consistently yields lower bytes transmitted than NRH for varying grid sizes and number of flows. Hence, for all further simulations, we use the MDH heuristic.

b) Comparison of bytes transmitted for intelligent tree construction scheme and no aggregation scheme when number of flows are varied: We next compare our tree construction methodology with the simple multicast scheme that does not perform any aggregation. Figure 15 plots this comparison for a grid of 50 $\times$ 50 nodes, when the number of flows is varied. In our setup, n flows corresponds to n publishers, and we plot the costs for 10, 20, 35, and 50 flows. We observe that our scheme performs significantly better than when no aggregation is performed across flows, and is able to reduce the number of bytes transmitted by up to 50%.

c) Comparison of number of bytes transmitted for intelligent tree construction scheme and no aggregation scheme when grid size is varied: We then varied the grid size of the network and studied the performance of the intelligent tree construction and no-aggregation schemes, while the number of multicast trees is kept constant at 35. We plot this in Figure 16. From this figure, we observe that our scheme performs significantly better than the no-aggregation scheme. Our tree construction methodology together with inter-tree optimizations is able to reduce

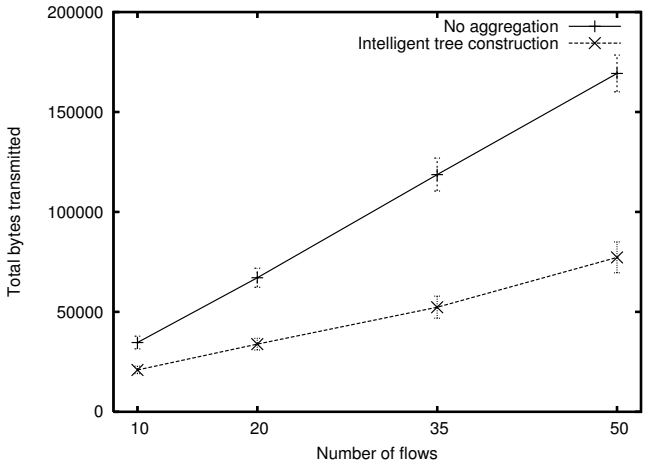

Fig. 15. Total number of bytes transmitted vs. Number of flows for comparing our tree construction methodology with no aggregation

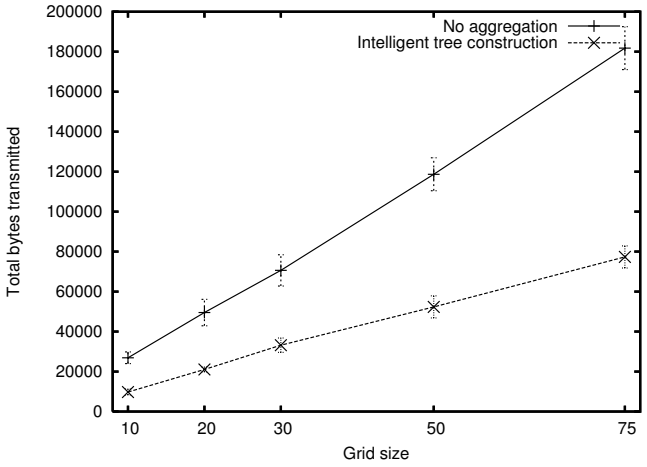

Fig. 16. Total number of bytes transmitted vs. Grid size plot comparing our tree construction methodology with no aggregation

the number of bytes transmitted by more than 50%.

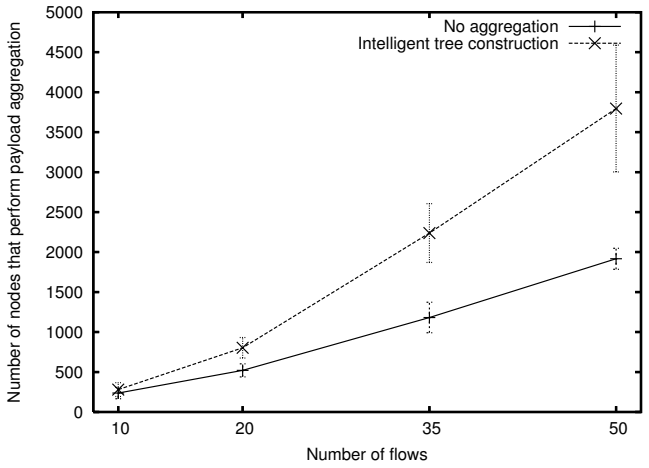

Fig. 17. Number of nodes that perform payload aggregation vs. Number of flows plot comparing our tree construction mechanism with no aggregation

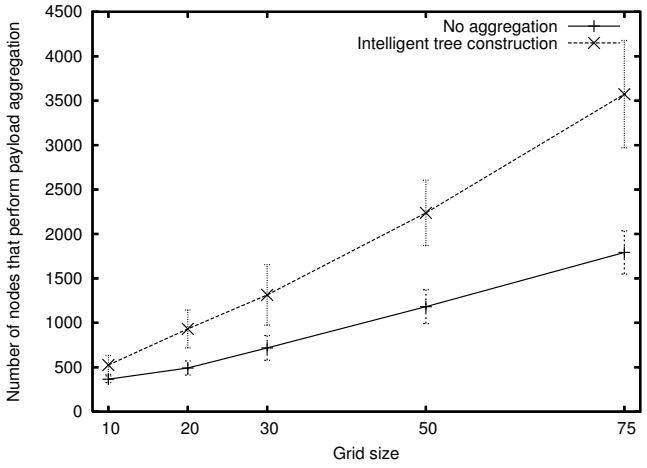

Fig. 18. Number of nodes that perform payload aggregation vs. Grid size plot comparing our tree construction mechanism with no aggregation

d) Comparison of number of nodes performing payload aggregation for opportunistic and intelligent tree construction algorithms when number of flows and grid size are varied: The performance improvement of our proposed solution over the no aggregation scheme is due to both the intelligent tree construction, as well as the inter-flow aggregation. To understand the contribution of each independently, we compare the intelligent tree construction with an opportunistic aggregation algorithm. The opportunistic aggregation algorithm is the same as the no aggregation scheme, except that it performs inter-flow aggregation (opportunistically), but does not attempt to construct multicast trees so as to increase the extent of aggregation. We use the number of nodes that perform

aggregation as the performance metric, as this directly reflects on how efficiently we are able to construct multicast trees to increase the extent of aggregation. Figure 17 plots this comparison for different number of flows (10, 20, 35, and 50). Likewise, Figure 18 plots this comparison for different grid sizes (10, 20, 30, 50, and 75). Both figures show that our methodology to construct multiple multicast trees is able to find many more common intermediate nodes that perform aggregation across multiple flows.

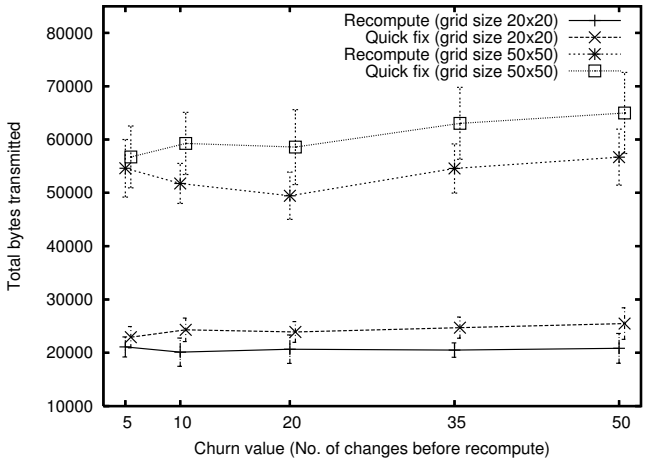

Fig. 19. Comparing cost (in number of bytes transmitted) of the quick fix solution with recomputed cost for different churn values and two grid sizes

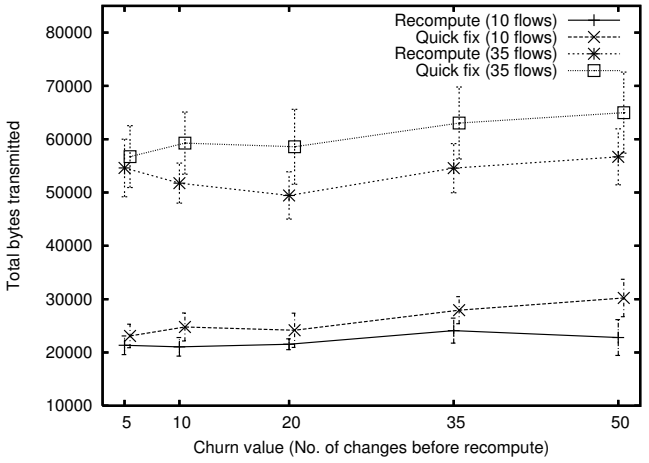

Fig. 20. Comparing cost (in number of bytes transmitted) of the quick fix solution with recomputed cost for different churn values and two flow sizes

e) Study of effect of churn for various grid sizes and number of flows: We introduce churn in the set of subscribers by iteratively removing a subscriber and adding a new subscriber. Publishers queried by the new subscriber are identified in the same manner as before. Since recomputing all the multicast trees of the publishers for every change in the set of subscribers is costly, we propose a quick fix solution. When a subscriber leaves the system, the publishers that serve this subscriber remove the portion of their multicast tree that routes to the subscriber, without disconnecting the tree. When a new subscriber joins, each publisher that serves this subscriber, adds the cheapest path (based on node costs) from the new subscriber to its multicast tree, without changing the rest of the tree. After a fixed time interval, we compute the total cost of all multicast trees constructed by publishers. The number of changes in the set of subscribers (each change is a removal followed by the addition of a subscriber) within this time interval is termed as 'churn value'. A churn value of 50, thus implies that the rate of churn is 10 times faster than when the churn value is 5. We compare the cost of the above mentioned quick fix solution, with the cost of recomputing all the multicast trees based on the new set of subscribers and publishers they query, for different churn values. Figure 19 compares these two costs for different churn values and two grid sizes, when the number of publishers is kept constant at 35. Figure 20 compares the two costs for different churn values and two flow sizes, while the size of the network is kept constant. The number of flows is equal to the number of

publishers in the system, and each flow corresponds to the dissemination of data from a publisher to its subscribers. We observe that as the churn value increases, the cost of the quick fix solution with respect to the recomputed cost also increases. This is due to the fact that as churn increases, the node costs used to add paths to new subscribers become less relevant. Thus, there is a clear trade-off between the quality of aggregation possible amongst multicast trees and the computation overhead. Based on the extent to which poor quality can be tolerated, one could adjust the time after which the multicast trees will be recomputed by the publishers.

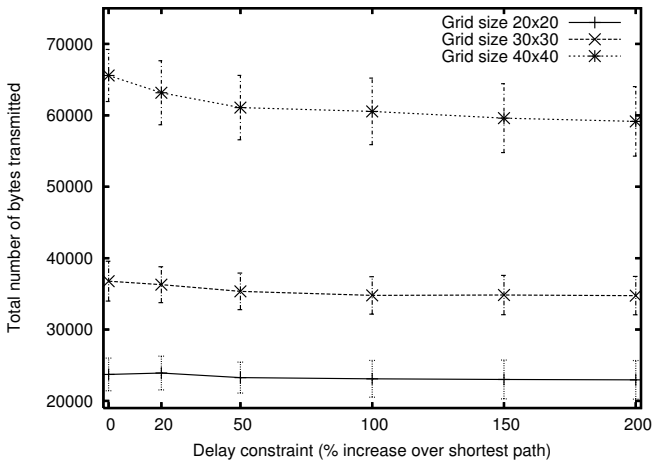

Fig. 21. Total number of bytes transmitted vs. Delay constraint for different grid sizes

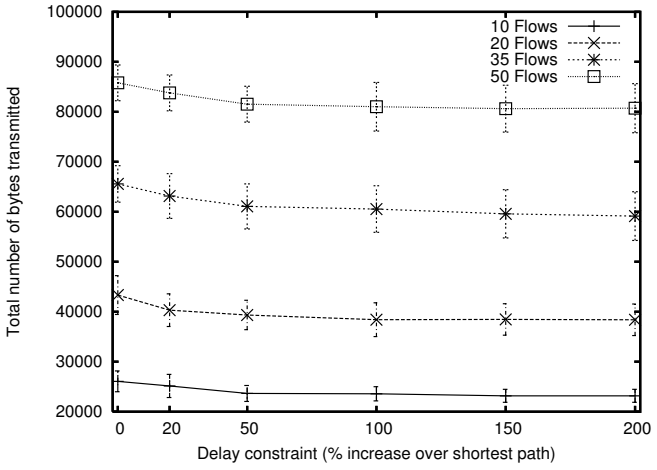

Fig. 22. Total number of bytes transmitted vs. Delay constraint for different number of flows in the network

f) Evaluation of the proposed solution in the presence of delay constraints, when number of flows and grid size are varied: We then evaluate the delay constrained multicast tree construction algorithm proposed in Section III-B. For each *(publisher, subscriber)* pair we impose a delay constraint in terms of number of hops, expressed as a certain percentage more than the shortest path between each node pair. We consider delay constraints of 0, 20, 50, 100, 150, and 200 percent more than the shortest path between each node pair. Delay constraint of 0% more corresponds to the case where only the shortest paths are allowed to be chosen. Delay constraints of 200% more than the shortest paths are typically equivalent to the case of not imposing any delay constraints. Figures 21 and 22 plot the total number of bytes transmitted versus delay constraints for different grid sizes and number of flows (number of flows between the publishers and subscribers) in the network, respectively. Expectedly, we observe that when the delay constraints are more stringent (lower values for the delay constraint), the number of bytes transmitted is higher. However, this increase in the number of bytes transmitted in the presence of delay constraints, is quite small compared to the total cost. This suggests that our tree construction methodology, even in the absence of delay constraints constructs trees with relatively short paths between each publisher and subscriber. The tree construction methodology does not unduly increase the length of paths between publishers and subscribers

in order to be able to perform more aggregation. Further, the increase in the number of bytes transmitted is higher for larger grid sizes and more number of flows in the network.

VII. Conclusions

In this paper, we have modeled the multiple-source multiple-sink data dissemination problem in sensor networks as a content-based publish-subscribe problem. As sensor networks are energy constrained systems, our objective is to reduce the amount of data transmitted by the nodes in the network. While previous research concentrated on optimizing each flow independently, in this paper we addressed the problem of inter-flow energy optimization to achieve network-wide energy savings. We proposed a methodology to intelligently construct multicast trees, one for each publisher, so as to increase the extent of aggregation across multicast trees. We then described how inter-tree optimization at intermediate nodes, through aggregation of multiple publisher-subscriber flows, can be performed. Our simple analytical study of an idealistic system, showed the potential for energy savings using our approach. Through extensive simulation studies we demonstrated that our methodology can reduce the number of bytes transmitted by up to 50%, compared to a solution that does not attempt to aggregate across multiple publisher-subscriber flows. We also extended the basic publish-subscribe problem to incorporate application specific delay constraints. An efficient algorithm was developed, which performs inter-tree optimization while meeting these delay constraints. As a proof of concept, we implemented and evaluated our algorithms on TOSSIM, a TinyOS network simulator.

References

[1] D. Culler, D. Estrin, and M. Srivastava, "Overview of sensor networks," *IEEE Computer*, vol. 37, no. 8, pp. 41–49, August 2004.

[2] Crossbow Technologies. [Online]. Available: http://www.xbow.com/

[3] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking, MOBICOM*, 2000, pp. 56–67.

[4] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation, OSDI '02*, December 2002, pp. 131–146.

[5] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[6] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker, "Difs: a distributed index for features in sensor networks," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003, pp. 163–173.

[7] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with ght, a geographic hash table," *Mob. Netw. Appl.*, vol. 8, no. 4, pp. 427–442, 2003.

[8] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *Sensys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 214–226.

[9] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra, "Matching events in a content-based subscription system," in *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, PODC*, 1999, pp. 53–61.

[10] G. Banavar, T. Chandra, B. Mukherjee, J. N. R. E. Strom, and D. C. Sturman, "An efficient multicast protocol for content-based publish-subscribe systems," in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, ICDCS*, 1999, pp. 262–272.

[11] The Gryphon Project. [Online]. Available: http://www.research.ibm.com/distributedmessaging/gryphon.html

[12] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *Proceedings of the 25th Conference on Computer Communications, INFOCOM '06*, 2006.

[13] D. Sandler, A. Mislove, A. Post, and P. Druschel, "Feedtree: Sharing web micronews with peer-to-peer event notification," in *Proceedings of the 4th International Workshop on Peer-to-Peer Systems, IPTPS*, February 2005.

[14] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in cooperative environments," in *Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP*, 2003, pp. 298–313.

[15] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh, "Efficient epidemic-style protocols for reliable and scalable multicast," in *21st IEEE Symposium on Reliable Distributed Systems, SRDS*, 2002, pp. 180–189.

[16] S. Bhattacharya, H. Kim, S. Prabh, and T. Abdelzaher, "Energy-conserving data placement and asynchronous multicast in wireless sensor networks," in *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, 2003, pp. 173–185.

[17] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: a publish/subscribe middleware for sensor networks," *Springer-Verlag Personal Ubiquitous Comput.*, vol. 10, no. 1, pp. 37–44, 2005.

[18] P. Costa, G. P. Picco, and S. Rossetto, "Publish-subscribe on sensor networks: a semi-probabilistic approach," in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, MASS*, November 2005, pp. 323–332.

[19] V. Marathe and T. Herman, "The design of an interval routing enabled publish/subscribe communications protocol for ad hoc sensor networks," November 2002.

[20] G. Cugola and H.-A. Jacobsen, "Using publish/subscribe middleware for mobile systems," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 4, pp. 25–33, 2002.

[21] Y. Huang and H. Garcia-Molina, "Publish/subscribe in a mobile environment," *Kluwer Wireless Network*, vol. 10, no. 6, pp. 643–652, 2004.

[22] D. Moore, J. Leonard, D. Rus, and S. Teller, "Robust distributed network localization with noisy range measurements," in *Proceedings of the 2nd ACM International conference on Embedded networked sensor systems (SenSys)*, 2004, pp. 50–61.

[23] P.-J. Wan, G. Colinescu, and C.-W. Yi, "Minimum-power multicast routing in static ad hoc wireless networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 3, pp. 507–514, 2004.

[24] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 39–49.

[25] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Sensys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 138–149.

[26] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proceedings of ASPLOS*, November 2000, pp. 93–104.

[27] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 126–137.