

# SENSTRAC: Scalable Querying of SENSOR Networks from Mobile Platforms Using TRACKing-Style Queries\*

Stefan Pleisch      Ken Birman  
Department of Computer Science  
Cornell University, Ithaca, NY 14853  
{pleisch | ken}@cs.cornell.edu

## Abstract

*Future applications running on mobile platforms will sometimes need to query sensors and track sensor data over time. This paper proposes a novel, but natural, solution to querying sensors from mobile platforms, based on the publish-subscribe paradigm. Various options are discussed and the most promising one is included into an implementation. Our evaluation focuses on scalability.*

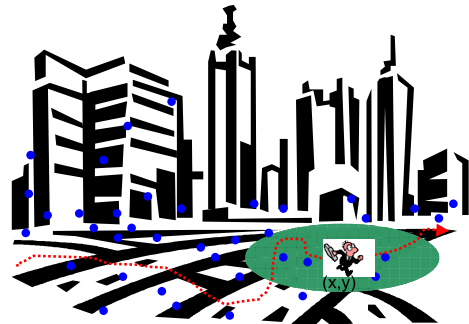
## 1 Introduction

With the widespread availability of wireless technology and the deployment of an increasing variety of sensors, information generated by sensors is becoming available to applications running on mobile nodes. Retrieving this information in a reliable, efficient manner will be an important building block for many applications. However, unreliable, low bandwidth communication links and node mobility make efficient, reliable, and scalable information retrieval a challenge.

Fig. 1 shows a typical scenario, including a person moving through an urban environment. Our work focuses on scenarios with many mobile query nodes (i.e., nodes that issue queries), as is typically the case in urban environments. Another example for this latter case is an amusement park. Every visitor carries a PDA or cell phone running the relevant application. This application allows the user to query the environment in order to ask about queue length, waiting time, ride status, directions (e.g., shortest path), etc.

In both scenarios, the query nodes want to track the data of particular sensors over time. For instance, the length of a waiting queue may be interesting for the query node when it drops below a certain threshold. Hence, the query node needs somehow to be notified when this is the case. Note that queries can be much more complex, involving multiple sensors or types of sensors. While traditional approaches typically assume power-constrained sensor nodes and thus try to minimize the number of sent packets, the applications and sensors we have in mind will not be that power-constrained, although they will need to cope with wireless communication issues.

\*Our effort is supported by the Swiss National Science Foundation (SNF), NSF Trust STC, the NSP NetNOSS program, and the DARPA ACERT program.



**Figure 1. A person is walking on a path through a city area, monitoring the randomly distributed sensors for an increase in temperature, traffic information, pollution, and wind patterns. The person's position is denoted by  $(x, y)$  and the transparent square around him designates the area within which the sensors are of interest to him. On his PDA, the monitoring application constantly updates the results of his query.**

For instance, sensors can be integrated into devices that are connected to electrical power, such as light bulbs. Moreover, critical sensors such as the ones used for disaster response will probably have sufficient power during periods of activity.

We like to think of a mobile query node as passing through a series of stages. At "boot" time, it learns the types of the sensors available in the system, much as a database system reads in its schema. Next, the application expresses an initial set of queries. Now, as the device enters a region, it can seek out the sensors in that region that match the types of interest in the query, giving rise to a virtual database on which the query can be evaluated. An example query is the following: "Return the ID, location, and temperature of any temperature sensor within 0.5 miles to the east of my position that measures a value over 100 degrees F". Finally, depending upon application logic, we may wish to track the evolution of the query result over time, as sensor values change and the querying node moves around, and the application might post additional queries that focus on smaller sets of sensors that are

of special interest, e.g., “Return the wind pattern of any wind sensor within 20m of temperature sensor  $s$ ”. In this scenario, the query nodes want to track the data of particular sensors over time. For instance, the length of a waiting queue may be interesting for the query node when it drops below a certain threshold. Hence, the query node needs somehow to be notified when this is the case. Note that queries can be much more complex, involving multiple sensors or types of sensors. While traditional approaches typically assume power-constrained sensor nodes and thus try to minimize the number of sent packets, the applications and sensors we have in mind will not be that power-constrained, although they will need to cope with wireless communication issues. For instance, sensors can be integrated into devices that are connected to electrical power, such as light bulbs. Moreover, critical sensors such as the ones used for disaster response will probably have sufficient power during periods of activity.

Traditional approaches [6, 22, 24] generally use so-called *in-network processing and aggregation (INA)*, in which the query is flooded within a certain area and a routing tree rooted at the query node is used to return the result to the query node. If the query permits, the results are aggregated on their way to the query node. This type of aggregation, which we call *intra-query aggregation*, reduces the size of the query reply and thus scales well in the number of sensors that are tracked. However, flooding the query to a certain area is not always suitable, especially if the area is remote from the query node. Message overhead increases linearly with the number of query nodes, thus limiting scalability. Moreover, these approaches generally only consider stationary query nodes.

Inspired by the work in [16] and [13], we propose to use the publish-subscribe paradigm to approach the problem of querying sensors from mobile nodes. In our approach, a query node periodically runs an algorithm to identify the sensors it wishes to track. It then “subscribes” to updates, which these sensors periodically “publish”. The query node is now able to repeatedly evaluate the query, presumably updating a map or other application-specific user interface. At some second frequency, the query node recomputes the sensors of interest. Thus perhaps the query node decides which sensors to monitor every minute, but the sensor nodes send updates every few seconds. In a sense, instead of intra-query aggregation we transform the query into subscriptions to sensor updates and aggregate at the level of these subscriptions. This allows us to scale up to a large number of query nodes, as query nodes take advantage of and share preexisting subscriptions.

We mention here for completeness another form of aggregation, which we call *inter-query aggregation* and which aggregates at the level of queries from different query nodes. Inter-query aggregation is a hard problem and falls outside the scope of this paper.

The paper’s main contribution is the novel, highly scalable mapping from queries to topics and the corresponding underlying sensor network structure. More specifically, we show

that structuring the sensor network into a regular grid provides a convenient underlying network structure for this class of applications. The mapping is entirely driven by the querying application. Our performance simulation measures the impact of query node mobility and the number of considered sensors on the quality of the query result and the message overhead. It shows that our approach scales well even for large numbers of query nodes.

Considerable work has been done in the context of generic publish-subscribe systems for wireless ad-hoc networks. Our work is distinguished from this prior art in that we study the mapping of a generic query onto the publish-subscribe paradigm. In contrast, earlier systems provide generic pub/sub and focus on subscriber handovers (e.g., [21, 7]). Here, we use a lease-based approach and we show how this publish-subscribe approach integrates itself well to querying sensor networks. Using this approach leads us to think about a rather conventional layering of mobile query applications onto publish-subscribe over a mesh-structured sensor networks. Such a layering separates the routing layer from the application. This results in two mappings: from the queries to (subscriptions to) topics, and from the topics onto the underlying sensor network structure. While the first mapping involves the properties of the query, the second depends on the properties and the location of the sensors. Routing is left to the network infrastructure.

The contributions of this paper are thus twofold: (1) it proposes a publish-subscribe-based approach as a natural way to scalably query sensors from mobile platforms and track their sensed values over time; (2) it suggests a highly scalable mapping from queries to topics and the corresponding underlying sensor network structure. More specifically, we show that structuring the sensor network into a regular grid provides a convenient underlying network structure for this class of applications.

The remainder of the paper is structured as follows: In Section 2 we define a query model. Section 3 discusses how to generate the result of the query, in particular the mapping from query to topic and from topic to the underlying sensor network. In Section 4, we present SENSTRAC and we show the corresponding simulation results in Section 5. Section 6 puts our work in larger context with existing work and Section 7 concludes the paper.

## 2 Query Model

We consider queries that track sensor data over time; as updates are received the result of the query continuously evolves. Early results of the query may not be entirely accurate but converge rapidly towards an accurate result as more sensors relevant to a query are located and updates are processed. Mobility can also introduce transient inaccuracy. In this sense, our query model is different from the so-called one-shot query model traditionally used to query sensor networks. We believe that it reflects the needs of many types of applications running

on mobile platforms, especially applications monitoring state within the sensor network.

We say that a query *depends on* a sensor if this sensor’s value is required to compute the query result. Conversely, we say that a sensor is *included* by a query.

Typically, queries are constrained by geographical boundaries - generally somewhere in the proximity of the query node. The geographical boundaries are defined by the query node’s *area of interest (AoI)*. For simplicity, we consider a two-dimensional area of interest, which is represented by the smallest square<sup>1</sup> that includes all affected sensors; the generalization to three dimensions is straightforward. The AoI generally moves with the query node.

We also assume that the sets of sensors upon which a query depends overlap considerably between two instances of a query from the same query node, unless the AoI is explicitly reassigned by the query node. In intuitive terms, we assume that the query nodes move at a moderate speed (e.g., walking or running speed), although our approach could also support faster-moving query nodes. Generally, the query node speed that can be supported also depends on the transmission range. The larger the transmission range, the longer two nodes are likely to be within each other’s transmission range.

We support any query that depends on current and future values of a set of sensors; queries that ask for sensed values in the past are not supported. Once a query is running, sensor data is tracked as it evolves, however, hence queries can depend on a sequence of values from a sensor. For instance, the query node can ask to be notified when any temperature sensor detects a drop exceeding 60 degrees F.

Clearly, not all queries can be answered with the same efficiency and accuracy. For instance, queries that contain observer-dependent predicates [4] may (temporarily) miss the detection of this predicate and thus return an inaccurate result. Since updates are communicated via messages to the query node, the evaluation of observer-dependent predicates depends on the order in which these messages are received. The problem of observer-dependent predicates is a general one in distributed systems and not limited to our model, but it does require attention. In our model, accuracy is a complicated function of update rate, mobility rate, transmission ranges, and network load; here, we explore the question using simulations.

We now turn to the question of query representations. Notice that these representations – and in particular the table definitions associated with them – are independent of the underlying communication paradigm. This has the important benefit that the underlying communication paradigm is transparent to the user.

<sup>1</sup>The reason for using a square becomes apparent later in the paper. It is related to the fact that cells are represented as rectangles.

## 2.1 SQL Syntax

Queries are expressed in traditional SQL syntax [6]. They depend on a set of tables defined across the sensor network that conceptually represent the available data. Figure 2 shows the core table. Notice that a sensor can have multiple types, provided that it has a different ID for every type (e.g., SID=1 and SID=4). Nonetheless, for simplicity, we assume in the remainder of the paper that a sensor has a single type.

SID	LOCX	LOCY	TYPE
1	x1	y1	temperature
4	x1	y1	queueLength
2	x2	y2	temperature
3	x3	y3	temperature

**Figure 2. The SensorTypes table contains the ID (SID), the coordinates (LOCX, LOCY), and the type of the sensors.**

The SensorTypes table (and other tables defined below) are not “stored” at any single location. Instead, they can be considered as “virtual” tables. Indeed, the way the information in these tables is collected and stored is the main focus of this paper.

From the definition of the SensorTypes table it becomes immediately clear that the area of interest is defined by the  $x$  and  $y$  coordinates found in the columns LOCX and LOCY. The definition of the SensorTypes table needs to be communicated to the query node as a template form (potentially in XML format). Revisiting the query example from the introduction leads to the following SQL expression (SQL keywords and column names are written with uppercase letters):

```
SELECT Temperature.VALUE, Temperature.SID,
       SensorTypes.LOCX, SensorTypes.LOCY
FROM Temperature, SensorTypes
WHERE Temperature.SID = SensorTypes.SID
AND SensorTypes.LOCX
   BETWEEN xcoord1 AND xcoord2
AND SensorTypes.LOCY
   BETWEEN ycoord1 AND ycoord2
AND VALUE > 100F
```

This query returns the temperature, ID, and location (in  $x$  and  $y$  coordinates) of all sensors whose  $x$  and  $y$  coordinates lie in the interval  $[xcoord1 \dots xcoord2]$  and  $[ycoord1 \dots ycoord2]$ , respectively. For every sensor type there exists a virtual table that contains the columns SID, VALUE, which contains the latest measured temperature value, and TIME. The example refers to the table of temperature sensors (i.e., TYPE = temperature). Again, the definition of this table can be communicated to the interested query nodes upon arrival in the relevant area. This is discussed in the next section.

## 3 Querying the Sensors

Traditionally, a tree-based in-network aggregation protocol queries sensors by flooding the entire query within the

AoI.<sup>2</sup> While this works well for single or small number of subscribers, it does not leverage the fact that multiple subscribers may require a reply from the same sensor. In other words, it is generally very hard to combine the queries from two different subscribers. Hence, such an approach does not scale well in the number of subscribers. On the other hand, it does a good job of supporting intra-query aggregation; particularly if aggregation can be applied and not all results are needed by the subscriber.

A major focus of our work is scalability. Hence, the above approach is not completely satisfactory once the number of subscribers scales beyond a small number.

Moreover, tree-based in-network aggregation works best if the AoI is centered around the query node. However, we can easily imagine cases in which the AoI may not include the location of the query node and may not be directly accessible from the query node, e.g., if it is located in the next valley in a mountainous region. In this case, flooding is no longer the most suitable and efficient query distribution mechanism.

The alternative explored here uses the publish-subscribe (pub/sub) communication paradigm to collect query results. Pub/sub is a widely used communication paradigm and a variety of specifications have appeared over time (e.g., [25, 8]). Its most important feature is the decoupling of the message sender from the receivers, and the asynchronous nature of the communication. In its simplest form, publishers (the sensors in our case) publish messages to a particular *topic*, while subscribers (query nodes) subscribe to all the topics that match their interests. The pub/sub system acts as an intermediary, hence the publisher does not need to know the subscribers. If the subscriber loses interest in the messages published on a particular topic, it unsubscribes from the topic. Many pub/sub systems provide message filtering on topics, which allows a subscriber to specify his interest in more detail. Only messages that match the topic and also pass the filter are delivered to the subscribers.

In contrast to the traditional pub/sub systems, our subscriptions are lease based, i.e., they time out after a certain time. As a consequence, we do not explicitly support the unsubscribe method traditionally present in pub/sub systems. Indeed, unsubscribe only makes sense if it is guaranteed that the unsubscribe eventually removes all subscriptions. In a mobile system, this cannot be ensured and thus seems too much of a constraint to be supported.

Querying sensors using the pub/sub paradigm requires that we implement two different mappings (see Figure 3(a)): (1) the mapping from the query to topic subscriptions, and (2) the mapping from topics to the actual sensors. These two mappings are not entirely independent; a particular choice in one

---

<sup>2</sup>We distinguish between *broadcast*, which is a transmission that is received by all nodes within the transmission range of a node, and *flooding*, in which every receiver re-broadcasts the message and the message eventually reaches all nodes within a certain hop count of the node that has initiated the flooding.

mapping may influence the choices in the other. While (1) is a more abstract, high-level mapping (Section 3.1), (2) is concerned with implementing pub/sub and thus structuring the sensor network (Section 3.2) to provide efficient pub/sub.

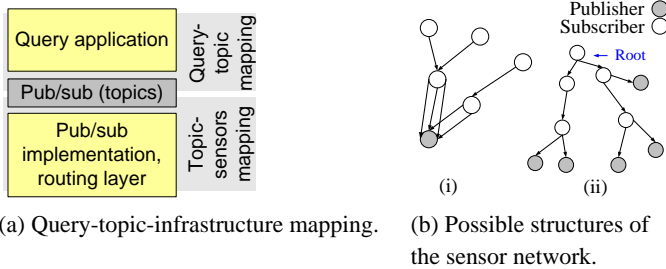
### 3.1 Mapping Queries to Topics

The mapping of queries onto topics involves a trade-off between the number of subscribed topics and the number of messages unnecessarily delivered to the subscriber. Every topic relevant to a query incurs the cost of subscribing to it. In contrast, subscribing to a topic to which sensors not included by the query will publish values can result in situations in which undesired messages are delivered to a query node, forcing it to filter and discard them. Ideally, the query node attempts to minimize the number of sensors from which it receives messages as a result of its topic subscriptions. Due to the properties of pub/sub and the lack of a priori knowledge of the present sensors, the relevant sensors are not explicitly known to the query node. Rather, the query node can only estimate which sensors are considered based on the topic.

A trivial mapping is to allocate one topic to every sensor. Such a mapping can efficiently handle sensor-specific queries, but creates huge numbers of topics, a source of potentially high overhead. More interesting are topics that permit a query source to subscribe to multiple nodes: individual topics to which a group of sensors publish. This raises the question of the most appropriate grouping(s). Since the query node defines an area of interest for its query, grouping the sensors according to geographical regions makes sense, but several other groupings are also worth considering. Insofar as it is very likely that the query node issues queries that depend on types of sensors, it might be useful to support a topic grouping at this granularity. Another grouping might be based on related sensors. This latter grouping presumes intimate knowledge on the relations among the sensor types. For instance, it may make sense to combine temperature sensors and sensors measuring wind patterns, as these types of sensors are likely to be queried at the same time in order to detect fires. Finally, any of the above grouping strategies can be combined.

Since for any query the primary selection criteria on the sensors is the area of interest, it makes sense to first group the sensors into geographic areas and then to assign topics within each area. This limits the geographic span of the sensors that publish to a topic, improving the scalability in the size of the sensor field. Indeed, assume that subscriptions involve sensors spread over a huge area. Lacking a geographic hierarchy, we would face obvious scalability problems. The difficulty here is to find a geographical grouping that maps closest most query nodes' typical area of interest. The simple grouping considered here is that of a grid of cells. Within a cell, sensors can then be organized into groups according to their types. Using this grouping, queries can be quite easily mapped onto topics.

Depending on the topics, additional infrastructure may be



**Figure 3. Mappings and sensor network structures.**

needed to make sure that nodes can compute the mapping from query to topics. In order to perform the mapping, query nodes (and potentially sensor nodes, if they perform the mapping) have to be aware of the available topics. If topics are chosen in some non-deterministic way, then this information needs to be explicitly communicated to the query node. To simplify the identification of topics and to avoid having to explicitly send topic names around in the network, we define a one-to-one mapping between topic names, types, IDs, and geographical regions. Assume, for instance, that a topic is defined for every cell in a grid overlaying the sensors. Such a topic could have the same name as the corresponding cell, e.g., *B-4*, while temperature sensors in this cell publish to topic */B-4/Temperature*. Revisiting the SQL query in Section 2.1 results in a subscription to topics */B-4/Temperature*, */B-5/Temperature*, */C-4/Temperature* and */C-5/Temperature*, assuming that the AoI is covered by cells *B-4, B-5, C-4* and *C-5*.

### 3.1.1 Aggregation and Filters

In-network processing and aggregation can greatly reduce the number of messages sent in the network [22]. However, in the pub/sub communication paradigm, in-network aggregation is not a natural fit. Although filters derived from the queries can prevent the sending of useless messages, true query result aggregation is not possible. Assume, for instance, a query that asks for the maximal temperature in a region. Traditional pub/sub cannot process the query in the network and hence it cannot compute the maxima. The reason for this is that filters are applied to single messages and generally do not store state between messages, as is generally needed for computing aggregation functions such as MAX. Indeed, this separation of routing and query application is the major focus in this paper. This is not necessarily a drawback: many application areas, e.g., scientific observations, require the entire set of data from included sensors for later postprocessing or verification, rather than an aggregate of the same data.

Yet, filtering can still occur. Assume that the query is mapped onto a topic that covers a larger area than the AoI. In this case, a filter can be defined that only lets pass messages from sensors located within the AoI. Some of these filters can be generated automatically from the query by convert-

ing predicates on columns (in the WHERE clause of the SQL statement) to filters. For instance, predicate “VALUE > 100” in the example of Section 2.1 could be converted into the corresponding filter. This requires that the update messages from the sensors have a format known to the query node (or the mapping engine generating the filters), perhaps including the column names as attribute names in update messages. Knowing the topics to which the update message is published allows any node to infer the type of the sensor.

## 3.2 Structuring the Sensor Network

In this section, we present our pub/sub architecture for query applications in sensor networks and discuss the kind of topics that can be efficiently supported.

Sensors and query nodes communicate by establishing (mobile) wireless ad hoc networks. Nodes within transmission range of each other can communicate directly. More distant nodes rely on other nodes to forward messages. A routing protocol sets up a route between a sender and a distant destination.

In such settings, a pub/sub architecture can use peer-to-peer-based communication among subscribers and publishers, or it can rely on brokers that intermediate between subscribers and publishers. While the peer-to-peer-based architecture has the advantage of balancing the load more evenly across the sensor network, it suffers from disadvantages in terms of scalability. Indeed, this communication pattern prevents subscribers from taking advantage of already existing subscriptions. Instead, each subscriber needs to explicitly connect to the publishers. In a sensor network, it is also highly likely that even geographically close subscribers establish completely different routes to the same publisher. As a consequence, any published message is forwarded along both paths to these subscribers. However, instead of sending two messages, it is more efficient to send only a single message along a common path, and only then duplicate the message if the paths to the subscribers separate. Shared paths thus reduce the number of transmissions by intermediate nodes. This can be achieved by relying on intermediate brokers, which take over the role as “routers”. Updates are sent as a single copy between brokers and only need to be duplicated for the last part of the routing path, namely from the broker to the subscribers. Moreover, the broker does not need to forward a new subscription if it already has existing subscriptions that consider all topics that are in the new one.

On the routing level, peer-to-peer-based architectures incur high route maintenance costs, as they need to maintain many routes between publisher and subscriber pairs. Maintaining routes in mobile environments is expensive, in particular if the publisher or the subscriber is mobile. Moreover, discovering a route to the publisher may be costly, especially if the publisher is far away, and may involve flooding parts of the network.

In contrast, a broker-based architecture can establish routes between pairs of brokers. Since in our setting the sensor nodes

are (mostly) stationary, brokers should be positioned on those. Using an overlay network that primarily routes messages between brokers has the distinct advantage of keeping a large part of the routing infrastructure stable and only modifying the relatively short routes that suffer from mobility, i.e., the routes between subscribers and brokers. By reusing existing routes as much as possible, the route maintenance cost is distributed over multiple subscriptions. Moreover, the brokers will eventually be well-known in the sensor network and many sensors will have routing information to the brokers in their routing table. In comparison, some peer-to-peer-based architectures require nodes to store routing information to all publishers.

The remainder of this section looks more closely at peer-to-peer-based pub/sub architectures, exploring three scenarios (pub/sub over point-to-point, over tree-based, and over multicast communication) and giving reasons that make them not practical in our setting. We then present our broker-based pub/sub architecture (Section 3.2.1).

**Pub/Sub Over Point-to-Point** We considered, but decided against using point-to-point messages (e.g., using AODV routing [26]) to communicate subscriptions (see Fig. 3(i)) and the updates between single publishers (i.e., sensors) and the subscribers (i.e., query nodes). The core issue is that query nodes are mobile and thus a huge routing overhead is incurred. Moreover, if a publisher is interested in the updates of multiple sensors (as is typically the case for generic queries), it needs to send a point-to-point message to every publisher in order to subscribe to this sensor. Clearly, this does not scale well even for a small AoI and our earlier work has shown that the network gets overloaded and thus packets are dropped in the sending buffers of the nodes. We will not further explore this option.

**Pub/Sub Over Tree-Based Approaches** Another option is to build an overlay routing tree either (1) rooted at every publisher, or (2) rooted at the subscribers. Approach (1) constructs one routing tree per publisher (see Fig. 3b(ii)), in which the subscribers are generally located at the leaves of the tree (some subscribers may act as intermediate nodes). If a subscriber moves, then the routing tree of every included publisher needs to be adjusted, which leads to a high route maintenance overhead. Moreover, route discovery requires that the publishers find at least one node that knows a route to the tree. If the publisher is very far from the subscriber, route discovery can become expensive.

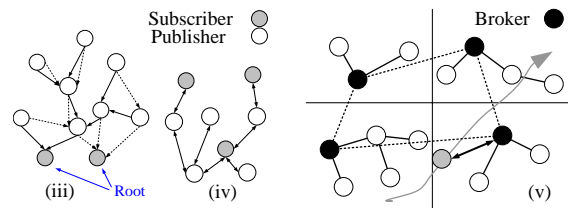
Approach (2), inspired by prior research on in-network aggregation [9, 22], constructs a tree for every subscriber (see Fig. 4(iii)). The subscriber floods its subscriptions into the sensor network, with the relevant publishers responding along a tree structure implicitly built during the flooding phase. This scales well in the number of included publishers, but not in the number of subscribers, due to the need for one flooding broadcast per issued subscription. It is suitable for stationary

subscribers, but lacks appropriate support for their mobility, making tracking sensors over time costly. To adjust to subscriber mobility, one could imagine a so-called proxy root that forwards the results to the subscriber. This requires that a path exists from the proxy to the subscriber. Moreover, every subscriber potentially uses another proxy, which does not scale well, raising again route discovery and maintenance issues. Both Approach (1) and (2) are not explored any further. A representative finding, comparing Approach (2) with the algorithm we favored, appears in Fig. 6(g) and (h).

**Pub/Sub over Multicast Groups** The above approaches all set up routing infrastructures dedicated to single publishers or subscribers, or both in the case of the point-to-point approach. They all suffer from drawbacks in our model. Hence, we now study an approach that implements pub/sub on top of multicast groups such as provided by MAODV [27]. Publishers and subscribers express their interest in a particular topic by joining the corresponding multicast group and thus become a *member* of this group (see Fig. 4(iv)). All messages are published to the entire group; hence, the publishers (in our case the sensor nodes) also receive the messages, which adds additional load on the network. This is especially the case if the sensors are on the path between two query nodes in the multicast tree. In general, the load increases with the number of members. It also increases with the number of nodes that route updates between members of the group, without being part of the group themselves.

The route discovery problem is also prominent in this approach. To join a group, the subscriber contacts any member of the group, which is reasonably efficient only if a member is in proximity of the subscriber or if the subscriber, or any other node close by, knows how to find a member of the group.

Finally, maintaining the multicast group is expensive in terms of message overhead. Frequent changes as caused by the mobility of the subscribers may incur a high overhead. Earlier experiments have confirmed this and we will not further pursue this approach.



**Figure 4. Further options for structuring the network.**

### 3.2.1 Broker-based Architecture

Having ruled out the use of peer-to-peer-based architectures, we now present the architecture that we will explore further. To address the route discovery and maintenance issues we use a broker-based architecture, in which some sen-

sensors are designated as brokers with a special role that we address below. The first issue that arises is how to position the brokers in the network. As queries use the AoI as primary criterion (see Section 2), we cluster the network into geographic regions. We thereby assume that the sensors know their approximate coordinates, either by using on-board GPS or by inferring their location from their neighbors [23]. The clustering can either be static or dynamically computed. We adopt a static clustering by overlaying a static grid over the sensor space, dividing the latter into a set of *cells*. The important advantage of static clustering is that every node a priori knows the clustering and can compute another node’s cell ID solely based on this node’s coordinates. In contrast, dynamic clustering dynamically arranges the sensor nodes into clusters according to some criteria. As a consequence, the clustering needs to be communicated to the query nodes.

Each cell contains one or, in the case of partitions within the cell, multiple brokers. Publishers send updates to the broker in their cell (or in their partition within their cell). Subscribers send their subscriptions to any close broker (see Fig. 4(v)). The brokers communicate updates and subscriptions among them, along the “logical” communication links represented by dashed lines in Fig. 4(v). Hence, the communication between cells is routed through the brokers.

The selection of brokers is an interesting topic, but outside of our scope here. We will explain how to select brokers later when we present our implementation.

### 3.3 Combining the Two Mappings

In the previous sections, we have discussed possibilities for the query-topic mapping (upper mapping) and for structuring the sensor network (lower mapping). In principle, any combination between the two mappings is possible. For instance, we could create one topic per sensor in the upper mapping, and structure the network such that all sensors are in the same multicast group, or we could create group topics and acquire updates using the point-to-point approach to implement pub/sub. However, not all combinations make sense and the two mappings are actually not independent from each other.

Our discussion in Section 3.2 has shown that the lower mapping drives the decision of the best mapping. As we have proposed a broker-based architecture based on a grid of cells, we define a topic structure that matches this choice. On the first level of the topic hierarchy are the topics that refer to cells. Then, on the second level, topics can be defined based on the type of sensors or any other criteria that is adequate for the sensors in the cell. For instance, topic “/B-4/Temperature” refers to all temperature sensors in Cell B-4.

A special case arises if the query considers a single or small number of sensors. In this case, we could define single sensor topics and map the query to the corresponding topic(s). However, single sensor topics need to be defined at the second level or higher of the topic hierarchy. This simplifies the routing of the subscription to the correct broker. Otherwise,

routing subscriptions to the broker will require that each intermediate broker knows the location of the sensor. Another approach is to subscribe to a more general topic and to define a filter based on the SID attribute (see Section 3.1). As queries to specific sensors are rare, we use this approach.

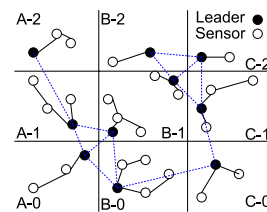
## 4 SENSTRAC

In this section, we present the system we implemented. A general principle is that all data related to maintaining the routing and pub/sub infrastructure (e.g., subscriptions) stored at any node expires after some time, unless it has been updated or renewed in the meantime. We use this approach to make sure that the data referring to crashed or unreachable sensors (sensors to which the current sensor does not have a communication path) eventually disappears from the system. Hence, after a node failure all traces of this node will eventually be removed. Also, the system may temporarily be in a transitional phase. Tagging data with an expiry date allows the system to leave transitional (suboptimal) states and arrive at (optimal) stable states.

### 4.1 Structure of Sensor Network

We distinguish between *intra-cell* and *inter-cell* routing, using a variant of the landmark hierarchy [30]. Intra-cell routing addresses the routing infrastructure among the (sensor) nodes within a particular cell. Inter-cell routing, in contrast, governs the routing among the cells.

**Intra-Cell Routing.** The scope of messages sent in the context of intra-cell routing is limited to that particular cell. Flooded messages generally travel one hop into the next cell. Any node that receives a flooded message from a foreign cell does not echo this message.



**Figure 5. Intra-cell routing (solid lines) and inter-cell overlay network (dashed lines).**

In the intra-cell routing scheme, we rely on a *leader* node. Leaders handle routing, while brokers are query processing intermediaries. Although we will use the leaders as brokers in our implementation, we use different terms because one could imagine using different nodes for these different roles. For instance, in some settings one may prefer to use a specially powerful node as a broker. The leader is selected dynamically, according to a certain criteria. This criteria needs to be deterministic and can be, for instance, the node with the lowest ID among the nodes proposing themselves as leaders.

Every leader periodically floods a leader hello message to the cell. Upon reception of the leader message, the sensors in the same cell build a shortest path tree rooted at the cell leader (see Fig. 5). Leaders can change over time. For instance, if one leader is running low on resources, it can stop proposing to be a leader, and eventually another node will take over [10]. The same mechanism also handles leader failures.

Partitions within a single cell in the sensor network may lead to multiple leaders (e.g., Cell B-1 in Fig. 5). These partitions may have been artificially introduced by subdividing the grid into cells. Partitions may merge once communication links are reestablished and a single leader will emerge. Otherwise, multiple leaders (one per intra-cell partition) persist. They will be able to communicate through leaders of other cells, as discussed in the next section.

**Inter-Cell Routing.** Inter-cell routing governs communication among the leaders of different cells. All messages are sent from and directed to a leader. We use two mechanisms by which leaders learn of the existence of other leaders. The first periodically forwards leader hello messages overheard from neighboring cells to the leader of this cell. We call the leaders of neighboring cells, of which the leader learns, *neighbor leaders*.

The second is based on gossiping leader information among neighbor leaders. Periodically, a leader chooses a random subset of all leaders it is aware of within a certain range (measured in leader hops, i.e., the number of intermediate leaders before getting to the destination leader) and sends it to a random number of its neighbor leaders. Gossiping leader information results in an overlay mesh among the leaders (see Fig. 5).

We use the Ad hoc On-demand Distance Vector (AODV) [26] routing protocol for gossiping and, in general, for all messages (e.g., also updates) sent between two leaders. AODV has been designed for mobile ad hoc networks. As our sensors are relatively stationary we set AODV’s configuration parameters to conservative values, thereby minimizing the message overhead while still handling some degree of sensor mobility. Clearly, other routing protocols are also possible (e.g., [20]). Also, a purely geographic routing is possible. However, such protocols do not always find a path to the destination, although the sensor network may be connected. This is especially the case if the routing path needs to route around obstacles such as areas with low or no connectivity. We leave the exploration of alternatives for future study.

## 4.2 Implementing Pub/Sub

A broker channels subscriptions and updates between interested parties. As noted, our implementation reuses the leaders for this second role. Sensors publish their updates by sending them to the cell leader (broker). Query nodes send their subscriptions to the closest broker, usually the one within the same cell.

Upon reception of a subscription, the broker adds the subscription to its subscription list. Then, it maps the subscription to the cells and subscribes to the broker that is on the shortest path to the broker(s) in the corresponding cells. This is done recursively until either no more subscriptions are available, or a broker has already subscribed to another broker for the same subscription. The latter test avoids loops in the subscription mechanism, which occur if the overlay broker routing table is incomplete at some brokers and does not contain the shortest paths, but some longer paths. In this case, suboptimal subscription decisions may be taken. However, this only occurs during transitional phases in the system (after link failures between brokers, or the failure or re-affectation of a broker).

Although loops are prevented in the subscription mechanism, update messages may still loop. We use a simple form of caching to prevent any broker from forwarding an update message more than once.

Having forwarded a subscription for a topic to another broker, a broker will not forward any new subscriptions on the same topic for some time  $t$ . This allows the broker to take advantage of already existing subscriptions and reduces the number of subscriptions sent between brokers. However, if the first subscription is lost, then no updates published to this topic will arrive during time  $t$  and the time until a new subscription is received by the broker. Hence, the value chosen for  $t$  balances a tradeoff between reducing the number of sent subscriptions and the consequences of subscription message loss.

## 4.3 Routing Between Query Node and Broker

The query node periodically queries its neighboring sensors for their leader. Recall that, in our implementation, a leader is also a broker, hence the query node receives information about the closest broker. Often, there will only be a single broker, but if it receives information about more than one, the query node picks one used previously, or otherwise breaking ties by favoring one that is closer (in hop count).

The routing between query node and broker takes advantage of the fact that each sensor node knows a route to the cell leader (broker) and thus easily can route the message from the query node to its leader. While the routing path from the query node to the leader is given, the inverse is not true. To forward the updates to the query node, we use the following approach: When intermediate sensor nodes forward subscriptions from the query node to the broker, they store a copy locally.<sup>3</sup> They then use the stored subscriptions to determine whether or not to forward an update from the broker to the query node. If an active subscription exists and the corresponding filters are passed, the intermediate sensor node rebroadcasts the update. The advantage of using broadcasts rather than point-to-point communication is that a parent node only broadcasts an update once, instead of sending it to all its children sensor nodes

<sup>3</sup>To save memory, intermediate sensor nodes can also store an aggregate of the subscriptions and filters.

one after the other. Thus, the parent sensor node in the routing tree does not explicitly know its children nodes. Rather, it forwards the update based on the currently active subscriptions and the corresponding filters. This routing scheme is different from the approaches traditionally used in multicast trees.

When the query node moves, the routing path to the leader may be broken. As a consequence, the first sensor in the routing path from the query node to the leader periodically broadcasts a hello message during periods of broadcast inactivity. Any query node relying on this node for routing to the leader receives the hello message and uses it to detect a link breakage. If it detects a link breakage, it tries to establish a new route to the same broker, resending a previous subscription. Link breakages are only repaired between two subscriptions by the same query node. When a link breaks, the query node tries to reconnect to the same broker. If none of its neighboring sensor node knows a path to this broker – indicating that the query node has changed cell – a new subscription is sent to the new broker.

## 5 Simulation

For our simulation we used JiST/SWANS v1.0.4 [1, 5], a simulation environment for ad hoc networks. Java applications written for a real deployment can be ported to the simulation environment and then placed under a variety of simulated scenarios and loads. Jist/Swans intercepts the calls to the communication layer and dynamically transforms them into calls to the simulator’s communication package.

### 5.1 Setup

We consider a set of sensor and query nodes. While sensor nodes are stationary or relatively immobile, query nodes are mobile. Communication between two nodes  $m$  and  $n$  occurs in an ad hoc manner and may be asymmetric, i.e.,  $n$  may be able to communicate with  $m$ , but the inverse may not be possible. The transmission range of a node is 88m. We use a CSMA MAC protocol as defined in the 802.11 standards [14], but without the RTC/CTS and ACK mechanism. Communication can be subject to interference, in which case the message cannot be received. Interference can occur without the sender being able to detect it (this is called the *hidden terminal problem* [3]).

Our work does not inject artificial packet loss, although we do model disconnections due to mobility, transmission range limits, and the hidden terminal problem just mentioned (using JiST/SWANS’ *RadioNoiseIndep* package, which uses a radio model identical to ns-2 [2]). Unless otherwise mentioned, we use the default values defined in JiST/SWANS, such as a bandwidth of 1Mb/s.

In general, the variance in a single node’s simulation results for ad hoc networks is high. This is due to the many sources of randomness: distribution of the sensor nodes, the paths of the query node, the time the sensors send an update, etc. To compensate for this in the single query node case, the query node moves along a straight line from coordinates

[200, 200] to [900, 900] in the field of 1200x1200m with origin [0, 0]. A total of 600 sensor nodes are uniformly distributed within the sensor field, which ensures, with high probability, that at least one route exists between any two sensor nodes. In other words, all sensor nodes are in the same partition (with the exception of partitions introduced by the gridding.) For multiple query nodes, we use the random waypoint model [17] with a fixed speed and zero pause time, thereby removing the randomness caused by varying speeds and pause times.<sup>4</sup> We limit the scope of the query node’s movement such that its AoI does not extend beyond the field boundaries. This prevents unrealistic results due to the measurement setup.

The sensor and query nodes start up at random times and positions. When they are all up and running, we start our measurements (also called steady-state simulation).

## 5.2 Results

In this section, we present the results of our simulation. The query node periodically, every 60s, sends its query, i.e., subscribes to the relevant sensors. The sensors publish their current value every 50s. To enable the evaluation of our approach, every sensor reports the current time as its value. This value is then used to measure three types of sensor coverage, i.e., the sensors from which the query node received an update of the corresponding type over all sensors currently in the AoI: *recent* denotes the ratio of sensors for which the query node has received an update within the latest subscription period (known value is less than 60s old); *1-stale* the ratio from which the query node has missed the update in the latest subscription period (60 to 120s old); and *n-stale* if the latest update is older (more than 120s old). N-stale is sometimes omitted for space reasons. The sum of recent, 1-stale, and n-stale sensor coverage is 1, meaning that 100% of the sensors in the AoI are covered.

The sensors are added to the sensor field at times uniformly distributed between 0 and 100s and the cell size is set to 300m. The default AoI contains all sensors with coordinates  $600 \leq x, y \leq 900$  (corresponding to a single cell), amounting to roughly 40 to 50 sensors with a total number of 600 sensors, or with distance smaller than 200m from the query node’s position. All results give the average over 20 runs in different uniform sensor distributions. The approximate 95%-confidence intervals (CI) are:  $\pm 0.05$  for sensor coverage in the case of single query nodes;  $\pm 0.1$  for single query node with moving AoI;  $\pm 0.001$  for multiple query nodes; and  $\pm 5\%$  for message overhead measurements.

In the following, we evaluate the following properties of SENSTRAC: its behavior in the face of query node mobility, its scalability, the impact of the mapping of the query to topics, and the message overhead. Unless explicitly stated otherwise, we use the above default values in our measurements.

<sup>4</sup>Note that in [32] it has been shown that the random waypoint model is not entirely appropriate. However, for our measurements, this has no immediate impact.

### 5.2.1 Mobility

We first look at the effects of the query node mobility on sensor coverage.

*Stationary Query Node and Stationary AoI.* We start with the case of a single stationary query node located at position [200, 200], which thus always generates identical queries. The upper-most curve in Fig. 6(a) shows the recent sensor coverage. Thus, the query node knows the most recent value of virtually all the sensors in the AoI.

*Mobile Query Node and Stationary AoI.* The other three curves in Fig. 6(a) show the results for the same experiment, but with a mobile query node. The vertical lines in Fig. 6(a) indicate the time at which the query node changes from one cell to the other. Such a cell change triggers resubscriptions to the broker of the next cell, which causes a temporary decrease in the sensor coverage (around 780s and from 1140 to 1260s). In Fig. 6(b) we show the result for a single query node that uses the random waypoint model.

*Mobile Query Node and Mobile AoI.* The graph in Fig. 6(c) shows the case of a single query node travelling through the sensor field along a straight line. In this simulation run, the AoI travels with the query node. More specifically, the query node is interested in all the sensors that are within 200m of its current position. Compared to the case of a stationary AoI, the moving AoI may involve at times more than one cell and thus requires to contact more than one broker. As a consequence, the sensor coverage fluctuates to a greater degree, and is generally slightly lower than in the case of stationary AoI. In the same graph, the forth curve (y2-axis) illustrates the lease-based nature of SENSTRAC. It displays the message overhead over time with 100 query nodes and shows that the increase in the message overhead eventually diminishes once all the subscriptions time out and update messages are no longer sent to the query nodes (around 1200s).

The next two graphs show the impact of the update rate and the speed on sensor coverage. In Fig. 6(d) we use the same setup as in Fig. 6(a), but the sensors send an update every 30, 50, and 70 seconds. It comes as no surprise that the recent sensor coverage decreases with decreasing update rate. Similarly, the recent sensor coverage generally decreases with increasing speed of the query node. The results of this experiment, again with the query node moving along a straight line but with a stationary AoI, are shown in Fig. 6(e). Notice that query nodes with higher speeds reach the end point of the line (at location [900,900]) faster and thus the simulation run stops earlier.

### 5.2.2 Scalability

We now evaluate the scalability of SENSTRAC with respect to the number of sensor and query nodes. Fig. 6(f) shows the sensor coverage (y1-axis) and the message overhead (y2-axis) averaged over time (1200 seconds) and all the query nodes. It shows that the message overhead increases with increasing number of sensor nodes. Clearly, the more

sensor nodes are within the AoI, the more updates are routed to the query node. However, the sensor coverage is nearly the same for all the considered numbers of sensor nodes. Moreover, the message overhead increases with increasing number of query nodes.

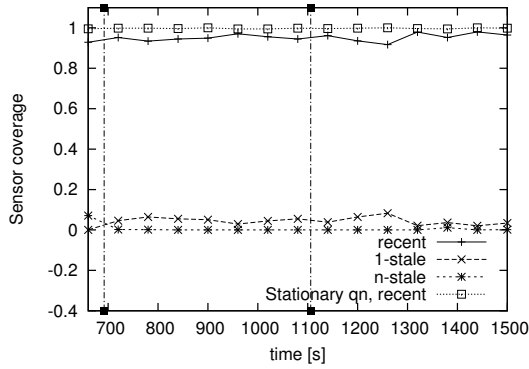
In Fig. 6(g), we compare the message overhead of SENSTRAC with traditional tree-based in-network aggregation (INA). We consider 700 sensors and use the same AoI with radius 200m for both, and we do not aggregate query replies. Notice that the choice of this particular AoI is biased towards the INA approach. Since the INA approach uses polling, we flood the query at double the frequency than in SENSTRAC. Still, an updated sensor value may not be discovered in the INA approach for 30s in the worst case. We can see that the message overhead for SENSTRAC increases fast for low numbers of query nodes, but then the increase diminishes with increasing number of query nodes (see curve labelled “total”). This is an indication for the scalability of our approach. Indeed, with a high number of query nodes it becomes more likely that a broker to which a query node sends a subscription already has an active subscription. This allows newly arriving query nodes to take advantage of existing subscriptions. In contrast, INA message overhead costs increase linearly with the number of query nodes. The two message types in SENSTRAC that add the most to the message overhead are the inter-cell messages and the query node messages (qn). The former result from sending updates between the brokers and include all the messages sent by intermediate hops, while the latter count the number of messages exchanged between query node and the broker, including intermediate nodes. Moreover, intra-cell packets are the messages needed to maintain the routing structure within a cell and the updates sent to the cell leader. Finally, the AODV packets indicate the number of packets sent to maintain/discover the routes between the cell brokers. Total packets gives the sum of these packets.

Fig. 6(h) shows the corresponding sensor coverage. SENSTRAC’s recent sensor coverage is slightly higher than INA’s for more than 60 query nodes.

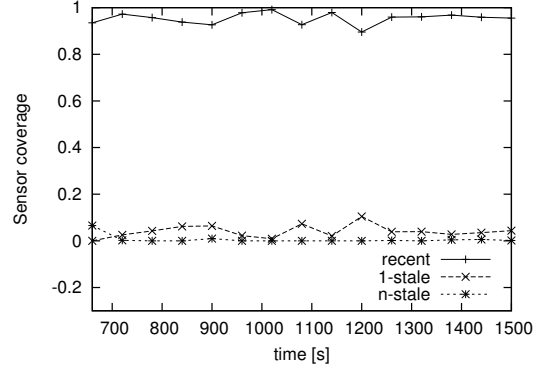
### 5.2.3 Mapping of Query to Topics

In this section, we evaluate SENSTRAC with varying queries and cell sizes.

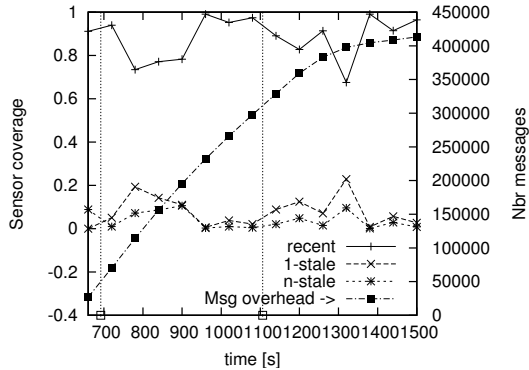
*Varying cell size.* Fig. 7(a) and Fig. 7(b) highlight the impact of the cell size on sensor coverage and message overhead. Consider the curve for cell size 600 in Fig. 7(a). At time 1106 the query node changes from one cell into another, right at the intersection of four cells. As a consequence, the resubscription process leads to a sharp decrease in sensor coverage. Before this, no cell changes occurred and the sensor coverage was very high in comparison with the graphs for smaller cell sizes. Fig. 7(b) measures the network load with respect to varying cell sizes. The considered packet types are the same as in Fig. 6(g). With increasing cell size, the number of intra-cell and qn packets increases and the number of inter-



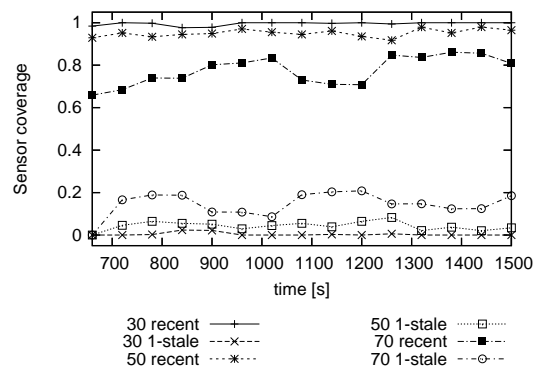
(a) Single query node (qn) with stationary AoI.



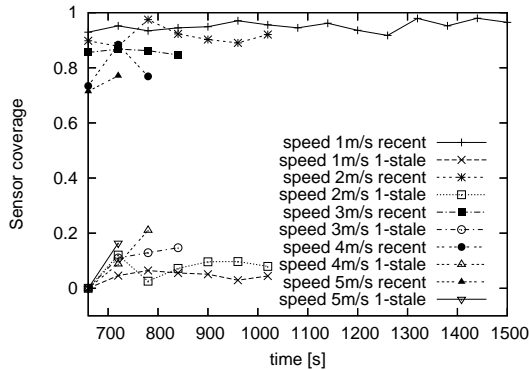
(b) Single query node uses random waypoint model.



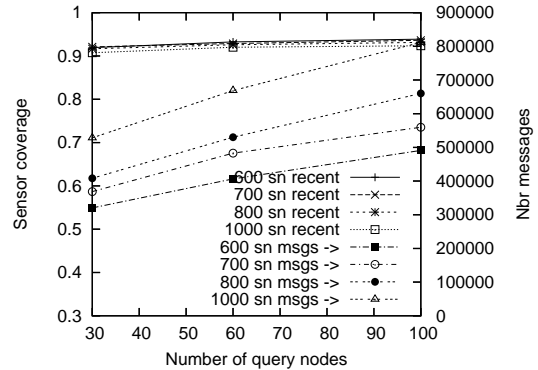
(c) Single query node and AoI move along same line.



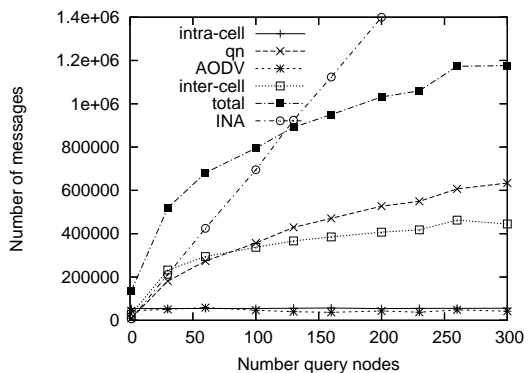
(d) Single query node, varying update rate (in seconds).



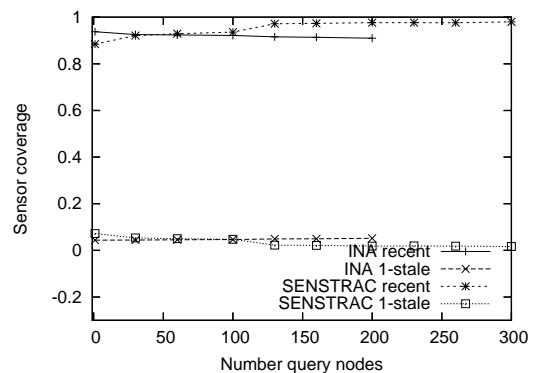
(e) Single query node, varying speed.



(f) Mobile AoI.



(g) 700 sn, mobile AoI, comparison with INA.



(h) Sensor coverage comparison with INA.

**Figure 6. Single query node sending queries with AoI range 200m and a speed 1m/s.**

cell packets decreases. In the case of a single cell (of length 1200m), no inter-cell packets are sent. The AODV messages decrease as well, except in the case of four cells of length 600m. Here, the cell leaders are generally many hops away so that the route discovery becomes quite expensive. It may even involve sending multiple discovery messages with increasing hop count. Our chosen cell size is 300m, which has the lowest total message overhead. In the graph we do not consider the messages sent by the query node. Since we only consider a single query node, this number is very low and thus not significant.

### 5.2.4 Message overhead.

Since SENSTRAC assigns special functions to some nodes (for instance the brokers), the load may not be equally distributed among the nodes unless the load-balancing scheme explained in Section 4.1 is used. In Fig. 8 we display the total number of messages sent over time by every node, without using this load-balancing scheme. The nodes with the highest message load are the leaders/brokers of cells in the center of the field through which the query node moves.

Another indicator on the load of a node is the maximum size of its message output queue. By nature of the CSMA-based MAC protocol, the sender backs off for some time if the transmission medium is busy, before attempting to resend the message. If many messages are to be sent during this time, this leads to an increasing number of messages waiting to be sent in the output queue. The longer the output queue, the higher the message latency. As some messages (e.g., the subscription messages) are more important than others, a large impact on sensor coverage may result from this. We have measured the maximum output queue size in a field with 700 sensor nodes. With a single query node moving according to the random waypoint model, only four sensor nodes have a maximum output queue size two, the others have at most one message at the time in the queue. Running the same experiment for 60 query nodes increases the maximum output queue size to three (for two nodes), and two for 14 nodes. Thus, there is no significant increase.

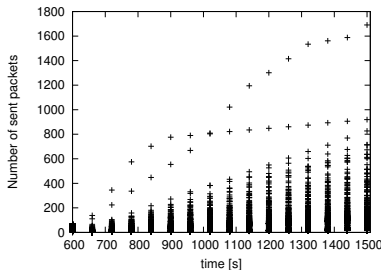


Figure 8. Sent packets per node over time.

In Fig. 9(a) we show the impact of the AoI on the quality of the query reply for 30 and 60 query nodes. As before, we select a stationary AoI that matches a cell. We then move the AoI – keeping its size unchanged – to cross one or four cell

boundaries (labelled “1 c bdry” and “4 c bdries”). The results show virtually no increase in the message overhead, nor a decrease in the recent sensor coverage. Now, we increase the size of the AoI to four times the size of the original, and to the entire sensor field. Not surprisingly, the quality of the query reply decreases, while the message overhead increases.

*Varying AoI size.* Clearly, the size of the AoI has an impact on sensor coverage. Common sense dictates that sensor coverage decreases with increasing AoI size. Fig. 9(b) shows the impact of the AoI size on sensor coverage. Here, we consider a single query node moving in a straight line together with the AoI. The AoI radius varies between 200m and 600m. In general, no particular AoI size performs significantly better than the others. The reason for that is that even smaller AoIs may require the query node to subscribe to multiple brokers, and a larger AoI does not necessarily trigger subscriptions to more brokers. Thus, increasing the AoI has only minor impact on sensor coverage. Notice that in the beginning and the end of the execution, the runs with large AoI may not consider significantly more sensors, as part of the AoI is outside the sensor field (the query node is only 200m, resp. 300m, from the away from the field boundary).

## 6 Related Work

Our work draws from a large body of earlier work:

In the context of sensor networks, many approaches exist. We have already presented approaches based on in-network aggregation, e.g., [6, 9, 22, 24]. In [33], Zoumboulaski *et al.* propose the use of active rules to accelerate the detection of events occurring in the environment, and install actuators at the sensors, which trigger the sending of a notification if a particular event occurs. They briefly mention the use of a pub/sub infrastructure to communicate the message to the base station, but no details are given. Also, [33] does not look at scalability issues.

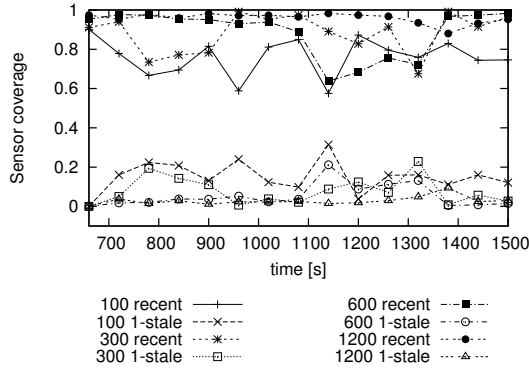
Directed Diffusion [16] can be seen as a publish-subscribe mechanism, which is implemented using the tree-based architecture rooted at the publisher. We have discussed tree-based architectures in Section 3.2 and explained why we did not consider them in our work. Moreover, our simulations run with a much higher number of query nodes.

The ACQUIRE mechanism [28] uses the concept of a mobile agent and sends the query through the sensor network, while acquiring more and more results on the way. This may result in high latencies and large query result messages.

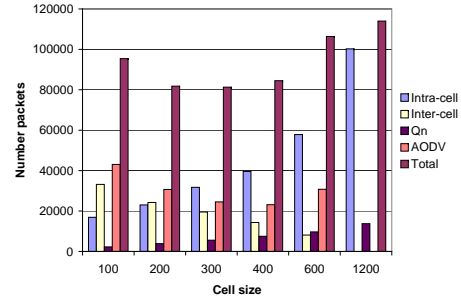
While these approaches all consider stationary query nodes, the following approaches address the issue of mobile query nodes.

In [13], Huang and Garcia-Molina explore different algorithms to construct routing trees rooted at the publishers. Again, the reader is referred to Section 3.2 for a discussion of tree-based architectures.

Mobile database access has been studied in [15]. However, Imielinski and Badrinath consider mobile users that ac-

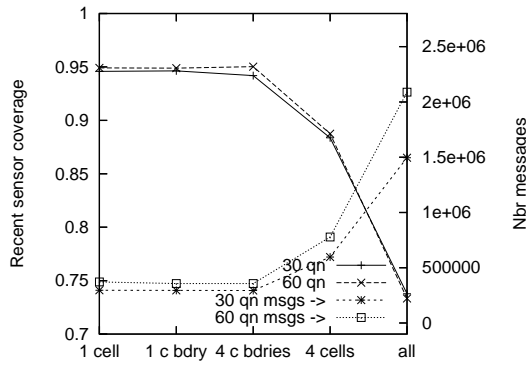


(a) AoI moves with qn, varying cell sizes.

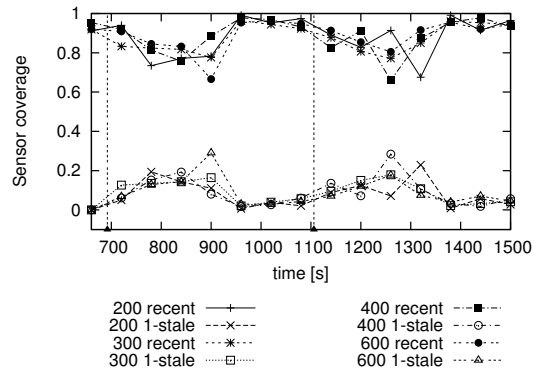


(b) Message overhead with varying cell sizes.

**Figure 7. Multiple query nodes sending queries for stationary AoI and a speed 1m/s.**



(a) Stationary AoI with varying size and position.



(b) AoI moves with the qn, varying sizes.

**Figure 9. Multiple query nodes sending queries for stationary AoI and a speed 1m/s.**

cess databases which are interconnected by a fixed network. Hence, their model is very different from the model considered in our work.

Tong *et al.* [29] suggest that sensor networks be extended with mobile agents, which move into proximity of the sensors to collect their results. Our approach is different, as we rely on routing within the sensor network to obtain the query results. Our query nodes cannot move into proximity of all sensors in which they are interested in order to collect the results.

Pub/sub has become a mature technology in fixed networks. Research on pub/sub in mobile ad hoc networks is more recent and has mainly focused on mobile subscribers and publishers relying on a fixed broker infrastructure to support them. The focus of our work, in contrast, is not to come up with a new pub/sub implementation for ad-hoc networks. Rather, we show that pub/sub can effectively support querying sensor networks. Our contribution is thus the mapping and the application of pub/sub to querying applications. Most existing pub/sub systems provide generic pub/sub solutions and they do not consider the particular mapping to query applications, such as for instance the tight dependence on geographic regions. With this orientation, they naturally focus on how to forward notifications to the subscriber once the subscriber

has moved (as suggested for instance in [21] and [7]). Nevertheless, our approach could also be implemented over existing pub/sub middleware, although our approach is extremely light-weight. Moreover, all subscriptions time out after a short time; as a consequence, no unsubscribe method is provided.

In [21], Siena is extended to handle mobile subscribers and publishers. Caporuscio *et al.* add a so-called *move-out* and *move-in* proxy to the subscriber to handle the latter's mobility. These proxies make sure that the notifications are forwarded to the subscriber's new location. Similarly, Fiege *et al.* [7] propose to rely on the brokers for handling mobility, rendering mobility issues related to pub/sub as transparent as possible to the subscriber. In our setting, these approaches are too costly given that the usefulness of an update expires after some time and leader-to-leader communication involves many intermediate nodes.

Yoneki and Bacon [31] have implemented pub/sub over mobile ad hoc networks. They assume that also the brokers are mobile and thus use ODMRP [19] to distribute the subscriptions to the brokers. This is costly, as it involves flooding. Moreover, it suffers from the other disadvantages discussed in Section 3.2. In our approach, sensors are mostly stationary and thus we can avoid flooding huge areas of the network.

In [18], Kim et al. propose an approach to route messages to mobile subscribers (called sink in [18]), based on intermediary accessor nodes. These accessor nodes are defined by the sink. The protocol corresponds to the approach presented in Fig. 3b(ii) and requires that a routing path be defined between the sink and the source. As they may be quite distant, setting this routing path up is costly, as it requires first to know the source and then to discover a routing path to it. This also increases the size of the routing tables, as they need to be aware of more entries.

In contrast, we do not rely on accessor nodes. While our brokers may be seen as sort of accessor nodes, they remain static and do not change in order to prevent frequent reconfigurations. Moreover, the subscribers only need to be aware of the local leader. Hence, finding sensors is an easier and more efficient operation. Finally, [18] does not consider the actual application, but just looks at a way to route the messages from a source to the mobile sink.

In recent years, many routing protocols based on hierarchical routing have been proposed [10, 11, 12]. While these routing protocols generally rely on clusters that are set up dynamically, our routing depends on a well-defined grid to form clusters. This clustering is driven by the query applications we are considering.

## 7 Conclusion

The paper has presented a natural and scalable way to query sensor networks from mobile platforms. We propose a layered approach in which the query application is mapped onto a pub/sub system. In a first step, we map queries to topics, and then show an architecture for pub/sub that is efficient in the setting we consider.

We have implemented the proposed architecture on the JiST/SWANS network simulator and have measured various aspects of our simulation. Our measurement results show the scalability of SENSTRAC with respect to the number of query nodes and its flexibility with respect to the AoI's position.

Although this paper limited itself to simulation, a merit of the JiST/SWANS technology is that the simulated code is executable on real platforms with only minor modifications. Accordingly, in our future work we hope to begin real experiments using actual sensors and query nodes.

## Acknowledgments

The authors are very grateful to Rimon Barr for his help on JiST/SWANS, Robbert van Renesse for many discussions on routing in sensor networks and on pub/sub, and Rohan Murty for his implementation of MAODV.

## References

- [1] JiST/SWANS. <http://jist.ece.cs.cornell.edu>.
- [2] The network simulator - ns-2. <http://www.isi.edu/nsnam/ns>.
- [3] D. Allen. Hidden terminal problems in wireless LAN's. In *IEEE 802.11 Working Group Papers*, 1993.
- [4] O. Babaoglu and K. Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In S. Mullender, editor, *Distributed Systems*, chapter 4, pages 55–96. Addison-Wesley, Reading, Massachusetts, USA, second edition, 1993.
- [5] R. Barr. *An efficient, unifying approach to simulation using virtual machines*. PhD thesis, Cornell University, Ithaca, NY, 14853, May 2004.
- [6] P. Bonnet, J. Gehrke, and P. Seshadi. Towards sensor database systems. In *Proc. of the Conference on Mobile Data Management*, Jan. 2001.
- [7] L. Fiege, F. Gärtner, O. Karsten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Proc. of the ACM/IFIP/USENIX Int. Middleware Conference*, number LNCS 2762, pages 44–63. Springer Verlag, June 2003.
- [8] M. Hapner, R. Sharma, J. Fialli, and K. Stout. *JMS specification*. Sun Microsystems Inc., USA, 1.1 edition, April 2002. <http://java.sun.com/products/jms/docs.html>.
- [9] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In *SOSP'01*, pages 146–159, Banff, Alberta, CN, 2001.
- [10] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS'00*, Jan. 2000.
- [11] X. Hong, L. Ma, and M. Gerla. Multiple-landmark routing for large groups in ad hoc networks. In *Proc. of IEEE Military Communications Conferences (MILCOM 2002)*, Anaheim, CA, Oct. 2002.
- [12] X. Hong, G. Pei, and M. Gerla. Landmark routing for large ad hoc wireless networks. In *Proc. of IEEE Global Communications Conference (GLOBECOM 2000)*, San Francisco, CA, Nov. 2000.
- [13] Y. Huang and H. Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *Proc. of the 4th Int. Conference on Mobile Data Management (MDM'03)*, pages 122–140, London, UK, 2003. Springer-Verlag.
- [14] IEEE. 802.11 specification (part 11): Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, June 1997.
- [15] T. Imielinski and B. Badrinath. Querying in highly mobile distributed environments. In *Proc. of the 18th Int. Conference on Very Large Data Bases*, pages 41–52, Vancouver, Canada, 1992. Morgan Kaufmann Publishers Inc.
- [16] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. of the 6th Int. Conf. on Mobile Computing and Networking (MobiCOM'00)*, Boston, MA, Aug. 2000.
- [17] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.
- [18] H. Kim, T. Abdelzaher, and W. Kwon. Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks. In *Proc. of 1st Int. Conf. on Embedded Networked Sensor Systems (SenSys'03)*, pages 193–204, New York, NY, USA, 2003. ACM Press.
- [19] S. Lee, M. Gerla, and C. Chiang. On-demand multicast routing protocol. In *Proc. IEEE Wireless Communications and Networking Conference, 1999 (WCNC. 1999)*, volume 3, pages 1298–1302, 1999.
- [20] J. Li, J. Jannotti, D. D. Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proc. of*

- the 6th Int. Conference on Mobile Computing and Networking*, pages 120–130, Aug. 2000.
- [21] A. W. M. Caporuscio, A. Carzaniga. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Transactions on Software Engineering*, 29(12):1059–1071, Dec. 2003.
  - [22] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of 5th Symp. on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, Dec. 2002.
  - [23] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *Proc. of the 2nd Int. Conference on Embedded Networked Sensor Systems (SenSys'04)*, pages 50–61, Baltimore, MD, Nov. 2004.
  - [24] S. Nath, P. Gibbons, and S. Seshan. Synopsis diffusion for robust aggregation in sensor networks. In *Proc. of the 2nd Int. Conference on Embedded Networked Sensor Systems*, Baltimore, MD, 2004.
  - [25] B. Oki, M. Pflügl, A. Siegel, and D. Skeen. The information bus - an architecture for extensible distributed systems. In *SOSP'93*, pages 58–68, Asheville, NC, 1993.
  - [26] C. Perkins and E. Royer. Ad-Hoc On Demand Distance Vector Routing. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, Feb. 1999.
  - [27] E. Royer and C. Perkins. Multicast Operation of the Ad-hoc On-demand Distance Vector Routing Protocol. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile Computing and Networking (MobiCom'99)*, pages 207–218, Aug. 1999.
  - [28] N. Sadagopan, B. Krishnamachari, and A. Helmy. The ACQUIRE mechanism for efficient querying in sensor networks. In *Proc. of the IEEE Int. Workshop on Sensor Network Protocols and Applications (SNPA'03)*, Anchorage, Alaska, May 2003.
  - [29] L. Tong, Q. Zhao, and S. Adireddy. Sensor networks with mobile agents. In *Military Communications Conference (MILCOM'03)*, volume 1, pages 688–693. IEEE, Oct. 2003.
  - [30] P. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *Proc. of the Symp. on Communications Architectures and Protocols*, pages 35–42, Stanford, CA, United States, 1988. ACM Press.
  - [31] E. Yoneki and J. Bacon. An adaptive approach to content-based subscription in mobile ad hoc networks. In *Proc. of the 2nd Conference on Pervasive Computing and Communications Workshops*, pages 92–97, 2004.
  - [32] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *INFOCOM 2003*, Apr. 2003.
  - [33] M. Zoumboulakis, G. Roussos, and A. Poulouvasilis. Active rules for wireless networks of sensors and actuators. In *Proc. of the 2nd Int. Conference on Embedded Networked Sensor Systems (SenSys'04)*, pages 263–264, Baltimore, MD, Nov. 2004.