An Area-Based Overlay Architecture for Scalable Integration of Sensor Networks

Lampros Pappas[1] and Spyros Lalis[1,2]

[1] Center for Research & Technology Thessaly
Technology Park of Thessaly
1st Industrial Area, 38500, Volos, Greece
lampros.pappas@gmail.com
[2] Dept. Computer & Communication Eng.
University of Thessaly
Glavani 37, 38221 Volos, Greece
lalis@inf.uth.gr

Abstract. With many different sensor networks being deployed, there will be an increased need to facilitate their uniform and efficient access at a large, perhaps even global, scale. To this end, we present an overlay-based architecture for organizing and querying multiple sensor networks based on their geographical area. The nodes of the system, representing query processors and individual sensor network gateways, are organized in a hierarchy which is used for query forwarding and result delivery. The key features of our system are: (i) support for the dynamic addition and removal of query processors and sensor network gateways; (ii) automatic hierarchy construction and awareness of sensing capabilities based on explicit metadata information; and (iii) efficient query multiplexing and result de-multiplexing within the overlay. We present a first evaluation of the proposed architecture. Results indicate that our design considerably reduces the communication between the nodes of the overlay as well as the actual sensing load at the edges (sensor networks) of the system.

1 Introduction

In the last years many wireless sensor networking platforms have been developed and deployed for various purposes, such as weather condition probing, smart agriculture, pollution measurement, surveillance of areas, and health monitoring. As sensing and sensor networking technology becomes more mature, numerous such installations will emerge in different places. One can imagine a couple of sensor networks deployed in each block, tens or hundreds in a medium-sized city, up to thousands in a region, not to mention an entire state or country.

Each sensor network will probably be deployed for a specific application in mind, and probably be operated by a different authority. Nevertheless, it could turn out to be useful for a wider (open) group of applications; some of them may be conceived long after the sensor network is deployed. It could be in fact very beneficial to let clients access *multiple* individual sensor networks in a transparent fashion. In this case, the seamless integration of distinct sensor networks and their unified access, at a large scale, is of major importance. Providers should be able to make the deployed sensor networks available to others, if desired, in a straightforward way. Clients wishing to retrieve sensor data for a given domain should be able to do this without being aware of every possible subsystem or having to deal with its internals.

To this end, this paper presents an overlay-based architecture for integrating a large number of sensor networks that cover different geographical areas in a unified infrastructure which can be efficiently queried from any computing device with Internet connectivity. Each one of the available sensor networks is represented by a gateway node which is responsible for intercepting queries, forwarding them to the actual sensor network, and sending back the results. Individual sensor networks can join and leave in a dynamic fashion, without disrupting the operation of the infrastructure. The system self-organizes itself into a hierarchy according to the area and sensing metadata of each node. Special focus is on supporting long-lived queries with wide area coverage, by transparently distributing and (de)multiplexing queries and results, taking into account the available sensor networks and sensing capabilities.

The rest of the paper is structured as follows. Section 2 gives an overview of related work. Section 3, 4 and 5 presents the system architecture, the formation process of the hierarchical peer overlay, and the management of queries, respectively. Section 6 provides a first evaluation of the proposed design via simulations. Finally, Section 7 concludes the paper.

2 Related Work

Several gateway systems have been proposed for making sensor devices and systems accessible through the Internet. VIPBridge [1] is a platform through which users can send queries to many, distinctive sensor networks. The idea is to map each sensor to a unique IPv6 address via a bridge-component that is aware of the number and type of available sensors. Every time an application needs data (or meta-data) from a sensor, the respective query is sent at the IP level; it is then received by the bridge which transforms it and forwards it to the target sensor through the proper (legacy) protocol. The results produced by sensor devices follow the reverse route. VIPBridge and other similar projects [2, 3, 4, 5] focus on the ability to access individual sensor nodes or an entire sensor network in a transparent way, but do not provide sufficient support for the scalable integration and querying of many different sensor networks.

A more advanced approach is presented [6] where services from different sensor networks can be accessed and combined via JXTA [7]. Specifically, JXTA is used to form a network of P2P nodes which act as a bridge for a specific sensor network, declaring the corresponding attributes in a JXTA advertisement. Sensor data is exchanged, merged and filtered amongst the nodes through JXTA messages. Client applications access the available sensing services through a UPnP [8] Gateway (a UPnP proxy is created for each service). Location-based queries are supported, but the organization of the P2P overlay is not driven by their location. Node discovery and message routing are conducted by the JXTA middleware in a location-agnostic way, thus it is not possible to forward/merge queries and reuse results based on the actual physical location of the sensor networks that participate in the system.

Recently, there are efforts for integrating sensors into grid computing applications [9], mainly focusing on making special equipment remotely available. SPRING [10] integrates wireless sensor networks with grid computing by using proxies as interfaces between the sensor networks and the grid, but does not address the issue of scalability and transparent access for a large number of individual sensor networks. In general, grid-related work is concerned with the efficient processing of large amounts of sensor data, regardless of the geographical area they were produced.

Microsoft's SenseWeb [11] is a platform that features mechanisms for storing sensor data, processing queries, aggregating and presenting results through easy-to-use web-interfaces. The system consists of four main components: the GeoDB sensor index, the DataHub data publishing toolkit, the IconD aggregator and the client-side GUI. GeoDB is a geographically indexed database for sensor metadata descriptions. DataHub is the gateway component for a concrete sensor subsystem, providing metadata to the GeoDB database as well as the actual sensor data in response to user queries. User queries are issued through a web-page [12] to IconD, the system's query processor and aggregator. Given a query's attributes, IconD queries GeoDB for sensor metadata and then the appropriate DataHub services to get the actual sensor data. When this becomes available, IconD properly aggregates it to create the result, which is finally displayed at the client interface. However, SenseWeb gathers and aggregates data *centrally* differing from the architecture we propose that provides *distributed* query dissemination and data gathering.

The HiFi [13] project aims in the collection, aggregation and filtering of data produced by large fan-in systems. The proposed architecture can be used in scenarios where organizations with hierarchical structure need to process large amounts of data produced in their edges. The major HiFi components are the Meta Data Repository, a globally accessible registry, the Data Stream Processor which is responsible for the data stream processing on each node (and may be implemented using any stream processing technology - TelegraphCQ stream query processor [14] is used in the first system prototype), and the HiFi Glue, which runs on each node and seamlessly binds it to the rest of the system. A prototype has been implemented based on the TinyDB sensor database system [15].

IrisNet [16] is a distributed architecture enabling convenient deployment of wide area sensing services, collecting data from heterogeneous sensor networks. IrisNet comprises of so-called sensing agents (SAs) and organizing agents (OAs). All agents run on Internet-connected PCs. Each SA is connected to one or more sensing devices and provides a common runtime environment for the running services, to share and filter the sensors' data. SAs are easily programmable, making possible to install simple code for filtering or storing sensor data and to add support for special hardware. Each OA holds a database, storing sensor data from various SAs in XML format. In turn, OAs are organized in groups and combine their local database into a distributed one, dedicated to a specific service. A location-based hierarchy is used to efficiently access the group's database, also supporting data replication and migration. Client queries are processed via XPath and other XML technologies.

The Global Sensor Networks (GSN) project [17] introduces a middleware which supports flexible integration and discovery of sensor networks and provides distributed querying, aggregation and combination of sensor data. The key entity in GSN is the so-called virtual sensor which abstracts a data stream coming from an actual sensor or other virtual sensors. Sensor network owners create virtual sensor descriptions (in the form of key-value pairs) and publish them to a directory so that virtual sensors can be discovered and contacted based on any combination of their properties, for example, geographical location or sensor type. GSN nodes are organized in a peer-to-peer network targeting greater scalability.

Hourglass [18] also addresses the need for rapid development and deployment of applications that consume data from multiple, heterogeneous sensor networks. The concept of a so-called circuit is used to link a set of data producers, a data consumer, and in-network services into a data flow. Control messages are used to set up the sensor data channels that travel over multiple services. Data provision, consumption and processing is abstracted in the form of services, which must be implemented to support a minimum functionality in terms of registration and circuit management.

Our work shares some of the design features of Hifi, IrisNet, GSN and Hourglass, which also introduce peer to peer architectures for distributed collection and aggregation of data from different sensor networks. However, although these systems do support location-based queries, the node hierarchy is either not related to location or is statically defined a priori. On the contrary, our system allows the node hierarchy to be formed dynamically as a function of the location information provided by each subsystem, and adapts query processing in order to exploit newly appearing sensor networks, also for already running queries. We discuss the differences between these projects and the system we propose in Section 6.

3 System Overview

We have developed a prototype system that enables the straightforward integration of several different sensing subsystems covering different areas, in order to support applications that wish to continuously receive sensor values for a wide area and a long period of time. This section gives an overview of the system, focusing on the main design aspects and describing the core elements as well as the interaction between client applications and the system.

3.1 System Elements

The core elements of the system, implemented in Java, are: (i) the *Registry*; (ii) the *Query Processor* (QP); (iii) the *Sensor Network Gateway* (SNG); and (iv) the *Client Front-End* (CFE). An indicative system configuration is shown in Figure 1.

The Registry manages the formation of the query processing overlay, deciding about the position of each QP in the hierarchy and keeping track of the respective parent-child relationships. Query Processors contact the Registry when they wish to join or leave the system, and Client Front-Ends contact the Registry to find out which QP should be used to submit a query to the system, based on the area concerned. The Registry is also contacted if a Query Processor is suspected to have failed in order to adjust the overlay as needed.

Query Processors are responsible for handling queries for a particular area, i.e., for a subset of the system's hierarchical name space. A QP accepts queries from one or more clients (via the Client Front-End) or other QPs, hands them over to an actual

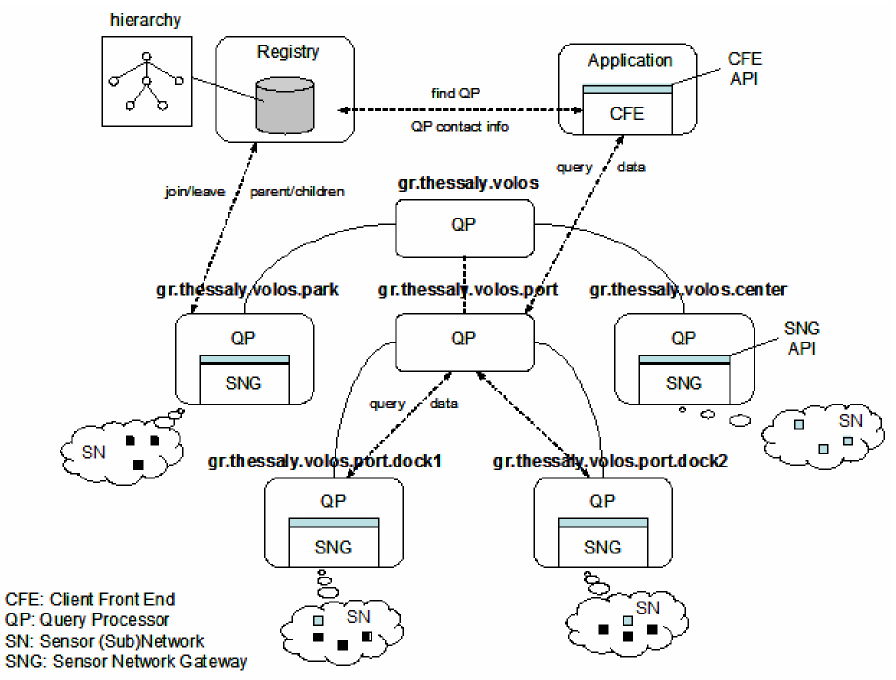

Fig. 1. Indicative system configuration

```
List<String> getSensorTypes();
String getArea();
int submitQuery (String sensorType,
                 String aggrOp,
                 int samplingPeriod);
List<Data> getQueryData(int queryID);
void cancelQuery(int queryID);
```

Fig. 2. Key primitives of the Sensor Network Gateway interface

```
int postQuery(String queryString);
List<Data> getResults (int queryID);
void cancelQuery(int queryID);
List<String> getAvlAreas(String areaOfInterest);
List<String> getAvlSensors(String areaOfInterest);
```

Fig. 3. Key primitives of the Client Front-End interface

sensor network subsystem and/or forwards them to other QPs that cover a part of the area concerned. When results become available, it sends them back to the clients and QPs that issued the respective queries. The QP is implemented in Java and interacts with the sensor network via a Sensor Network Gateway component (SNG) which performs the necessary communication and protocol/data conversion.

The Sensor Network Gateway (SNG) is co-located with the QP on the same machine and is invoked via a well-defined interface (an excerpt is given in Figure 2). This component provides abstract query submission and result delivery functions that are independent of the technology used to implement a sensor network. Its role is to mediate between a concrete sensor network and a QP that represents it to the rest of the system. In order to integrate a sensor network in the system, a class implementing the SNG interface must be developed, which is responsible for (i) issuing queries to the actual sensor network or sensing device, (ii) gathering the data produced by it, and (iii) temporarily storing this data until the QP component retrieves them. Obviously, the component that implements the SNG functionality must be able to communicate with the sensor network or device driver via the appropriate protocol stack or driver, while performing all the necessary protocol and data conversions.

Finally, clients issue queries to the system and receive results via the Client Front-End. The CFE is a Java component which can be included in a conventional Java program (an excerpt of the interface is given in Figure 3). It submits client queries to the Query Processor suggested by the Registry and receives the corresponding results.

3.2 Area Names

The query processing overlay is organized based on the area covered by each sensor network subsystem. Areas are explicitly specified in the form of DNS-like names, denoting a path with reference to a hierarchically structured name space. For example, *eu.gr.thessaly.volos.port* can be used to denote the port area in the city of Volos in the region of Thessaly in Greece in Europe. When joining the system, each Query Processor must provide the Registry with the name of the area for which it provides data. Similarly, a client that issues a query to the Client Front-End must specify the area of interest in order for the CFE to find (via the Registry) the most appropriate QP for submitting the query.

The use of such names simplifies the self-organization of QPs in the form of a tree structure and allows for a natural and simple inference of coverage relationships. For example if a QP registers for *eu.gr.thessaly.volos.port* and another QP registers for *eu.gr.thessaly.volos.port.dock1*, the latter becomes a child of the former. Also, if two QPs register for *eu.gr.thessaly.volos.port.dock1* and *eu.gr.thessaly.volos.port.dock2*, respectively, both of them will be considered if a client asks for data in the area of *eu.gr.thessaly.volos.port* or *eu.gr.thessaly.volos* etc.

It is important to note that the name hierarchy is *not* a priori specified. It *emerges* as a side-effect of the names employed by the participating QPs. Of course, QPs may use “incompatible” names, giving rise to different sub-domains, even though they cover the same area. For example, one QP could register for *eu.gr.thessaly.volos.port* while another QP could register for *eu.gr.thessaly.volos.harbor*. This cannot be avoided unless there is an authority (or hierarchy of authorities) responsible for assigning names to QPs. Our design does not preclude the existence of such authorities but neither imposes it. It is also possible for two QPs to register for the same area name, in which case it is simply assumed that they cover the same area.

3.3 Sensor Types

Each Query Processor informs the Registry about the sensor types that are supported for the area covered. This information is updated dynamically, also within the overlay, as Query Processors join or leave the system. As a consequence, a parent QP is aware of the sensor types supported by its children. This information is exploited to forward queries in a targeted fashion, only to the Query Processors which cannot provide data for the desired type of sensor.

The names used to specify sensor types are ASCII strings. Just as with area names, there are no predefined sensor types. Hence each QP may register for any sensor type name(s) / ASCII string(s) it desires. While it is desirable to use the same names for the same type of sensors, enforcing a common type system or ontology is beyond the scope of our work.

3.4 Query Submission and Reception of Results

Queries are submitted to the system via the *submitQuery* method of the Client Front-End (see Figure 3). The querying language is intentionally kept modest, designed to support applications that wish to continuously receive sensor values for a particular area of interest. In essence, queries are of the following form:

```
return  sensor data of type <SensorType>,
        aggregated using operator <AggregatorOp>,
        from area <AreaName>,
        at a sampling period <SamplingPeriod>,
for the next <Lifetime> minutes
```

Apart from the type of the sensor data to be retrieved, clients must specify the aggregation (avg, sum, min, max, void) that should be performed on the data produced by the participating sensing subsystems. Aggregation is performed only at the level of individual sensor networks, but not for values produced by different subsystems. This enables the application to issue a single query for a (very) large area yet receive a separate value for each of the sensor networks that can provide data for various (overlapping or disjoint) sub-areas. If desired, the application can combine these values to produce a higher-level (also in terms of semantics) aggregate. If the aggregation parameter is void, the system will return distinct values for every single sensor of every sensor network for the area of interest.

The area parameter denotes the area from which sensor data need to be retrieved. The application must also specify the period at which to sample the underlying sensor networks. This setting affects both the rate at which data will be produced and shipped back to the client but also the monitoring "granularity". Notably, the traffic between the sensor network subsystems and the client could be drastically reduced by adopting an event-oriented approach, e.g., by allowing the client to submit a data filtering/processing expression in order to receive sensor data in a more refined way. We do not to support this because we focus on applications that wish to continuously receive data rather than just be notified when an "event of interest" occurs. Needless to say, this makes support for query/result (de)multiplexing even more important.

Finally, each query is also given a lifetime. When the lifetime of a query expires, it is removed (garbage-collected) from the system. Client queries can be long-lived, having a lifetime of days or months, depending on the nature and requirements of the application.

When a client issues a query the Client Front-End contacts the Registry to find the Query Processor responsible for the specified sensor type and area of interest. If no matching QP exists, the query is rejected and an error code is returned. Else, the query is submitted to the QP suggested by the Registry. Once a query has been successfully submitted, the client may invoke the *getResults* method of the CFE to retrieve data as it arrives (blocking can be avoided by using a separate thread for this). The results are returned in the form of a list, each entry holding a distinct (aggregate) sensor value along with the area name of the subsystem that produced it. The client may cancel a query at any point in time via the *cancelQuery* method. A query remains active for its specified lifetime, unless it is explicitly cancelled. This allows a client to disconnect from and re-connect to the system in a seamless fashion, making it possible to tolerate connectivity problems. Currently, results are buffered until an (internal) threshold is reached with new results overwriting older ones; in principle it would be possible for the client to specify the desired buffer space and replacement policy for each query.

4 Maintenance of the Overlay

The hierarchical overlay of Query Processors is maintained as QPs join and leave the system or fail. All decisions regarding the position of each QP in the tree structure are taken by the Registry, which has a global view of the system (knows which QPs are registered for what areas). Even though QPs communicate in a peer-to-peer fashion, as far as overlay management tasks are concerned, they act exactly as instructed by the Registry, informing it in case they fail to proceed as required.

4.1 A Query Processor Joins the System

The joining Query Processor contacts the Registry, providing information about the sensor types supported and the area covered. Based on the area name supplied by the joining QP and the names used by the registered QPs, the Registry decides which QP should become the parent and which QPs (if any) should become the children of the joining QP, and sends a corresponding reply. When the joining QP receives the reply, it proceeds as follows. First, it sends an *ADD-CHILD* request to its parent, including its address, sensor types supported and the area covered, which in turn updates its children list and sends an acknowledgement. Then, the QP sends a *SWITCH-PARENT* request to each of its children, if any, with the same information. As a result, each child updates its parent information and sends an acknowledgement to the joining QP. It also sends a *REMOVE-CHILD* request to its old parent, which updates its children list. Finally, the joining QP informs the Registry that the process was completed successfully, and is ready to handle requests sent by clients and other QPs.

4.2 A Query Processor Leaves the System

When a QP wishes to leave the system, it contacts the Registry to inform it about its intended departure. The Registry replies suggesting a new parent for its children, if any. Upon receipt of the reply, the QP first sends a *REMOVE-CHILD* request to its parent, which updates its children list and sends an acknowledgement. Then, it sends a *SWITCH-PARENT* request for the new parent suggested by the Registry to each of its children, if any. As a result, each child updates its parent information and sends an acknowledgement. It also sends an *ADD-CHILD* request to its new parent, which in turn updates its list of children and sends an acknowledgement. Finally, the leaving QP informs the Registry that the process was completed successfully, and from that point onwards it is no longer part of the system.

4.3 A Query Processor Fails

A QP may fail or become unavailable for a long period of time. This will be eventually detected by its parent or its children when attempting to communicate with it. In this case, a *CHECK-FAULT* request is sent to the Registry, which double checks the problem. If the Registry decides to ignore the QP, it updates the overlay and notifies the QPs that are affected by this change. Specifically, it sends a *REMOVE-CHILD* request to the parent of the failed QP and a *SWITCH-PARENT* request along with a suggested new parent to each of the children of the failed QP. These requests are processed as usual.

Failures may also occur at various points during the join and leave protocol. If the joining QP does not receive an acknowledgement from its parent, it sends a *CHECK-FAULT* request to the Registry, and returns an error message that informs the sensor network owner to try and join later, when the hierarchy will be fixed. If it has successfully contacted its parent but cannot contact one of its children, it sends a *CHECK-FAULT* request to the Registry. If the Registry confirms that the child has failed, it sends a “switch parent” request to each of its children to be added as children of the joining QP. Else, the Registry sends an *ADD-CHILD* request to the joining QP for the child it just ignored. It is also possible that the joining QP itself fails in the midst of the join procedure. This will be detected either by a QP or by the Registry, and the overlay will be eventually “rolled back” to its old configuration. During the leave process, if the departing QP is not able to contact either its parent or one of its children, it sends a respective *CHECK-FAULT* request to the Registry and proceeds as usual. In other words, the leave process always terminates successfully without blocking the QP.

Last but not least, the Client Front-End may suspect that the Query Processor it used to submit a client query to the system has failed. It then sends a *CHECK-FAULT* request to the Registry and periodically inquires the Registry for a QP that can be used to re-submit the query.

4.4 Areas of Responsibility and Promotion of Query Processors

The Registry arranges the overlay so that a QP registered for an area name at the k^{th} level of the name space hierarchy will have as children QPs registered for various sub-areas at the $k+n^{th}$ level of the hierarchy, where $n>=1$.

The "natural" situation would be for n=1, i.e., for each QP registered for a given area to have as children QPs registered for direct sub-areas. For example, QP *eu.gr.thessaly.volos* could have as children QPs *eu.gr.thessaly.volos.port* and *eu.gr.thessaly.volos.park*. However, this is by no means guaranteed in our system. Not only are QPs free to choose the area names for which they register, but it is also quite likely that these names will only sparsely occupy the name space. As a consequence, it can be that n>1, i.e., a QP may have as children QPs registered for a sub-sub-area or sub-sub-sub-area of the parent. In this case, the parent is responsible for accepting queries targeted at any of the intermediate levels (for which there is no registered QP). The areas of responsibility of each QP are defined by the Registry and communicated to QPs as needed. For example, if QP registered for *eu.gr.thessaly* has as children QPs registered for *eu.gr.thessaly.volos.port* and *eu.gr.thessaly.volos.park* (i.e., there is no QP registered for *eu.gr.thessaly.volos*), it will also be responsible for accepting queries for *eu.gr.thessaly.volos*.

The absence of "intermediate" QPs in the name space, and thus in the overlay structure as well, may lead to situation where a QP registered for an area at the k^{th} level of the hierarchy has "too many" children for sub-areas at the $k+n^{th}$ level of the hierarchy, where n>1. To achieve better scalability, if the number of children at the level k+n becomes greater than a threshold, one of those children is "promoted" to act as a parent for the rest, and consequently becomes responsible for the corresponding area. For example, assume a promotion threshold of 3 and QP *eu.gr.thessaly* having 3 children registered for *eu.gr.thessaly.volos.port*, *eu.gr.thessaly.volos.center* and *eu.gr.thessaly.volos.park*. If a QP joins for *eu.gr.thessaly.volos.promenade*, then one of the 4 QPs registered for *eu.gr.thessaly.volos.** (in our implementation, it is the joining QP) will be promoted to be the parent for the other 3 and its area of responsibility will include *eu.gr.thessaly.volos*. Note that the promotion feature does not apply to children that already have a "direct" (in terms of name space) parent.

4.5 Examples

In Figure 4, join and leave examples are presented in order to illustrate the promotion and demotion scheme. Figure 4a displays the system's initial state. Assuming that the child threshold is equal to 3, when a QP registered for *eu.gr.thessaly.volos.promenade* joins the system, is is promoted (becoming responsible for area *eu.gr.thessaly.volos*) and QPs registered for *eu.gr.thessaly.volos.port*, *eu.gr.thessaly.volos.park* and *eu.gr.thessaly.volos.center* become its children (Figure 4b).

If the QP registered for *eu.gr.thessaly.volos.center* leaves the system, proper messages are sent and the QP registered for *eu.gr.thessaly* becomes again responsible for area *eu.gr.thessaly.volos*. The QPs registered for *eu.gr.thessaly.volos.port* and *eu.gr.thessaly.volos.park* become children of their former father (*eu.gr.thessaly*) and QP registered for *eu.gr.thessaly.volos.promenade* is properly demoted (Figure 4c). This happens because the indirect children-QPs of *eu.gr.thessaly* are again 3, just below the promotion threshold.

Notably, the QP registered for *eu.gr.thessaly.larissa* is not affected by these transformations of the overlay, because it is a direct (genuine) child of *eu.gr.thessaly*.

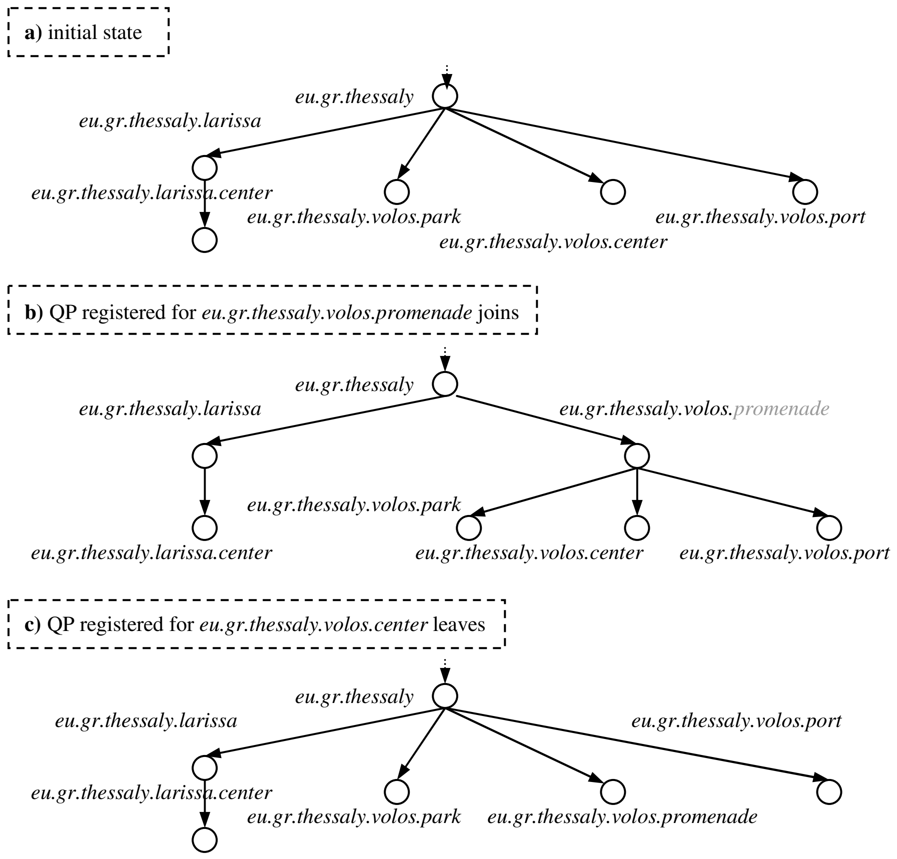
a) initial state
eu.gr.thessaly
eu.gr.thessaly.larissa
eu.gr.thessaly.larissa.center
eu.gr.thessaly.volos.park
eu.gr.thessaly.volos.center
eu.gr.thessaly.volos.port
b) QP registered for eu.gr.thessaly.volos.promenade joins
eu.gr.thessaly
eu.gr.thessaly.larissa
eu.gr.thessaly.volos.promenade
eu.gr.thessaly.volos.park
eu.gr.thessaly.larissa.center
eu.gr.thessaly.volos.center
eu.gr.thessaly.volos.port
c) QP registered for eu.gr.thessaly.volos.center leaves
eu.gr.thessaly
eu.gr.thessaly.larissa
eu.gr.thessaly.volos.port
eu.gr.thessaly.larissa.center
eu.gr.thessaly.volos.park
eu.gr.thessaly.volos.promenade

Fig. 4. Example scenario of Query Processors joining/leaving the system, triggering promotion and demotion of QPs in the hierarchy

5 Query Processing

Queries are processed based on the QP hierarchy, being submitted to the QP that is responsible for the area of interest. The hierarchy is exploited to forward queries to QPs that cover sub-areas. If possible, queries are multiplexed as they trickle down towards the individual sensor networks, and results are de-multiplexed as they travel back to the clients/QPs that issued the respective queries. The system also handles the dynamic addition and removal of QPs as well as the cancellation of queries.

5.1 Query Forwarding and Multiplexing

As already discussed, the Client Front-End submits client queries to the appropriate QP suggested by the Registry. When a QP receives a query and features a subsystem that supports the desired sensor type, it checks if there are similar queries (for same sensor type and aggregation operator) running locally. If such a query does not exist, the newly received query is submitted to the Sensor Network Gateway. Else, the new query is linked to that query in order to exploit the results being produced anyway; the SNG is invoked only if the new query has a shorter sampling period. In any case, the QP forwards the new query to any of its children that can provide data for the desired type of sensor (no multiplexing is attempted at this level). The reason for this is to be able to tolerate the failure of the QP that received the client query in a simple way, without forcing the Client Front-End to re-submit the query from scratch.

5.2 Result Delivery and De-multiplexing

Results travel along the reverse path, from each QP that features a sensing subsystem to the Client Front-End that submitted the query and/or the parent QP (if it forwarded the query). The transmission/delivery of results is asynchronous, depending on when they become available from the SNG or children QPs. The same data is transmitted only once for all linked queries, de-multiplexing it as needed for different CFEs.

5.3 Query Cancellation

A client may decide to cancel a query at any point in time. In this case, the Client Front-End sends a cancellation request to the Query Processor that is responsible for handling the query. Cancellations are forwarded to SNGs and other QPs in the same way this is done for queries. If the cancelled query is linked to other queries, it is removed; the SNGs are invoked only if the cancelled query had the shortest sampling period, which is set to the shortest sampling period of the remaining linked queries.

6 Evaluation

We have implemented a system prototype in Java, including an SNG for a real sensor network based on the Smart-Its platform [19], and tested it for various small configurations and client queries. We used the respective Java API in order to send and receive messages from the class that implements the SNG interface. More efficient SNG implementations, supporting other sensor platforms or sensor database systems like TinyDB, can be developed and merged transparently into our prototype.

To evaluate our system for a large scale, we developed a simple simulator used to perform several virtual experiments. The simulator parameters include the (i) number of QPs that participate in the system; (ii) the branch degree (the maximum number of sub-areas per area of the hierarchy); (iii) the probability of a QP participating for a sub-area; (iv) the promotion threshold; (v) the available sensor types; and (vi) the number of queries submitted.

In a first experiment we simulate the creation of the Query Processor overlay for 4000 QPs and a branch degree of 7, varying the probability of join for an area. The overlay is created in a top-down and breadth-first fashion, by considering each area of the hierarchy and deciding randomly (based on the join probability) whether a QP will register for that area, subject to the branch degree specified, until the specified number of QPs has been reached. In essence, the smaller the join probability is, the sparser the occupation of the name space is by the participating QPs. The experiment is performed with the promotion feature disabled and enabled (with a threshold of 7), measuring the maximum number of children of any QP in the resulting overlay. As shown in Figure 5, this number rises excessively as the name space occupation becomes sparser if promotion is not enabled. On the contrary, child promotion keeps the maximum number of children per parent to a rather steady and small value, independently of the occupation density of the name space.

In the second and third experiment we evaluate the benefit of the awareness regarding the sensor types supported by each QP and the query multiplexing support. The overlay is formed for 10000 QPs with a branch degree of 7 and a promotion threshold of 7, varying the join probability. Each QP is assumed to support up to 2 different sensor types, randomly chosen out of 8 possible values. Finally, 200’000 queries are submitted to the system, for randomly selected areas and sensor types, with the same sampling period and an infinite lifetime.

Figure 6 depicts the total number of queries forwarded and maintained at all QPs with and respectively without disseminating information about the sensor types supported by each QP. As it can be seen, the resources used are drastically reduced (roughly 80%) when parent QPs are aware of the sensor types supported by their children, grandchildren etc.

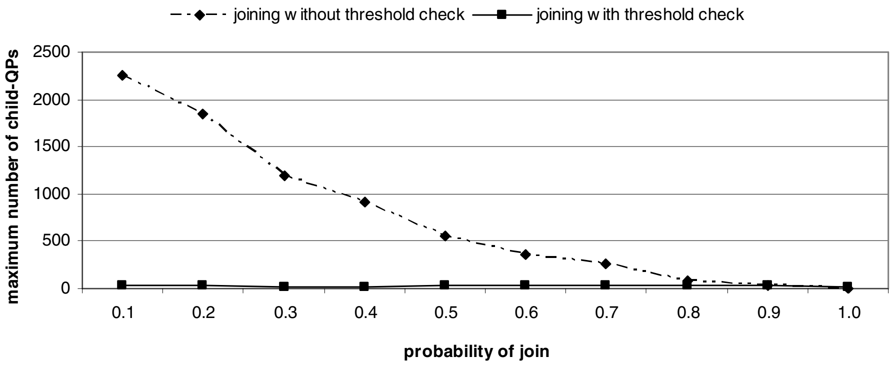

Fig. 5. Maximum number of child QPs with / without threshold check

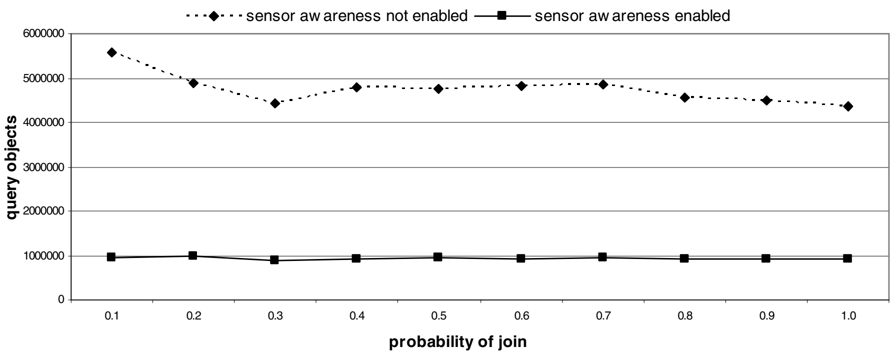

Fig. 6. Number of Query Objects with / without sensor type awareness

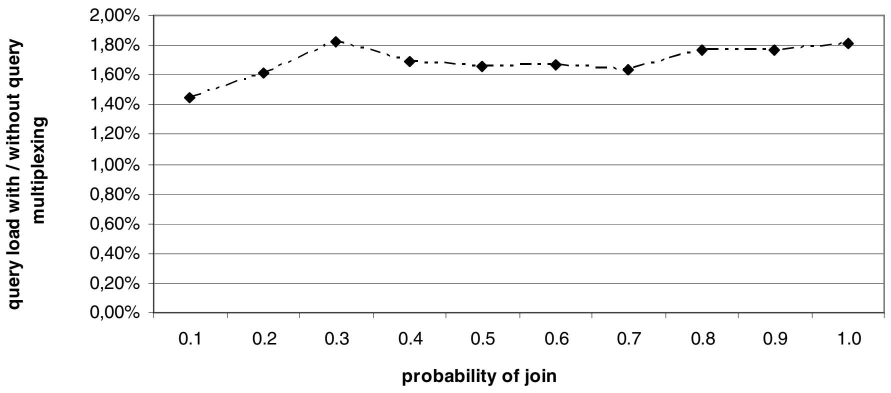

Fig. 7. Number of SNG Query Objects when query multiplexing is used to number of SNG Query Objects when query multiplexing is not used

Figure 7 plots the percentage of queries submitted to SNGs when multiplexing queries using as a reference the number of queries that would have been submitted in a system without multiplexing support. It is clear that query multiplexing significantly reduces the involvement of SNGs in providing results for the queries issued (by roughly 98%), which in turn translates to a reduced overhead/cost at each individual sensor network. Equally important, the larger the degree of multiplexing, the greater the amount of data that can be reused for different queries, and the smaller the amount of data that needs to travel back to QPs in a bottom-up fashion.

As we mentioned in Section 2, the system we propose shares some of the features of Hifi [13], IrisNet [16], GSN [17] and Hourglass [18], which also introduce distributed, overlay architectures for collection and aggregation of data from different sensor networks. However, there are some key differences amongst these architectures and our work, either in the goal or the implementation. Hifi nodes are organized in a fixed hierarchy according to the organization that they are deployed. On the contrary, we introduce a dynamic hierarchy, based on the nodes' declared locations. The children promotion scheme we described and evaluated avoids having nodes with excessive number of children, thereby eliminating possible congestion points. IrisNet is quite similar to our work as it focuses on location based queries, but in essence it is a distributed XML database. Data retrieved from the Sensing Agents are replicated to the Organizing Agents according to statistics held for the queries issued by user applications. Our system supports more efficient query multiplexing and sensor data reuse utilizing the constructed dynamic hierarchy and sensor type awareness, aiming at reducing the resource consumption at the edges of the system, i.e., the actual sensor networks. Hourglass and GSN are similar to Continuous Query systems with Hourglass focusing on maintaining quality of service in the presence of disconnections while GSN is more abstract and targets mainly on supporting flexible configurations. None of these systems facilitates the establishment of a hierarchy based on the sensor networks' location. Instead, the node hierarchy is statically defined and location is merely one of the query parameters used to route and filter queries within the system.

7 Conclusion

We have presented an overlay-based architecture for the straightforward integration and transparent querying of multiple sensor networks through the Internet, based on a structured hierarchical name space for denoting areas of coverage. The major innovation is the self-organization of the participating Query Processors in a suitable hierarchy, which in turn is used to support query forwarding and multiplexing towards the edges of the system where actual sensor network subsystems reside. Simulated experiments show that our approach scales nicely even for a large number of nodes and a sparse occupation of the name space. Our design supports the dynamic addition and removal of Query Processors and sensor subsystems, without interfering with the operation of the infrastructure and client applications using it.

The proposed architecture can be further enhanced and expanded. A distributed version of the self-organization algorithm could be pursued in order to perform adaptations of the overlay with a minimal or more asynchronous involvement of the Registry, which currently controls the entire process in a tightly coupled fashion. Of course, the failure handling protocol would have to be re-engineered correspondingly. Better availability and scalability could also be achieved by distributing the Registry. Finally, it would be interesting to apply the idea of self-organization with areas being specified in a more refined way, e.g., via circles or polygons with reference to a global coordinate system.

References

1. Lei, S., Xiaoling, W., Hui, X., Jie, Y., Cho, J., Lee, S.: Connecting Heterogeneous Sensor Networks with IP Based Wire/Wireless Networks. In: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA 2006), pp. 127–132 (2006)
2. Dunkels, A., Alonso, J., Voigt, T., Ritter, H., Schiller, J.: Connecting Wireless Sensornets with TCP/IP Networks. In: Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC 2004), Frankfurt (Oder), Germany (2004)
3. Dai, H., Han, R.: Unifying Micro Sensor Networks with the Internet via Overlay Networking. In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, pp. 571–572 (2004)
4. Marco, Z., Bhaskar, K.: Integrating Future Large-scale Wireless Sensor Networks with the Internet. USC Computer Science Technical Report CS 03-792 (2003)
5. Chen, M., Mao, S., Xiao, Y., Li, M., Leung, V.: IPSA: A Novel Architecture for Integrating IP and Sensor Networks. International Journal on Sensor Networks (IJSNet) 5(1), 48–57 (2009)
6. Isomura, M., Riedel, T., Decker, C., Beigl, M., Horiuchi, H.: Sharing sensor networks. In: Proceedings of 26th IEEE International Conference on Distributed Computing Systems Workshops, 2006. ICDCS Workshops, p. 61 (2006)
7. The JXTA technology community, `https://jxta.dev.java.net/`
8. The UPnP forum, `http://www.upnp.org/`

9. Bramley, R.: Instruments and sensors as network services: Making instruments first class members of the grid. Technical Report 588, Indiana University CS Department (2003)
10. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An Open Grid Services Architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum (2002)
11. Nath, S., Liu, J., Zhao, F.: Challenges in Building a Portal for Sensors World-Wide. In: First Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications (WSW 2006), Boulder CO (2006)
12. Microsoft's Sensormap, `http://atom.research.microsoft.com/sensewebv3/sensormap/`
13. Franklin, M.J., Jeffery, S.R., Krishnamurthy, S., Reiss, F., Rizvi, S., Wu, E., Cooper, O., Edakkunni, A., Hong, W.: Design Considerations for High Fan-in Systems: The HiFi Approach. In: CIDR 2005, pp. 290–304 (2005)
14. Chandrasekaran, S.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: CIDR 2003 (2003)
15. Madden, S., Hellerstein, J., Hong, W.: TinyDB: In-Network Query Processing in TinyOS. Release documentation, version 0.4 (2004)
16. The Iris Net documentation web page, `http://www.intel-iris.net/research.html`
17. Aberer, K., Hauswirth, M., Salehi, A.: The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical report LSIR-REPORT-2006-006 (2006)
18. Shneidman, J., Pietzuch, P., Ledlie, J., Roussopoulos, M., Seltzer, M., Welsh, M.: Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Harvard Technical Report TR-21-04 (2004)
19. The smart-its devices, `http://particle.teco.edu/`