Distributed In-network Actuation of Wireless Sensor Networks using a Declarative Query Interface

by
Asanka P. Sayakkara (*asanka.code@gmail.com*)
2008/CS/111

This dissertation is submitted to the University of Colombo School of Computing in partial fulfillment of the requirements for the degree of
Bachelor of Science (Computer Science)

University of **C**olombo **S**chool of **C**omputing,
No-35, Reid Avenue,
Colombo 07,
Sri Lanka.
http://www.ucsc.cmb.ac.lk
December 2012

Abstract

Declarative SQL queries are a way of abstracting out the underlying complexity in Wireless sensor-actuator networks (WSAN). Current SQL syntax to perform actuation tasks misleads the user because of their artificial way of expressing the actuation tasks thus violating the purpose of the declarative nature. Particularly with respect to distributed in-network actuation tasks where multiple sensor nodes and actuator nodes involve, the user may be required to write multiple queries based on the current actuation query syntax causing a negative impact on the performance. We address these issues in existing declarative layer with modifications to the virtual data table of the abstraction and thus a new syntax is suggested and implemented. We also introduce a new in-network execution strategy for suggested query syntax to address performance issues of distributed in-network actuation scenarios. Using prototype implementations, we provide evidence to prove that our solution addresses the identified issues of existing works.

"The dream is not what you see in sleep, dream is which does not let you sleep"

- Dr. Abdul Kalam
(famous scientist and a former president of India)

Acknowledgment

Since I emerged as a small child more than two decades ago I have gone through so many good and bad life experiences. Now as a university student at the final phase of my undergraduate life I realized how much I have improved upon those life experiences. So, at first I thank my parents, my relatives, my teachers and my friends who have added so many good things into my life to make me the person today I am. Without their life long support I would not come to this point and obviously this research work would not be here.

My two supervisors Dr. M.D.J.S Goonetillake and Dr. Kasun De Zoysa were the guiding lights of me in the research path I have been walking through for about a year. I learned the correct ways to lead a research work to the point it get published by working with them. Their experience and knowledge in the domain has helped me so much to make this work a success. Therefore I thank my supervisors for standing with me throughout the year to work on this research. Additionally Mr. Lakmal Weerawarne and Mr. Chathura Suduwella have contributed to my work by prompting different fresh ideas which have improved my understanding on the research problem and it's solution approach.

Our coordinators for the 4th year student research projects Mr. Dulan Wathugala and Mrs. Lasanthi De Silva have put a great effort to organize this main component of the final year students an unforgettable experience. They always monitored each of us to make sure we conduct our research work in a proper way. Eventually I believe their effort have generated so many good quality research works from the 4th year students in this year. I thank them for their effort and time they spent throughout the year to move us from student roles to researcher and scientist roles.

No research can stand alone without the contributions of the other people in the history who have spent their time on solving the early problems. My last thank goes to those all giants who made this world a better place by putting their creativity and innovation on solving problems throughout the human history.

Declaration

I, Asanka P. Sayakkara (Reg: 2008/CS/111) hereby certify that this dissertation entitled "Distributed In-network Actuation of Wireless Sensor Networks using a Declarative Query Interface" is entirely my own work and it has never been submitted nor is currently been submitted for any other degree.

Date	Students Signature

I, Dr. M. D. J. S. Goonetillake, certify that I supervised this dissertation entitled "Distributed In-network Actuation of Wireless Sensor Networks using a Declarative Query Interface" conducted by Asanka P. Sayakkara in partial fulfillment of the requirements for the degree of Bachelor of Science (Computer Science).

Date	Signature of Supervisor

Contents

Chapter 1

Introduction

Wireless sensor networks(WSN) [11] consists of small wireless enable embedded devices called motes in large numbers. They are used for tasks such as environmental monitoring and tactical surveillance[15, 22]. Motes are capable of taking readings from their built-in sensors and transmitting to a specified location for further processing and analyzing. By the nature of their applications, motes have to be low cost and should be able to run on battery power making them resource constrained in all aspects including memory capacity and CPU power. While motes are capable of sensing the environment, various real world applications raise the requirement of controlling the environment by performing actions based on sensor readings. To this end, actuators such as switches and regulators can be connected to motes to perform actuation tasks. Such networks are known as wireless sensor-actuator networks (WSAN)[1].

Due to hardware limitations of motes it is necessary to write low level application codes carefully managing the memory and saving battery power as much as possible for a longer service life. One approach of making the WSN users life easier is high-level programming abstractions which hide the low level hardware details. Database abstraction for WSNs [3] like Cougar [23] and TinyDB [13] are such abstractions where the whole network is represented as a virtual database table and each column of the table is mapped to a sensor type available in the network.

In such implementations, the data of the network is acquired by running SQL queries over the virtual table. SQL enables the user to specify which data to be acquired from the virtual table in a declarative way without bothering how the data is acquired. However with respect to complex sensing associated with actuating requirements, current declarative interface for sensor networks exhibit serious weaknesses making them unusable for such scenarios. Moreover, there's no easy or declarative mechanism to ob-

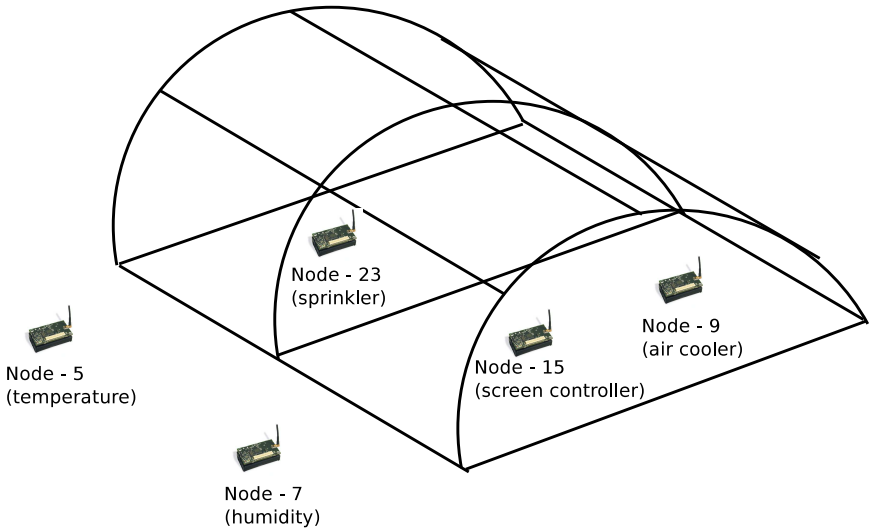

Figure 1.1: A greenhouse with multiple sensor and actuator nodes. The actuators include an air cooler, a screen controller and a sprinkler which are fixed inside the greenhouse while the sensors include a temperature sensor and a humidity sensor which are placed outside the greenhouse. Only the nodes relevant to the queries are given for simplicity.

tain meta level information on actuators in WSAN which is important for actuator monitoring purposes. We illustrate a scenario with respect to a greenhouse cultivation [4] for the purpose of pinpointing the weaknesses in the current approach.

1.1 An Actuation Scenario

In a greenhouse environment it is particularly crucial to manage the environment effectively by controlling the environmental parameters such as temperature, light, humidity, CO_2, ambient pressure and wind flow. There are actuators to control these parameters and these may include screen controllers/carton sliders to protect the cultivation from direct sunlight, heaters and air coolers to maintain humidity and temperature, fans to maintain the wind flow and also actuators to inject CO_2 to influence photosynthesis.

In order to obtain the real benefit of deploying WSAN in this context the end user should be able to monitor and thus to control the environmental parameters (which may subject to dynamic changes) through controlling relevant actuators accordingly. By the nature of the application, the network may have three types of motes which are motes with sensors only, motes with

actuators only and motes with both sensors and actuators.

Consider as such a scenario where a WSAN deployed for a greenhouse to monitor the condition of the field continuously and to perform actions when necessary. Assume that this network employs a TinyDB [13] like abstraction to monitor the network since TinyDB provides more facilities for WSAN. If climate sensors (say motes 5 and 7) fixed outside the greenhouse sense an outside temperature greater than 35 °C and humidity level greater than 45, the user requires to switch on an air cooler, a screen controller and a sprinkler attached to motes 9, 15 and 23 respectively which are fixed inside the greenhouse. Figure 1.1 depicts this greenhouse environment.

To check the relevant sensor readings of the relevant nodes, user has to send Query-1 on TinyDB abstraction layer. If this query generate results from motes 5 and 7, that means the sensors of our concern have the suitable values to perform the actuation. Based on that results, the user has to send actuation queries as shown in Query-2, 3 and 4 targeting at each node and their actuators accordingly.

Query-1

```
SELECT nodeid
FROM sensors
WHERE (nodeid=5 OR nodeid=7)
  AND temperature>35 AND humidity>45
ONCE;
```

Query-2

```
SELECT nodeid
FROM sensors
WHERE nodeid=9
OUTPUT ACTION air-cooler-on()
ONCE;
```

Query-3

```
SELECT nodeid
FROM sensors
WHERE nodeid=15
OUTPUT ACTION screen-ctrl-on()
ONCE;
```

Query-4

```
SELECT nodeid
FROM sensors
WHERE nodeid=23
OUTPUT ACTION sprinkler-on()
ONCE;
```

TinyDB's query language follows conventional SQL syntax to acquire data from the virtual database table with some additions to support exclusive requirements in sensor networks as explained in detail in the following section.

1.2 Problem Of Actuation Queries

We identify three main issues in current actuation queries in the declarative interface. Firstly when performing the actuation with existing queries, user has to write *SELECT* queries with the *OUTPUT ACTION* clause. Due to this new clause, the query does not return any data requested by the *SELECT* query and instead calls the low level function to perform the actuation. However when writing or reading *SELECT* queries, semantically it represents displaying of data relevant to the stated columns with respect to a database table. However, these actuation queries performed by enhanced *SELECT* queries are not intended to display or return any data thus confusing the query writer. This is due to the misuse of 'SELECT nodeid' when the exact node id is defined in the WHERE clause as in Query-2, 3 and 4. Therefore the user does not receive the true advantage of declarative interface when the syntax is in this type of an artificial way.

Secondly, when multiple sensors and actuators are to be involved in an actuation task as the given scenario, the user may have to write number of queries for checking different sensor conditions before actuating different actuators. This multiple query usage leads to time delays as human intervention is required to issue each query from a sequence of queries. Moreover if the user forgets/misses to issue a query from the corresponding query sequence the expected control outcome cannot be achieved. As such multiple query usage is not recommended if time delays between sensing and actuating is required to be as minimum as possible for time critical tasks. Additionally multiple queries results in more power usage of nodes reducing the service life of the network. These issues become worse when the number of nodes in the network grows in large numbers.

Finally since the existing data model of declarative interface does not provide an abstraction over actuators, they have to be activated by calling

low level functions. Therefore existing data model does not provide any easy mechanism to find the status of actuators if actuation tasks have to be done based on the current status of actuators.

1.3 Project Scope

The scope of this research is to fix two main issues identified in the declarative interface for wireless sensor-actuator networks. Those research problems are as follows.

1. Violation of declarative nature in existing actuation query syntax.
2. Inefficiency of performing actuation tasks using the existing query syntax.

In this research we address those issues in existing declarative layer with modifications to the virtual data table of the abstraction and with new syntax suggestions for actuation queries. We also introduce a new in-network execution strategy for suggested query syntax to address performance issues of distributed in-network actuation scenarios. We show that our suggested solution address the identified problems so that,

1. New syntax is more declarative than existing syntax.
2. New syntax can be efficiently used to perform actuation tasks than the existing syntax.

Chapter 2

Background and Related Works

Research on wireless sensor networks(WSN) [11] have been matured to a level where they can be a part of our day to day life. They have been deployed in industrial environments where reliability and real-time performance is a must. Recently traditional wireless sensor networks have integrated with a new functionality which has taken it to a new world which is full of new opportunities and at the same time new problems. That new functionality is the actuation. An actuator is a component which takes control commands as input and act on the deployed environment to make a change in it. WSNs integrated with actuators as nodes have created the new research domain widely known as Wireless sensor-actuator networks(WSAN) [1, 19].

In this chapter we discuss the existing WSAN related research particularly with the focus on actuation criteria. According to the scope of this research, our major concern is on declarative query interfaces to WSANs and therefore we categorize existing work into two different parts namely declarative interfaces and non-declarative interfaces to WSANs. We discuss about their functionality and compare and contrast them against different real world scenarios. We highlight gaps and pitfalls in existing works to identify the correct way to go.

2.1 Declarative query interfaces for WSAN

2.1.1 Advantage of declarative interface

With the introduction of the relational model[5] it became the ideal way of storing data in large amounts. There have been different languages introduced to express the criteria of querying data from these relational databases. Most of them are based on relational algebra and relational calculus. How-

ever finally Structured Query Language(SQL) which was previously known as SEQUEL became the de facto query language for relational databases which is simple, easy to learn and at the same time expressive enough to interact with relational databases[2]. The declarative nature of SQL made it possible for even ordinary people who are non-specialists in relational databases to interact with it easily.

Because of the underlying complexity of WSNs, researchers have considered the possibility of providing abstractions to ease the task of non-computer scientists to deploy and use WSNs. A major consideration among these works is the abstraction of WSNs as a database [3, 7]. In such database abstractions for WSNs, SQL unarguably became the query language of them because of the same features that helped it to be selected for relational databases.

2.1.2 Declarative interface for monitoring

There are several papers published pushing the idea of SQL database abstractions for sensor networks. One of the earliest such implementations is Cougar[23] which had declarative queries to acquire sensor data. The authors of Cougar pinpointed two main reasons which have motivated them to consider the possibility of SQL database abstractions for WSN environments. Firstly, the declarative queries are a very good way to interact with the WSN by the end users and user applications without knowing how the data is generated and processed within the network. Secondly they have noticed that in-network query processing saves energy in the network.

Cougar also came with in-network data aggregation and query optimization mechanisms which improve the functionality and advantages of SQL queries. For each query issued by the user level, the gateway node prepares a query plan which is optimized for better execution of the query. To realize in-network data aggregations, a leader node is used within the network according to the query plan for each query.

The introduction of TinyDB [13, 14] can be considered as a major achievement in the declarative query abstractions for WSNs. In the Cougar approach, it assumes the prior existent of sensor data on the nodes of the network. The queries of Cougar were applied to those previously retrieved data. When compared to Cougar, TinyDB introduced and implemented the concept of *acquisitional queries* where the queries are executed in the sensor nodes to acquire data from the sensors in real-time. Similar to Cougar, TinyDB views the WSN as a single database table with each sensor type mapped to a particular column. Tuples of the table are appended at real-time with a time gap between each two tuples.

TikiriDB [10] is another database abstraction layer for WSNs which fol-

lows a similar kind of approach to TinyDB. The major difference between TikiriDB and TinyDB lays on the support for multiple base stations in the former. Therefore WSNs that are shared between multiple users can be implemented using TikiriDBs SQL database abstraction. Syntaxs of queries provided in TikiriDB are almost similar to TinyDBs syntaxs.

In the following example we compare a conventional SQL query that we can find in a database management system(DBMS) like MySQL[18] with an acquisitional query. Consider a database table named as *sensors* in a conventional database. This table contains some environmental parameters taken previously and stored for a future use. The columns of the table are defined as *id*, *temperature* and *humidity* which all are integers. The *id* column is an auto increment field which gets incremented each time we insert a new tuple to the table. When we need to get the tuples from the table where temperature field has a value more than 25, we issue a SELECT query like the following.

Query-5

```
SELECT nodeid,temperature,humidity
FROM sensors
WHERE temperature > 25
```

The *SELECT* keyword specify the data fields which should be included in the result data set while the *FROM* keyword specify from which table the data should be retrieved. Finally the *WHERE* keyword specify the criteria to retrieve the data so that only the required data set will be retrieved from the table. The *SELECT* query in a conventional database works in this way.

Now lets consider a different scenario. There is a WSN deployed in a field where the nodes can measure temperature and humidity parameters of the environment. Each node is assigned an identification number to identify uniquely. This WSN has TinyDB installed on all the nodes and therefore the base station allows to issue SQL queries to acquire data. Say we need to acquire sensor readings from the sensor network in every second for a time period of 10 seconds but only if the temperature of the sensor readings are more than 25. The following query shows how we specify our criteria.

Query-6

```
SELECT nodeid,temperature,humidity
FROM sensors
WHERE temperature > 25
SAMPLE PERIOD 1s FOR 10s
```

The TinyDBs database model shows the whole WSN as a single database table named as *sensors*. In this model, all the sensor types in a node of the sensor network are mapped into the columns of the *sensors* table so that *id*, *temperature* and *humidity* become the columns. According to this model *SELECT*, *FROM* and *WHERE* keywords serve the same functionality of the Query-5. The *SAMPLE PERIOD* and *FOR* keywords are additions to the SQL syntax by TinyDB to specify how the sensor readings should be taken over the time. Here we have specified that we need readings in every second for a period of 10 seconds. Such a syntax is necessary since we are dealing with real-time data in the WSN. TinyDB provides various new syntaxes for data acquisition from the WSN which are defined in [13, 14].

2.1.3 Declarative interface for controlling

Even though Cougar provide queries to get sensor readings from a WSN, it does not provide any support to control the environment where the WSN is deployed using the same declarative queries. However TinyDB have added new keywords to its query language for controlling actuators. For example consider the following TinyDB query.

Query-7

```
SELECT nodeid,temperature
FROM sensors
WHERE temperature > 25
OUTPUT ACTION sprinkler-on()
SAMPLE PERIOD 10s
```

In this query *SELECT*, *FROM* and *WHERE* keywords are the same as previous queries. The *SAMPLE PERIOD* specifies that this query should be executed for every 10 seconds. The new keyword we find here is *OUTPUT ACTION* which can be used to specify a low level function. If there is some output data set after the execution of the query then the specified low level function gets called. In this particular scenario, if the temperature reading of a node is greater than 25 the mote calls the low level system function *sprinkler-on()*. Because of the presence of *OUTPUT ACTION* keyword in the query, the *SELECT* query prevents from sending the result set back to the base station.

The low level function which is given as the parameter to the *OUTPUT ACTION* keyword can be used to trigger an actuator device attached to the sensor mote. For example lets say that *sprinkler-on()* function can switch

on a sprinkler attached to a sensor node in the network. When the Query-7 is received to a node, it takes readings from its temperature sensor in every 10 seconds. If the value is greater than 25, the node calls its *sprinkler-on()* system function to switch on the sprinkler attached to the node. In this way we can use queries provided in TinyDB to control the environment. This case is an example for local in-network actuation since both sensor and actuator reside in the same node.

Even though a single query like the one above can be used in local in-network actuation scenarios, multiple complex queries have to be used in scenarios where distributed in-network actuation is necessary. To illustrate such queries of TinyDB consider the following scenario.

A WSAN has been deployed as a part of a building monitoring system where the systems responsibility is to monitor different environmental parameters in different parts of the building. In addition to that the system is responsible for controlling power supply of each of those parts of the building by switching power supply on and off. To achieve such a functionality, each and every sensor mote of the network is equipped with temperature, humidity and light sensors. For switching actuators, sensor motes are connected to specialized hardware components which provides facility for the mote to control power supply to different connected devices. The software layer of the system operates with the database abstraction of TinyDB.

Now we have a requirement to turn off a specific light bulb in a room if the light level of the environment goes higher than a threshold value. Lets say the light level measurements are taken by a node (nodeid 2) which resides near the window of the room while the light bulb controller is connected to another node (nodeid 3) that resides at the center of the room. This is a distributed actuation scenario since the actuator and sensor are residing in separate nodes. Based on the grammar definition of TinyDBs query language, we can identify three approaches to achieve the above actuation requirement.

Approach-1

First we can issue a query (Query-8) from the base station which will monitor the light level readings of the room. If light level exceed the threshold value (40), base station receives a result set from the sensing node. If so, base station issues the next query (Query-9) which will make the actor node to turn off the light bulb.

Query-8

```
SELECT nodeid,light
FROM sensors
WHERE nodeid = 2 AND light > 40
SAMPLE PERIOD 10s
```

Query-9

```
SELECT nodeid
FROM sensors
WHERE nodeid = 3
OUTPUT ACTION power-off()
SAMPLE PERIOD 1s FOR 1s;
```

Approach-2

We can issue a query (Query-10) from the base station which is a kind of a query that will gets executed only when a special event is triggered. The special keyword *ON EVENT* is used to specify the event name. This query is store on the sensor node and waits until the event triggers. After issuing this query we issue another query (Query-11) targeting the same sensor node. This second query continuously monitors the light level of the room. If it exceeds the threshold value it triggers the first query by sending the event signal which is specified by the *SIGNAL* keyword. When the event received to first query, sensor node broadcasts the select query which is embedded in the event query to the actuator node. Now the actuator node can turn the light bulb off.

Query-10

```
ON EVENT light-high():
SELECT nodeid
FROM sensors
WHERE nodeid = 3
OUTPUT ACTION power-off()
SAMPLE PERIOD 1s FOR 1s;
```

Query-11

```
SELECT nodeid
FROM sensors
WHERE nodeid = 2 AND light > 40
OUTPUT ACTION SIGNAL light-high()
SAMPLE PERIOD 10s;
```

Approach-3

Since the *OUTPUT ACTION* keyword in TinyDB permits to embed a *SELECT* query in it, we can issue a query like the below one targeting the sensor node. If the conditions in *WHERE* clause results true, the embedded *SELECT* query is sent from the sensor node to the actor node to perform the actuation.

Query-12

```
SELECT nodeid
FROM sensors
WHERE nodeid = 2 AND light > 40
OUTPUT ACTION
(
SELECT nodeid
FROM sensors
WHERE nodeid = 3
OUTPUT ACTION power-off()
SAMPLE PERIOD 1s FOR 1s
)
SAMPLE PERIOD 10s;
```

Among these three approaches, the first two involves more than one query resulting the drawbacks described in the chapter 1 while the third approach does not lead to those drawbacks in this particular scenario. However there are real world situations where the usage of embedded *SELECT* queries inside the *OUTPUT ACTION* clause also does not provide enough expressive power to perform an actuation task using a single query.

For example if the actuation decision has to be taken based on the light level taken by one sensor node and the room temperature taken by another sensor node, the *WHERE* clause of the enclosing *SELECT* query cannot be used to specify all the conditions by a single query. In such a situation we have to go for approach one and at first use multiple *SELECT* queries to check all the conditions. Finally the actuation decision have to be taken at the base station before sending the actuation query.

Similarly if we have to activate different hardware devices based on the readings of the sensor nodes we face difficulties again. For example if we have to turn off the light bulb connected to a one actuator node and ring a bell connected to another actuator node when the light level goes high, we have no way to express it by a single query due to the limited facilities in the *OUTPUT ACTION* clause. In that situation also we have to go for multiple

actuation queries sent from the base station which leads to the drawbacks explained previously.

This discussion highlights the fact that the existing distributed in-network actuation queries provided in TinyDB are not applicable in certain real world application scenarios and therefore it needs to be improved if we want to use declarative query interface in real world WSAN environments.

2.2 Non-declarative interfaces for WSAN

In the literature a common assumption has been applied when designing WSANs which is the actuator nodes in the network are more resource rich when compared to the sensor nodes specially in energy[1]. This is mainly due to the fact that actuator nodes are connected to external devices for performing actions and therefore they can be directly connected to an external power source without relying on batteries. Therefore computational and communication intensive tasks are more likely to be performed on actuator nodes rather than on sensor nodes.

The traditional WSN systems which are intended for collecting data basically follow a centralized architecture where sensor data is sent to the base station over the network so that anything to be done with data is performed outside the network. However when comes to WSANs with the addition of actuators the total paradigm got changed [9, 8]. WSANs involve a loop of event sensing, decision making and actuating unlike the WSNs where sensing is the only action of interest. Therefore applying the general WSN architecture of centralized base station control simply does not work. Scalability, associated delays and power consumption makes the centralized architecture almost inapplicable in large scale WSANs [12]. Therefore [8] suggests that it is better to move the actuation cycle into the network which is introduced as *In-network actuation*.

Similarly the study presented in [1] views the WSANs in two architectures as automated and semi-automated. The semi-automated architecture follows the traditional WSNs ie. sensor events are delivered to the base station where the actuation decisions are made and sent back to the actuators over the network. The automated architecture is the one called as in-network actuation in [8] where sensors and actuators work collaboratively within the network to perform the loop of actuation. Therefore in brief, in-network actuation is generally considered as the correct way to go in WSANs.

When considering the literature we can identify different approaches followed by researchers to achieve in-network actuation. The paper [8] identifies two different middleware designs used for implementing in-network actuation

as node-centric and network-centric designs according to the programmers point of view. In the node-centric middleware design, the programmer focus on the functionality of each an every node in the network while in the network-centric architecture the focus goes to the functionality of the network as a whole. Clustering, macro-programming and query abstraction layers are such network-centric middleware implementations.

According to [1], a proper coordination between nodes in the network is necessary to achieve the in-network actuation in a WSAN. The coordination between nodes can be decomposed further into two types as sensor-actuator coordination and actuator-actuator coordination. The former type considers how a sensor node should report an event detected to the corresponding actuator node while the latter type considers how multiple actuators should coordinate with each other to decide which actuator should perform the required action for a particular event received from a sensor node.

The research done in [12] presents a clustering approach to implement an efficient and scalable distributed in-network actuation. Sensor and actuator nodes are clustered together in a way that nodes interested at same environmental phenomenon or event source belong to the same cluster. However it is possible for a sensor or an actuator node to join into multiple clusters. This clustering mechanism involves a distributed algorithm which is performed on each and every node in the network. However according to [17], that type of predefined clustering mechanisms may not perform efficiently as we expect. If the environment is highly dynamic such that all the clusters in the network experience events frequently then predefined clustering works. But if the events are less frequent then this approach wastes energy of nodes to maintain clusters unnecessarily.

Avoiding such drawbacks [17] presents a different approach where node clustering is done on-the-fly as sensor events arrive. In their research they have considered both sensor-actor coordination and actor-actor coordination at different phases. The sensor-actor coordination is done using *data aggregation trees*(da-trees). When an event is detected by some sensor nodes in the network, one or more da-trees are generated by considering the relations of the sensor and actor nodes which are defined as *flows*. The root of a da-tree is an actor node while all the other non-terminals are sensor nodes. The leaves of a da-tree are event sources theoretically. When building a da-tree, geographic locations of the nodes are also taken into account so that an efficient communication within the da-tree is achieved.

After the da-trees are generated, event details are sent from sensor nodes in the tree to the root actor node. Event data propagation over the da-tree uses a data fusion algorithm so that each sensor node aggregates the received event details from child nodes with its own event details before

sending it to the upper node in the da-tree. Therefore the energy used in the communication is minimized. When an event is notified to an actor node it begins the actor-actor coordination using a *real-time auction protocol* [17] to select the appropriate actor node to perform the action for the detected event. This protocol considers lot of parameters like power consumption, geographic location and features of the event to select the appropriate actor.

A similar approach is presented in [20] with a focus in real-time performance of the in-network actuation. In that research they use geographic location based map trees to create clusters in real-time instead of the da-trees in [17]. When an event is detected by some sensor nodes they create a map tree covering there geographic location and sends the event details to the root of the tree where the root node delivers the received event details to its geographically closest actor node. To achieve such a functionality the nodes should be aware of their geographical location. In this sensor-actor coordination mechanism, the actor node to which the event details are delivered doesn't have to be the most appropriate actor node to perform the action because a separate actor-actor coordination mechanism decides it. A major difference of this work [20] to the previously described research [17] is the mobility support for the actor nodes in the actor-actor coordination mechanism in this research.

In the work done in [16], two actuation techniques namely *On-event triggered* and *Self triggered* actuation have been explored where both methods helps to reduce the energy consumption by reducing the messages passed between sensors and actuators. In the first method, the actuation decision is taken by the sensor nodes in a distributed manner and triggers an event to the actuator node to perform the action. Therefore the actuation decision is taken by the sensor nodes. Unlike it, the second method is driven by the actuator nodes where they takes the actuation decisions based on the data received from sensor nodes. In both techniques, an actuator node should be a root in a sub tree of the routing tree where the respective sensor nodes are child nodes of it. An algorithm called *tree wave algorithm* has been used to aggregate the messages sent within the routing tree which leads to the reduction of energy consumption.

The authors of [9] point out the complexity of WSAN platforms due to the wide variety of possibilities in real world deployments. For example the network can be homogeneous with similar nodes or heterogeneous with sensing and acting done on different nodes. Such complexities are hard to be handled by the application developers of WSANs and therefore they have introduced a macro-programming language named as SOSNA to address this issue. According to the authors, SOSNA macro-programming language abstracts away the underlying complexity of WSAN while providing the real-

time functionality. However since the actuation criteria of the WSAN have to be programmed using SOSNA, compiled and burned on the nodes flash chips, the interactiveness of the WSAN is not achieved since the behavior of the WSAN completely depends on the program logic. Comparing to this, declarative interfaces like TinyDB provides more freedom for the end users of WSAN to change actuation criteria on-the-fly even though existing queries are not expressive enough to do it in an efficient way.

Chapter 3

Methodology and Design

In our attempt to solve the identified issues in declarative interface, we begin with the current data model (Table 3.1) and build a better syntax support for actuation queries. In this process we consider requirements of distributed in-network actuation scenarios in order to come up with a solution that could be applicable in all such cases.

3.1 Virtual Data Table

Table 3.1 shows an example virtual table of a database abstraction over a wireless sensor-actuator network. According to the table, there are three nodes in the network and each node has two sensor types, temperature and humidity. The values of those sensors shown in the table are acquired in a particular instance of time by a *SELECT* query. If a node in the network does not have a particular sensor type the corresponding cell in the table will contain a NULL value.

Now consider each node in the network is equipped with two actuator types namely an air cooler and a sprinkler which are controlled electronically. Since the actuators in the network are not represented in the table, still the virtual table will look like Table 3.1 while the user have to control actuators using low level function calls in *OUTPUT ACTION* clause of *SELECT* query. Moreover, there's no way for the user to find the current status of each actuator in the network.

To overcome the problems associated with the current mechanism and thus to provide the benefits of declarative interface on in-network actuation we understand the importance of enhancing the virtual table of the database abstraction with actuator types. As such the status of each actuator type in the network is represented as an attribute in the virtual table in addition

Table 3.1: Traditional sensor table

nodeid	*temperature*	*humidity*
1	25	40
2	28	39
3	27	NULL
4	NULL	41

Table 3.2: Enhanced table with actuators integrated

nodeid	*temperature*	*humidity*	*air_cooler*	*sprinkler*
1	25	40	NULL	NULL
2	28	39	off	off
3	27	NULL	on	NULL
4	NULL	41	off	on

to sensor types as shown in Table 3.2. Most significantly this enables user to access actuators in each node just as the way sensors are accessed. Consequently when the user wants to know the current status of a particular actuator or a set of actuators in the network, simply *SELECT* queries as given below will do the job.

Query-13

```
SELECT nodeid, air_cooler, sprinkler
FROM sensors
ONCE;
```

Query-14

```
SELECT nodeid
FROM sensors
WHERE air_cooler='on'
ONCE;
```

Query-15

```
SELECT count(*)
FROM sensors
WHERE sprinkler<>NULL
ONCE;
```

3.2 Actuation Using Updates

Since sensor attribute values in the virtual table represent real time acquired data of the sensors, they can not be altered by the user. However actuator attribute values in the virtual table are alterable and this enables the actuator status to be changed according to the user requirements. This situation opens the door to control actuators in the network by applying updates on the virtual table using conventional *UPDATE* query syntax without using low level function calls as explained previously. However since these queries are executed over a WSAN, it is necessary to have the syntax support to specify the sampling rate of sensor /actuator attributes just as in the existing approach.

For instance if the user wants to turn the air cooler in node 2 to 'on' state only if the temperature reading of that node is greater than 20, a query like the one shown in Query-16 can be used.

Query-16

```
UPDATE sensors
SET air_cooler='on'
WHERE nodeid=2 AND temperature>20
ONCE;
```

When this query is sent to all the nodes in the network, each node will evaluate the predicates in *WHERE* clause. If a node reveals that those predicates are true on it, it can turn the air cooler actuator connected to it to 'on' state. This illustrates a *local in-network* actuation scenario.

3.3 Distributed In-network Actuation

When we need to perform in-network actuation tasks where multiple sensor nodes and actuator nodes involve in a single actuation, simple queries like the Query-16 can not be used. For instance consider the requirement that we need to turn the air cooler to 'on' state in both nodes 2 and 4 only if node 1 is having a temperature reading greater than 20. We consider the join of multiple table aliases can be used to address this requirement with the actuation criteria specified in *WHERE* clause of the *UPDATE* query as given in Query-17.

Query-17

```
UPDATE sensors AS sen, sensors AS act
SET act.air_cooler='on'
WHERE (act.nodeid=4 OR act.nodeid=2) AND
 (sen.nodeid=1 AND sen.temperature>20)
ONCE;
```

Two aliases for the *sensors* table have been taken as *sen* and *act*. We use the *sen* alias to specify the criteria of the sensor nodes while the *act* alias to specify the criteria of actuator nodes. This enables distributed in-network actuation tasks with different complex actuation criteria using a single *UPDATE* query unlike the existing actuation mechanism with *SELECT* queries. The grammar definition of this *UPDATE* query syntax is shown below.

```
UPDATE <table> [AS <alias>]
    {, <table> [AS <alias>]}*
SET [<alias>.]<attribute>=<value>
    {, [<alias>.]<attribute>=<value>}*
[
WHERE
 [<alias>.]<attribute>{<|>|=|<=|>=}<value>
 {{AND|OR}
 [<alias>.]<attribute>{<|>|=|<=|>=}<value>}*
]
[SAMPLE PERIOD <seconds> [FOR <nrounds>]]
 | [ONCE]
```

3.4 Efficient Execution of Updates

Even though the syntax presented in the previous section provides the support to declarative specify actuation criteria for WSANs, there's a weakness in executing this suggested query syntax which limit it from executing more efficiently than the existing queries. The problem occurs when performing distributed in-network actuation scenarios where different sensors and actuators from different nodes involve in a particular actuation scenario.

When an *UPDATE* query is issued to the network each node receive a copy of it. When a node is trying to execute the query, it can access only the sensors and actuators which are available on the node. For example consider the Query-17 presented in the previous section. When the nodes 1, 2 and

4 received that query they can start to execute it. Let's say the temperature sensor reading of node 1 is recorded as 24. Then node 1 can identify that the predicate segment ***(sen.nodeid=1 AND sen.temperature¿20)*** is true on that node. In the mean time node 2 and 4 identify that the predicate segment ***(act.nodeid=4 OR act.nodeid=2)*** is true on them. So, the complete set of predicates in the *WHERE* clause evaluates to true and therefore the actuation task should be performed now by node 2 and 4 since they belongs to the alias which mentioned the *SET* clause of the query. However the problem is node 2 and 4 has no way to get to know that the total *WHERE* clause of the the query has evaluated to true.

Because of this reason we need a way to execute *UPDATE* queries in different scenarios in WSANs including distributed in-network actuation scenarios. One obvious option is to collect the necessary sensor data and actuator status from the nodes to the base station and evaluate the predicates at the base station. Then the application running on the base station can identify the nodes which should perform actions and it can send commands to those nodes. However this method is almost same to the traditional way of performing distributed in-network actuation tasks using *SELECT* queries. The only difference is in the traditional method a human user involvement is necessary to evaluate the actuation criteria. But since sensor data has to come through the network to the base station to take the actuation decisions this method is highly inefficient. Because of these reasons we are suggesting a better execution strategy for *UPDATE* queries which uses less resources from the network to perform actuation tasks.

3.4.1 Details of the network

We assume that the network is organized as a tree which is rooted at the base station as shown in figure 3.1. Because of this tree-based architecture there are three types of nodes available in the network as *Root node*, *Sub-root node* and *Leaf node*. Each node type processes *UPDATE* queries in different ways as explained shortly. This network uses special messages to exchange information between nodes related to an execution of a query. Those messages are namely UPDATE, SUCCESS, PARTIAL-SUCCESS and COMMAND messages.

The UPDATE message is created at the base station (*Root node*) after the user issued an *UPDATE* query. The figure 3.2 shows the structure of the message. There are two main blocks in the message. First one is SET block which contains the information of the *SET* clause of the query while the second block is WHERE block which contains the information of the *WHERE* clause of the query. SET block is fragmented to parts while each part repre-

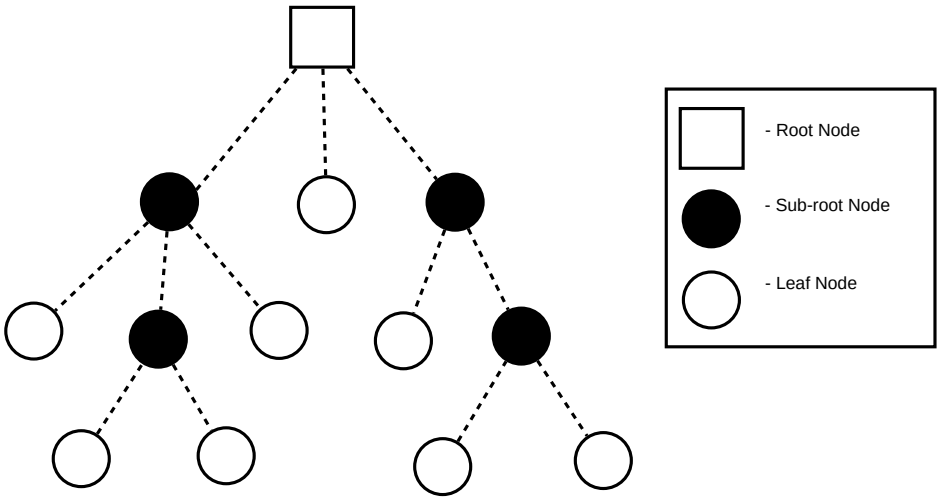

Figure 3.1: Routing tree of the network with three node types as root node, sub-root nodes and leaf nodes.

senting an alias used in the *SET* clause of the *UPDATE* query. These alias parts are more fragmented to represent each actuator-value pair. Similarly the WHERE block is fragmented to parts while each part representing an alias used in the *WHERE* clause of the *UPDATE* query. These alias parts are more fragmented to represent each predicate. Each alias included in the message are numbered to identify uniquely. UPDATE messages propagate from the *Root node* through the network until it reaches all the nodes in the network.

A SUCCESS message is issued by a node targeting at its parent node to inform that all the predicates in the *WHERE* clause of an *UPDATE* query has been evaluated to true in that branch of the tree rooted at message sender node. This message does not contain any other information. A PARTIAL-SUCCESS message notify that not all but some of the predicates in the query has been evaluated to true in the branch of the tree rooted at message sender node. The message contains alias numbers which have evaluated to true in that branch of the network along with the IDs of the nodes which have resulted these partial evaluations. A COMMAND message is issued to a node to perform an actuation using the actuators available on that particular node.

3.4.2 Functionality of nodes

The behavior of each node type in the network is as follows.

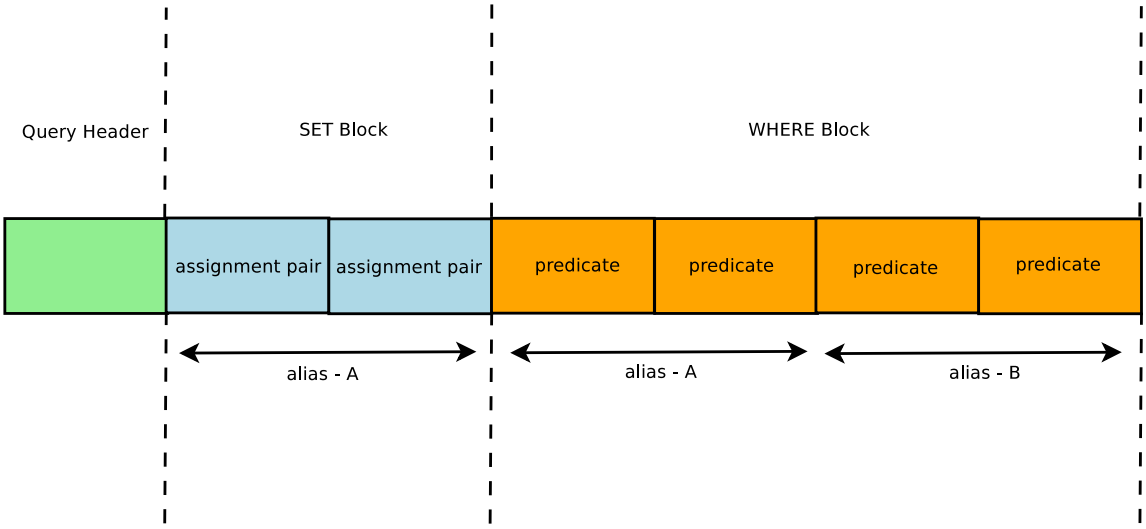

Figure 3.2: Structure of the UPDATE message.

Leaf node

A *leaf node* evaluates the predicate segments of an UPDATE message. If all of the segments evaluate to true, it will execute the actuation tasks given in the SET block of the message. Then it sends a SUCCESS message to the parent node. It means all the predicates are evaluated to true in this node. This message is useful for the parent node to perform its works.

If only few of the predicate segments evaluated to true, the *leaf node* sends a PARTIAL-SUCCESS message to the parent node mentioning the predicates that have been evaluated to true with its node id. If no any predicate segment evaluated to true, then *leaf node* exit from executing the query.

Sub-root node

A *sub-root node* waits a predefined time before it evaluates a received UPDATE message. If it receives at least one SUCCESS response from a child node, this *sub-root node* may do one of the following things. If it has received reports from other nodes with PARTIAL-SUCCESS of predicate segments, this *sub-root node* check whether SET block of the UPDATE message has some actuation related to those aliases that have been true in its children. If so, the *sub-root node* sends COMMAND messages to those particular children. Then *sub-root node* returns a SUCCESS message to its parent node.

Even though *sub-root node* does not receive a SUCCESS response from

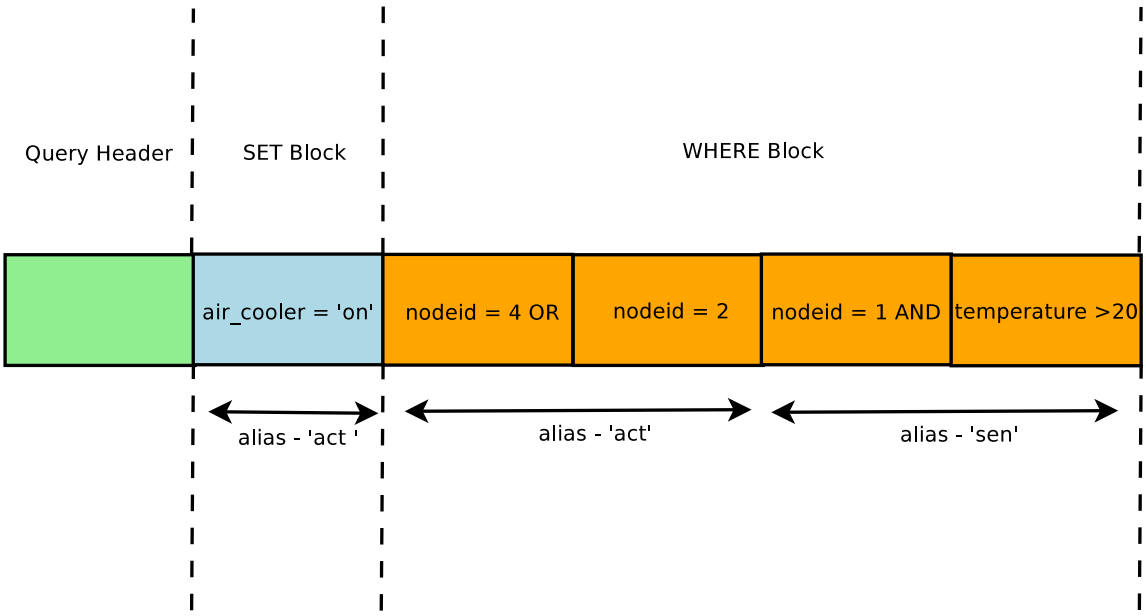

Figure 3.3: Content of the UPDATE message for Query-17.

a child node, if its child nodes PARTIAL-SUCCESS messages altogether including the *sub-root nodes* local evaluations makes a complete true of all the predicates, then the *sub-root node* can send COMMAND messages to the relevant child nodes. Then it can report a SUCCESS message to the parent node.

If all the child nodes responses are PARTIAL-SUCCESS messages and does not evaluate to true all together, this *sub-root node* forwards all the PARTIAL-SUCCESS messages including its own to the parent node.

If no child nodes responded with either SUCCESS or PARTIAL-SUCCESS reports and *sub-root node* did not have evaluate anything to true either, then this *sub-root node* exit from executing the query.

Root node

The *root node* does the same thing as *sub-root nodes* except one thing. Since *root node* does not have a parent node, it does the following. If *root node* has to report SUCCESS to parent, what it does is report to the client application saying the query executed successfully. If *root node* has to report PARTIAL-SUCCESS or no report to parent, it reports to the client application saying the query did not has any matching tuples in the virtual table to perform the UPDATE operation.

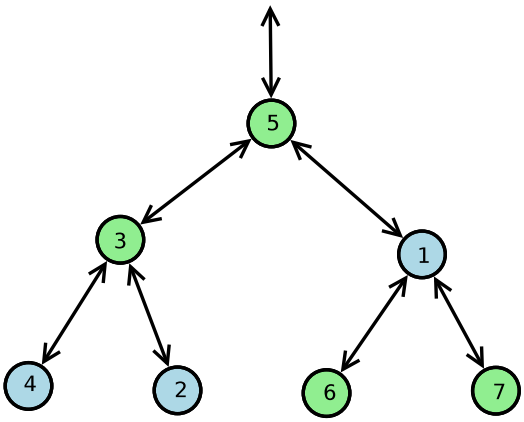

Figure 3.4: Placement of nodes 1, 2 and 4 in the network for the example case with Query-17.

3.4.3 An example case

To demonstrate how this query execution strategy works, let's consider the *UPDATE* query presented in Query-17. Figure 3.3 shows the structure of the UPDATE message which contains the information of Query-17. The placement of nodes in the network is as given in figure 3.4. Node 5 is a *sub-root node* in a large network and our respective nodes 1, 2 and 4 reside inside that branch rooted at node 5. When the UPDATE message is flooded through the network it reaches all the nodes in this branch too. Since node 2 and 4 are *leaf nodes* they immediately start execution after receiving the UPDATE message. In the meantime node 1 waits for a predefined time, collects responses if any from child nodes and start executing the query.

When node 2 and 4 realize that they belongs to the 'act' alias given in the query, they send two PARTIAL-SUCCESS messages to the parent node which is node 3. Since node 3 does not receive any responses from any other child node it evaluates the query. It realize that it does not belong to any alias in the query. So only thing it does is forwarding a PARTIAL-SUCCESS messages to the parent which is node 5 with the information received from node 2 and 4. When node 1 realize that it belongs to 'sen' alias of the query, it sends a PARTIAL-SUCCESS message to the parent node which is node 5.

Node 5 received three PARTIAL-SUCCESS messages and when it evaluate them, it realize that now this branch of the network has at least one node for every alias type in the query. Therefore node 5 checks the alias types given in the SET block of the UPDATE message and it realize that node 2 and 4 are the nodes which belongs to that alias type which is 'act'

alias. Therefore node 5 compose COMMAND messages and issue to the network targeting at node 2 and 4 to perform the actuation tasks. Then node 5 sends a SUCCESS message to its parent node which will propagate through the network and reach the *root node* or the base station. By receiving this message at the base station, it can signal the user that the query is executed successfully.

Chapter 4

Experiments and Results

For the purpose of evaluating the suggested solution, we are mainly concerned on three aspects of proposed query syntax and query execution strategy as listed below.

1. Applicability.
2. Declarative nature.
3. Performance impact.

For the applicability, our concern is whether the proposed actuation mechanism is able to cater for any type of actuation requirement that can occur in wireless sensor-actuator networks. The declarative nature aspect is the comparison of the declarative nature of the proposed syntax in writing queries for complex actuation tasks. We also evaluate the impact on the performance of actuation tasks when using the proposed query syntax and execution mechanism. Our prototype developments are designed to support evaluating these parameters as we explain in the next section.

4.1 Implementation Details

Our evaluation has two implementation requirements. Since we need to evaluate our suggested SQL query syntax, we need a prototype with the facility to write queries and see them work. On the other hand we need to evaluate our query execution strategy in a large scale with large number of nodes. We realized that it is better to use two prototype implementations which fulfill each of these requirements separately. This is because different simulators and WSN platforms provide different feature and we need to choose best one

depending on our each requirement. Therefore we have two implementations one is on Cooja simulator[21] which is used for declarative interface based evaluations while the second implementation is on GloMoSim[24] simulator which is used for our query execution strategy evaluations.

4.1.1 Declarative interface

We implemented declarative interface with the support for suggested query syntax based on TikiriDB[10]. We used TikiriDB source code and added the *UPDATE* query support to it. Figure 4.1 shows the high-level system architecture of TikiriDB. It has three components namely a *node application*, a *serial forwarder* and a *client application*. Figure 4.2 shows how these components look on the Cooja simulator GUI interface. The *client application* accepts SQL queries from the user, parses them and forward to the serial forwarder. The *serial forwarder* runs on the PC which is connected to the base station of the WSAN. Its task is to accept the queries which are coming from the *client application* and forward them to the WSAN. Each node in the network runs a copy of the *node application* which executes the queries and generates results.

In this prototype we did not implement the query execution strategy. Instead *UPDATE* queries are performed in the same way as we perform an actuation using traditional *SELECT* queries. When the user issues a *SELECT* query, it is parsed and sent to the network. Nodes respond with data which are provided to the user. When an *UPDATE* query is issued by the user, *client application* first collects the readings of the necessary sensors and status of necessary actuators. Then it processes those data to identify the nodes which have to change their actuator status according to the query. Then *client application* sends commands for those nodes to update their actuator status.

4.1.2 Query execution strategy

We implemented query execution strategy on GloMoSim simulator[24]. The advantage of GloMoSim over Cooja simulator is that GloMoSim comes with various routing protocols such as AODV, DSR, etc in default and it can be used to simulate large wireless networks with huge number of nodes. We implemented a special application for GloMoSim which provides the functionality to send an UPDATE message from a special node to the rest of the nodes in the network and execute it according to the strategy presented in the chapter 3.

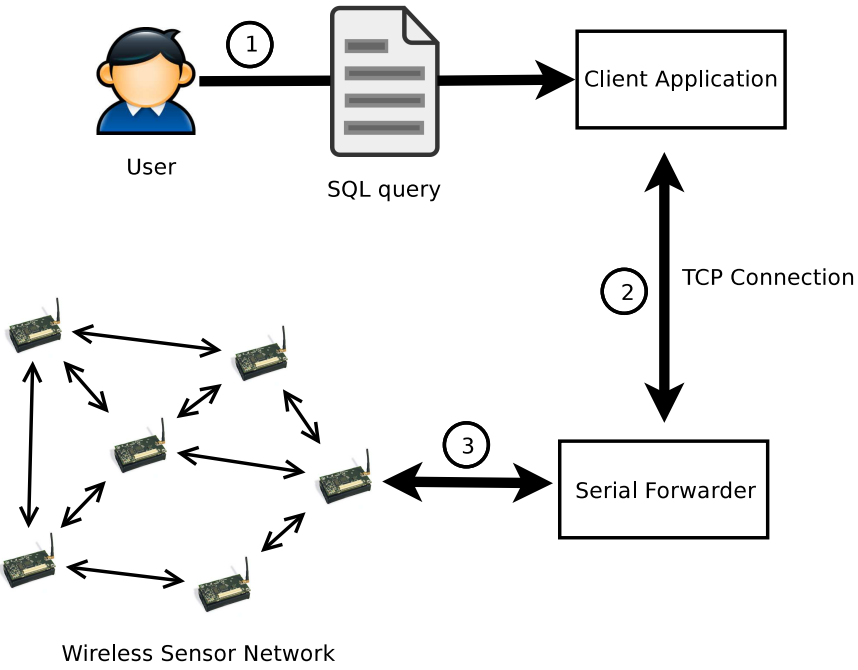

Figure 4.1: High-level system architecture of TikiriDB. (1) End user writes a SQL query and enter to the client application. (2) Client application parse the query, send it to the serial forwarder, receive the data from serial forwarder and send back to client application. (3) Serial forwarder exchange the queries and data between client application and the WSN.

4.2 Evaluation

4.2.1 Applicability

To evaluate the applicability of the new syntax for actuation we consider mainly three types of actuation scenarios that can occur in a wireless sensor-actuator network. Those are hybrid sensor-actuator nodes scenario, a pair of sensor nodes and actuator nodes scenario and multiple sensor nodes and actuator nodes scenario. Our goal is to show that each of these actuation scenario can be addressed by the suggested syntax.

Hybrid sensor-actuator node scenario

In this case both sensors and actuators of our concern are in the same node of the network. For instance, consider the node 2 in the virtual table showed in Table 3.2. We need to turn its air cooler and sprinkler to 'on' state if its temperature reading is greater that 25 and humidity value is lower than 45.

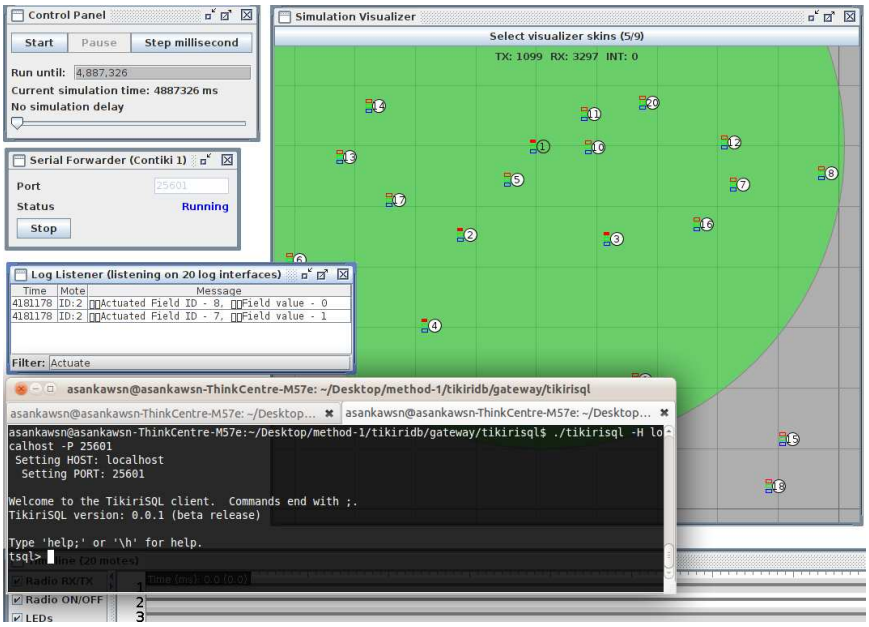

Figure 4.2: Prototype implementation based on TikiriDB[10] on Contiki OS[6].

We can perform this actuation by using a query like the one given in Quary-18.

Query-18

```
UPDATE sensors
SET air_cooler='on', sprinkler='on'
WHERE nodeid=2 AND
  temperature>25 AND humidity<45
ONCE;
```

A Pair of sensor node and actuator node scenario

The simplest distributed in-network actuation scenario is the involvement of two nodes where the required sensors are in one node and the required actuators are in another node. For instance we need to switch off the sprinkler of node 4 only if the node 1 has a temperature value greater than 20. We perform this actuation task by using the Query-19.

Query-19

```
UPDATE sensors AS sen, sensors AS act
SET act.sprinkler='off'
WHERE act.nodeid=4 AND
  sen.nodeid=1 AND sen.temperature>20
ONCE;
```

Multiple sensor nodes and actuator nodes scenario

In this case we consider actuation requirements where multiple sensor nodes and actuator nodes involve in an actuation task. For instance, we need to switch off sprinkler in node 4 and switch on air cooler in node 2 only if all the nodes in the network which are having temperature sensors report a value greater than 25. We perform this actuation task as shown in Query-20. Since the temperature attribute has to be evaluated on all the nodes in the network, it is not associated with any alias in the query.

Query-20

```
UPDATE sensors AS act1,sensors AS act2
SET act1.sprinkler='off',
    act2.air_cooler='on'
WHERE act1.nodeid=4 AND
  act2.nodeid=2 AND
  temperature>25
ONCE;
```

4.2.2 Declarative nature

Moreover we evaluated the declarative nature of the suggested syntax over the existing syntax by a user survey. We randomly selected a group of 10 students who have an average knowledge on acquisitional queries and we provided them with different actuation scenarios. We asked them to write SQL queries to perform those actuation tasks in both existing and the suggested approaches. Finally we collected feedback from them regarding how they found writing queries for actuation tasks in both approaches.

It turned out that each person participated in the survey considers the existing approach of performing actuation tasks has unnecessary syntax restrictions. All the participants consider the modified virtual table with actuators as attributes is more convenient to handle than the old table. As such they have mentioned that *UPDATE* queries can be easily written to perform

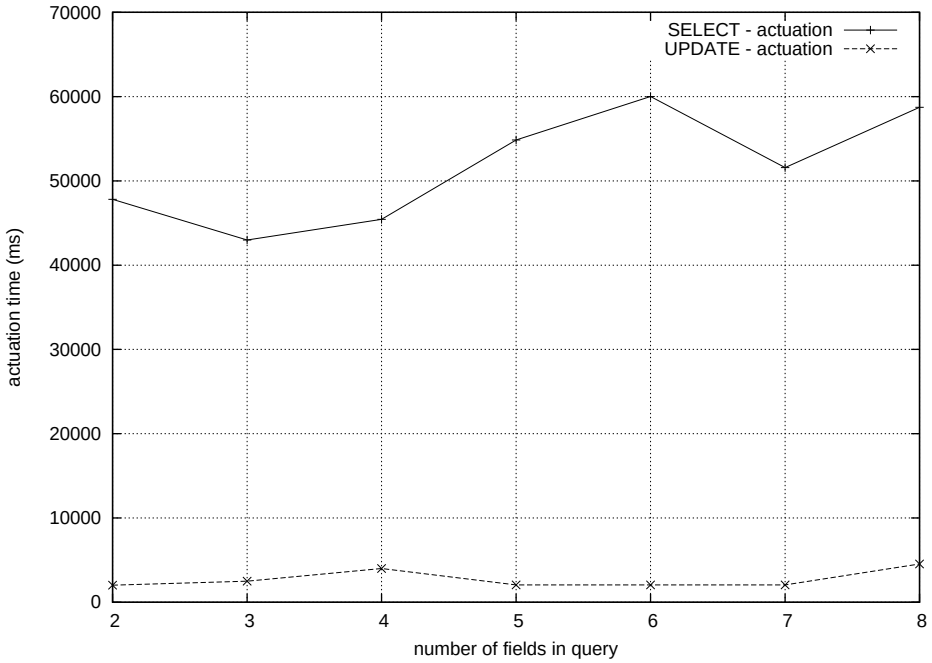

Figure 4.3: Actuation time variation against the number of attribute fields (sensors/actuators) in the query.

actuation tasks naturally without using artificially forced syntax parts of *SELECT* queries in existing approach. We further noticed that most of them have struggled to write queries using the existing syntax for the greenhouse scenario given in chapter 1 whilst they have written with a single query for the same scenario as given in Query-21 using the proposed syntax.

Query-21

```
UPDATE sensors AS sen,sensors AS act1,
  sensors AS act2,sensors AS act3
SET act1.air_cooler='on',
  act2.screen_ctrl='on',
  act3.sprinkler='on'
WHERE (sen.nodeid=5 OR sen.nodeid=7)
 AND
 sen.temperature>35 AND sen.humidity>45
 AND
 act1.nodeid=9 AND act2.nodeid=15 AND
 act3.nodeid=23
ONCE;
```

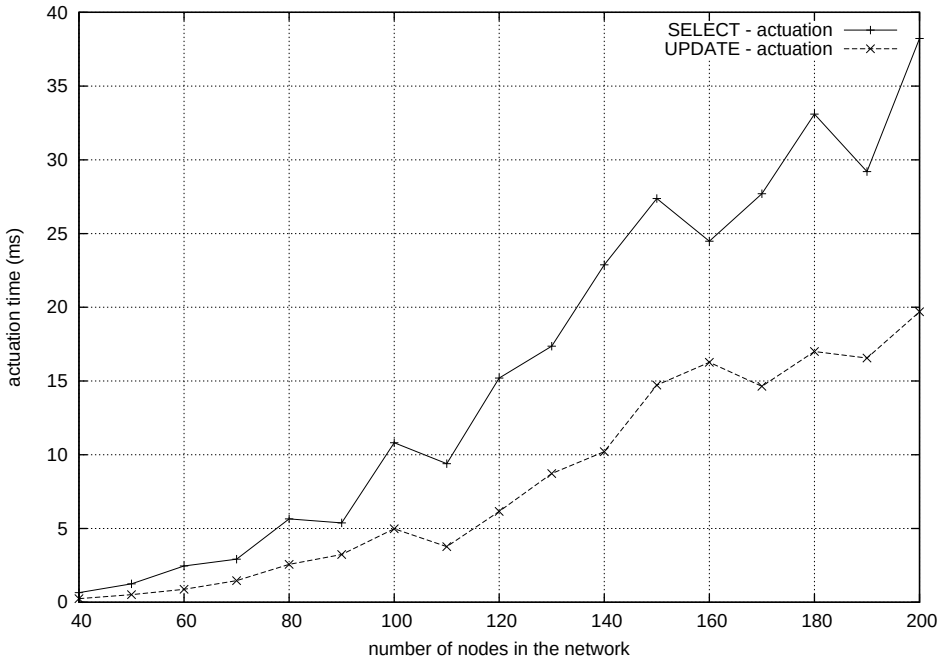

Figure 4.4: Actuation time variation against the number of nodes in the network. (P.S- The delay introduced by human intervention to the *SELECT* query based actuation is excluded in the graph.)

4.2.3 Performance impact

Actuation time

We collected the data of the time to perform an actuation against different number of attribute fields in the query using the Cooja simulator based prototype. For this purpose we considered various actuation scenarios which involve different number of attributes in the virtual table. With respect to these scenarios we used queries from both existing and suggested approaches. Figure 4.3 shows the time to perform an actuation against the number of sensor / actuator attributes involved in the actuation task. According to this graph theres a significant performance difference between the two approaches since traditional approach requires human intervention to issue multiple queries to perform an actuation task while new syntax eliminate multiple query usage.

We evaluated the actuation time variation with different number of nodes in the network using our GloMoSim simulator based prototype implementation to find whether any improvement has been achieved by the *UPDATE* query execution strategy. For this evaluation we excluded the time delay which gets introduced to the execution time by the human intervention to

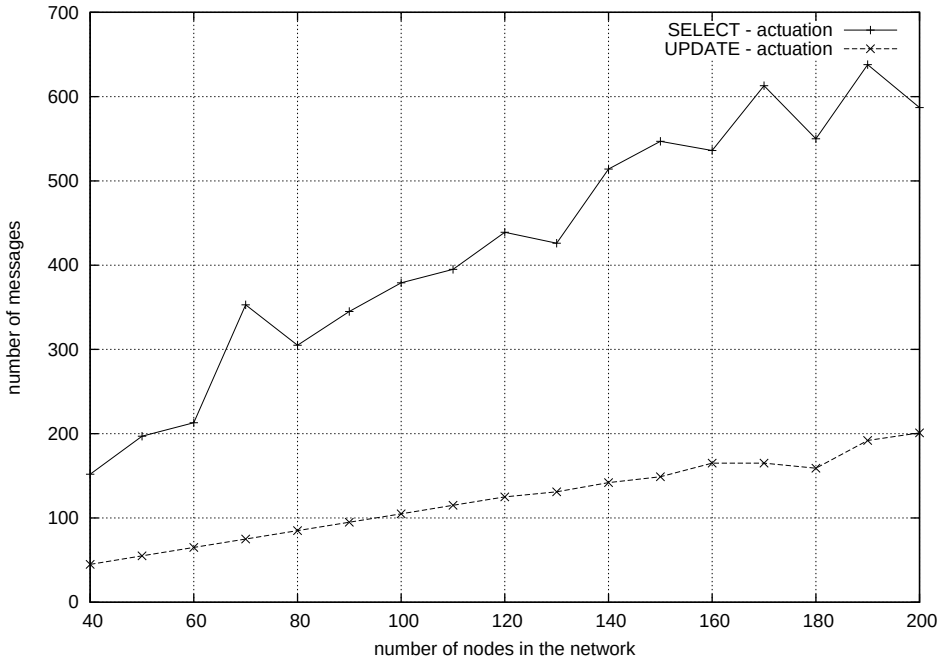

Figure 4.5: Communication cost variation against the number of nodes in the network.

run multiple *SELECT* queries. Figure 4.4 shows the results of this evaluation. According to this graph, the *UPDATE* query based actuation out performs the *SELECT* query based actuation. Since the human intervention delay is excluded from the data, this graph clearly shows that the *UPDATE* query execution protocol has contributed to improve the actuation performance.

Communication cost

We used the same GloMoSim based prototype and collected the number of messages passed between nodes to perform an actuation against the number of available nodes in the network. As the figure 4.5 shows the number of messages passed between the nodes is considerably lower when *UPDATE* queries are used to perform actuation tasks with the aid of our new execution strategy. Since the number of messages used in *UPDATE* query based actuation is lower than *SELECT* query based actuation, the battery power cost for communication can be assumed to be lower in *UPDATE* query based actuation than *SELECT* query based actuation.

Chapter 5

Discussion

In the chapter 4 we looked at different aspects of our proposed solution to the issues we identified in the declarative interface. Our prototype implementations were specifically designed to meet the requirements of the evaluation parameters and scenarios. Some aspects were evaluated using end-user surveys and example based analysis while other parameters were evaluated using prototype based experiments.

According to the user survey, performing actuation tasks with an *UPDATE* query is more convenient than using *SELECT* queries with low level function calls. The declarative nature of *UPDATE* syntax makes it easy to express complex actuation tasks within a single query when compared to the traditional approach where multiple queries may have to involve with system dependent function calls. Since the enhanced virtual table contains actuators as attributes, the user gets the opportunity to easily find out the current status of actuators in any given time. This also enables the user to perform in-network actuation tasks not only based on sensor values but also based on current actuator status.

As we have shown, *UPDATE* query syntax can be used to perform almost any complex actuation requirement that can arise in a wireless sensor actuator network(WSAN). The syntactic support along with the aliases can be used to coordinate multiple nodes with sensors and actuators by writing a single *UPDATE* query. Comparing to the traditional approach, this single query usage to perform an actuation has improved the performance since multiple queries take more time to execute with the additional time taken by human involvement. This enables the WSAN to respond quickly for changes in the environment.

Even though it appears like having NULL values in the virtual data table where some sensors or actuators are absent on a particular node as a drawback, it does not introduce any performance impact. When a node re-

ceives a query asking for some sensor / actuator attribute value which is not available on the node, the node sends only the values of the attributes which are available on the node. A Node does not send any messages to the *client application* to inform that the value of a particular sensor / actuator attribute is NULL. It's the client application which assigns a NULL value to each nodes data set where data is absent when arrive at the base station. Because of this reason, having lot of NULL values in the virtual data table does not represent any communication or any other resource overhead on the network. To state that more formally, the heterogeneity of nodes in the network does not introduce any overhead to the network.

Our query execution strategy has several drawbacks by its design. When executing a query we assume that a tree based message routing mechanism is available on the network which is rooted at the base station. This tree can be an overlay to a low level routing protocol. We did not evaluated the cost of building a tree where the bases station of the network becomes the root. However it is quite fair to ignore this cost because once the tree is built, the network does not need to rebuild the tree throughout its operational life time unless some sub-root nodes fail due to hardware/software failure making some branches of the network inaccessible from the base station. At this point, another important assumption plays a major role. We are considering a static network where the nodes does not move. Therefore node mobility has no way to effect the initiated message routing tree. Again this assumption is much fair to make since the node mobility requirement is less likely to happen in a wireless sensor-actuator network where nodes might have fixed into non-moving devices in the physical world. A good example is the green house application we explained in the chapter 1 where sprinklers, screen controllers and air coolers are not supposed to move.

Chapter 6

Conclusions

In this dissertation we presented a solution addressed to a problem in the declarative interface of wireless sensor-actuator networks. This problem has two aspects. First thing is current declarative interface for actuation violates the declarative nature of SQL queries making the database abstraction useless. Secondly when used in distributed in-network actuation tasks, current actuation queries are highly inefficient due to the multiple query usage for a single actuation task.

On addressing these issues we made enhancements to the existing virtual database table and introduced new actuation query syntax to the declarative interface facilitating to perform actuation tasks in a more declarative way. To solve the inefficiency of performing actuation tasks we introduced a new execution strategy for our proposed query syntax. Our evaluations show that in this new way of performing actuation tasks, queries can be easily written and actuation tasks are performed in less time when compared with the traditional approach. Additionally the resource usage of the network in terms of the number of messages exchanged is lesser in this new strategy than the traditional method. However the support for mobility of sensors and actuators in the network in the proposed query execution strategy is left as a future work.

Bibliography

[1] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: research challenges," *Ad Hoc Networks*, vol. 2, no. 4, pp. 351 – 367, 2004.

[2] M. M. Astrahan and D. D. Chamberlin, "Implementation of a structured english query language," *Commun. ACM*, vol. 18, pp. 580–588, October 1975.

[3] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Proceedings of the Second International Conference on Mobile Data Management*, ser. MDM '01. London, UK: Springer-Verlag, 2001, pp. 3–14.

[4] D. D. Chaudhary, S. P. Nayse, and L. M. Waghmare, "Application of wireless sensor networks for greenhouse parameter control in precision agriculture," in *International Journal of Wireless and Mobile Networks*, ser. IJWMN, 2011.

[5] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[6] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," *Local Computer Networks, Annual IEEE Conference on*, vol. 0, pp. 455–462, 2004.

[7] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker, "The sensor network as a database," 2002.

[8] P. E. Guerrero, D. Jacobi, and A. Buchmann, "Workflow support for wireless sensor and actor networks: a position paper," in *Proceedings of the 4th workshop on Data management for sensor networks: in conjunction with 33rd International Conference on Very Large Data Bases*, ser. DMSN '07. New York, NY, USA: ACM, 2007, pp. 31–36.

[9] M. Karpiński and V. Cahill, "Stream-based macro-programming of wireless sensor, actuator network applications with sosna," in *Proceedings of the 5th workshop on Data management for sensor networks*, ser. DMSN '08. New York, NY, USA: ACM, 2008, pp. 49–55.

[10] N. M. Laxaman, M. D. J. S. Goonatillake, and K. D. Zoysa, "Tikiridb: Shared wireless sensor network database for multi-user data access," *IITC*, 2010.

[11] F. L. Lewis, "Wireless sensor networks," in *Smart Environments: Technologies, Protocols, and Applications*, New York, NY, USA, 2004.

[12] Y. Lin and S. Megerian, "Low cost distributed actuation in large-scale ad hoc sensor-actuator networks," *Wireless Networks, Communications and Mobile Computing, International Conference on*, vol. 2, pp. 975–980, 2005.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: An acquisitional query processing system for sensor networks," *ACM TODS*, 2005.

[14] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 491–502.

[15] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, ser. WSNA '02. New York, NY, USA: ACM, 2002, pp. 88–97.

[16] M. Mazo and P. Tabuada, "On event-triggered and self-triggered control over sensor/actuator networks."

[17] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz, "A distributed coordination framework for wireless sensor and actor networks," in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, ser. MobiHoc '05. New York, NY, USA: ACM, 2005, pp. 99–110.

[18] MySQL, "Mysql database server," http://www.mysql.com/.

[19] A. Nayak and I. Stojmenovic, *Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication.* John Wiley and Sons, Inc., 2010.

[20] E. C. H. Ngai, M. R. Lyu, and J. Liu, "A real-time communication framework for wireless sensor-actuator networks," in *in Proc. of the IEEE Aerospace Conference, Big Sky*, 2006.

[21] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on.* IEEE, 2006, pp. 641–648.

[22] A. Sayakkara, W. Senanayake, K. Hewage, N. Laxaman, and K. De Zoysa, "The deployment of tikiridb for monitoring palm sap production," in *Real-World Wireless Sensor Networks*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6511, pp. 182–185.

[23] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Rec.*, vol. 31, pp. 9–18, September 2002.

[24] X. Zeng, R. Bagrodia, and M. Gerla, "Glomosim: a library for parallel simulation of large-scale wireless networks," in *Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on.* IEEE, 1998, pp. 154–161.