Distributed information extraction from large-scale wireless sensor networks

Gaura, E. , Brusey, J., Halloran, J. and Daniel, T.

Chapter 10
Distributed Information Extraction from Large-scale Wireless Sensor Networks

Elena Gaura, James Brusey, John Halloran, Tessa Daniel

10.1. Introduction

The increase in availability and affordability of wireless technology has led to a proliferation of large wireless sensor networks (WSNs) with increasing numbers of nodes deployed to resolve complex "informational" problems. Typically, however, nodes in these networks have limited resources including energy (given the limited battery power), memory, computing power and communication bandwidth. The coupling of high complexity global tasks and the reduced resource nodes make WSNs a rich research domain. The use of WSNs has been explored since the 1990's, in a number of application domains with deployments ranging from scientific research to battlefield surveillance for the military. Some early examples of successful deployments include habitat monitoring applications [1, 2]; agricultural monitoring [3]; healthcare [4]; disaster management and detection [5] and military applications [6]. Moreover, some newer applications involve the integration of multiple WSN systems spanning a variety of disciplines. The High Performance Wireless Research and Education Network [7] and ROADNet [8] projects, both in California, are two examples of early days multidisciplinary applications.

Although specific application requirements may differ from one WSN system to another, essential to all WSN systems is the ability to acquire data and convert this to usable information in order to fulfill the application needs. Many of the WSN information extraction approaches currently in operation are based on applicative query mechanisms where requests are initiated via queries written in an appropriate declarative language, posed to the network, and data or information generated as a result. Data processing in such applications usually follows one of two approaches: centralized or distributed. For some systems, resources are not as severely constrained; hence, minimizing energy usage is not a major concern. In such unconstrained systems, a centralized approach to processing is often used where sensed values are pushed to a

Elena Gaura
Cogent Computing Applied Research Centre, Coventry University, UK

Sergey Y. Yurish (ed.), *Modern Sensors, Transducers and Sensor Networks*

power-rich location for processing, which may involve cleaning and querying the data as part of more in-depth analysis.

In a large number of systems, however, constraints like computing and battery power dictate that applications be developed with energy-efficiency in mind. For many of these applications human intervention can be both time-consuming and expensive, for example, in terms of changing batteries or adding new nodes. Therefore, a key design objective is to extend the lifetime of the network as much as possible. It is well understood that power usage costs in WSNs are dominated by communication as opposed to computation [9-11]. Therefore, techniques that promote a decrease in communications while using in-network data reduction have been identified as key to creating more energy-efficient WSN applications. Distributed processing is proposed as a method that promotes processing of data on nodes in the network [10, 12]. This in-network processing takes advantage of nodes' processing power and may include techniques like data aggregation, fusion or elimination of redundant values through filtering [13, 14]. The net result is more energy-efficient applications since data transmission is reduced in exchange for in-network computation. While this is apparently of obvious benefit towards long-life systems design, in-network computation is still in its infancy.

Existing information extraction procedures (whose survey is at the core of the chapter) can be categorized into three main approaches: agent-based, query-based and macroprogramming. Of the three, query-based systems are the most popular mainly because they provide a usable, high level interface to the sensor network while abstracting away some of the low level details like the network topology and radio communication. They are very useful and provide a solution in cases where data needs to be retrieved from the entire network. With this ease of use, however, come a number of limitations.

The first limitation is in terms of what queries can be posed to the network. In general, the query-based applicative systems in use allow the issuing of restricted queries, ranging from those targeting raw sensor readings on nodes in the network to those requiring the computation of simple aggregates like average, maximum, and minimum over some attribute of interest, for the entire network. Second, the query languages used cannot easily express spatio-temporal characteristics, which are an important aspect of the data generated in WSNs. Third, it is quite difficult if not impossible to construct information requests that represent higher level behaviour, involve just a "subset" of the network (whether physical or logical), or require more complex in-network interactions between "subsets" in order to generate information. Furthermore, as distributed computation is not the main focus of SQL-based query languages (which support most WSN query-based approaches) implementing arbitrary aggregation, for example, is quite difficult [15].

In contrast, macroprogramming has been proposed as an approach to information extraction that provides a more general-purpose approach to distributed computation compared to traditional query-based approaches. As applied to WSNs,

macroprogramming approaches focus on programming the network as a whole rather than programming the individual devices that form the network. Many macroprogramming systems provide the ability to create programs that represent higher-level behaviour, a level of abstraction beyond that of the more popular query-based approaches. Global behaviour can be specified, programmed and then translated to node level code. Ideally, the programmer is not concerned with low level details like network topology, radio communication or energy capacity.

Of interest with some macroprogramming systems are the application-defined, in-network abstractions (some based on local node interactions) that are used in data processing. One example is the Regiment system [16] in which a programming abstraction called a region is used. A region is described as a collection of spatially distributed signals with an example being the set of sensor readings from nodes in a geographic area. Regions as opposed to individual nodes are programmed (for example, an rfold operator is used to aggregate the values in a region into a single signal which can then be communicated to the user or used in further computation).

Macroprogramming, however, still presents a number of challenges. First, creating a powerful macroprogram requires a learning curve for the programmer. Expressing high-level requirements are not necessarily as intuitive to the user as perhaps SQL-based approaches are. Second, although proposed as an approach that eliminates the need for node-level programming, many of the current macroprogramming systems provide node-specific abstractions, undermining the rapid development and productivity advantages macroprogramming is meant to provide. Finally, because of the wide semantic gap that exists between the high level program and the node level code, compiler construction is quite challenging. The code generated as a result of compilation has to cater for not just computation but node-level communication as well.

A third approach to information extraction looks at the extracting of information in a network-aware manner and tailors the mechanism to the type of information needed and the configuration of the network it needs to be extracted from. These models use agents to perform tasks, make decisions and collaborate to achieve more complicated tasks. An agent is simply a piece of software that performs a task without the need of user invocation to function. Many agent-based models are multi-agent systems in which multiple agents sometimes coded to function in different ways, collaborate, coordinate and organize to perform complex tasks and generate information. These models therefore introduce expressiveness and flexibility in terms of what functions they can facilitate in the network as well as in facilitating the ability for agents to make decisions while in the network. Agents can act autonomously, multiple agents can run on a node at the same time, and multiple applications can co-exist in the network. Mobile agents can move, clone themselves and act to deal with unexpected changes in the environment [17, 18].

Although attractive, in theory, the agent-based approach presents a number of challenges particularly in terms of the difficulty they present for non-expert users to design and program. First, given the complexity of distributed systems, it is a challenge to model

accurately what interactions will be taking place between components in the network when designing agents. Second, when implementing the agents, particularly mobile agents, the code must be able to run in unpredictable, resource-constrained environments. Forecasting accurately exactly what conditions will exist in the network is difficult and therefore difficult to program for. Third, there is difficulty in testing and debugging, given the distributed nature of the system. In mobile agent systems, tracking and understanding execution in the face of agents moving from node to node makes debugging and testing even more of a challenge. Given these difficulties, practical implementations of agent-based systems, particularly multi-agent and mobile agent systems, are not as widespread as query-based systems are. Each of the three approaches above is suitable for a number of applications. They each have strengths but pose challenges as well. The agent-based approach provides a high degree of expressiveness and flexibility as do the macroprogramming approaches, but they both are more difficult to implement into deployable WSN systems. The query-based systems have, as their key feature, ease of use but are limited in the types of queries that can be posed to the network. Macroprogramming tries to address this by providing powerful constructs for capturing higher level behaviour through global programs but introduce a steep learning curve to the user.

The authors here argue that it is desirable to take a hybrid approach that retains the simplicity and ease of use of traditional query-based approaches while allowing the inclusion of useful logical abstractions provided by macroprogramming approaches to facilitate construction and resolution of more powerful queries. Such an approach would need to incorporate some of the principles of agent-based systems, such as collaboration and decision making in the network, in assisting in query resolution. In this context it can be hypothesized that end-users of WSN applications could be provided with the ability to construct and pose higher-level information requests instead of simple queries requiring the collection of raw sensed values or the calculation of simple aggregates over the entire network. Complex query type constructs that allow the expression of phenomenological spatio-temporal characteristics would provide the means for exploiting the networked sensing concept at large scale. Moreover, providing the end-user with a system that produces responses to these higher level requests for information within the network instead of as a result of post-collection analysis of all data would most certainly demolish one of the largest road-blocks of the WSN technology in its route to adoption: ensuring user acceptability.

The chapter is structured as follows: Section 10.2 below concentrates on Agent-based approaches to information extraction; Section 10.3 surveys query-based systems and macroprogramming approaches and Section 10.4 looks at the open research issues the community faces with respect to information extraction, raises research question relating to the usefulness of in-network processing from an informational viewpoint and concludes the survey. In both Sections 10.2 and 10.3, example developments are identified, highlighting those that have been used in practical deployments. Wherever possible the architecture and mode of information processing (whether centralized or distributed) is identified. The key issues for discussion lie with:

- Identifying which of the methods proposed in the literature have been evaluated at implementation level. (This analysis is needed as a large proportion of the research effort is at theoretical level and not readily suitable for practical deployments.)
- Identifying which of the methods proposed in the literature make use of or support in-network information extraction. (This is important, as approaches that utilize in-network processing such as aggregation and filtering have been shown to be more energy efficient and therefore more desirable.)

10.2. Agent Based Approaches to Information Extraction

10.2.1. Agent-based Approaches and Architectures

The agent-based approach to information extraction tasks a network by injecting into it a program with some type of processing or decision making function. In this approach there is no attempt to devise ways of transmitting data to "collection" or "aggregation" points for processing, rather, the application is sent to the data instead [19]. The agent is able to collect local data and perform any necessary data aggregation. Autonomous agents can make decisions without user input. In the case of an autonomous agent, which also happens to be mobile, that decision may involve whether the agent should move to a particular node and, if yes, which node it should migrate to. Autonomous mobile agents moving from one node to another can be designed to be cognizant of issues that may exist in the network [20]. For instance, they can make decisions, perhaps to clone themselves, if necessary, to avoid faulty regions or nodes in the network. If an agent detects that a node has a problem and is unreachable it can adapt its route so that it avoids the faulty node [21]. The mobile agent therefore presents added flexibility in terms of decision-making and reliability in dealing with faults in the network.

[22] report three architecture models for agent-based wireless sensor networks. The classification is based on the number and type of agents being used and the mode of operation of the agent-based system.

In the first model, all nodes respond to a single agent usually housed at the base station or central processor. The advantage is that there is a single control point with the benefit being the ability to access the entire network and an increase in efficiency since transmissions from this central point could be better synchronized and collision reduced as a result. Collisions during transmission require a retransmission, which is quite costly. Therefore, synchronization brings with it direct energy savings. There are, however, a number of limitations to this model.

The first limitation is lack of scalability. In this model, the agent queries each node for a sensed value, which forms the input to the agent's deliberation (the agent follows a perceive-deliberate-act cycle). As the number of nodes increases, however, the time taken to deliberate also increases which can lead to time delays. Also, with larger networks it is more likely that multihop communication is needed. This can lead to even more time delays during data gathering as well as increased power consumption.

A second problem is observed in cases where there is a need for more than one agent to access nodes in the network simultaneously, for example, to execute different application tasks. Here an increase in the time delay occurs because one agent would have to wait for another to complete its task before accessing that node or set of nodes. This leads to delays in agents getting the data needed for processing and acting if needed. This model can be considered to utilize a centralized processing approach as all data is transmitted to a central point (albeit an agent) for further processing. In-network processing is absent.

In the second model, each node in the network hosts an agent that will determine how that node will behave. Control of the network is therefore distributed and agents can coordinate and collaborate to achieve goals both at a local and global level. A main advantage in using this model is its scalability. The ability to perform local computation without needing multihop messages to a central controller results in a reduction in energy usage. A second advantage is the ability of the network to perform tasks concurrently given the presence of multiple agents. However, in practice, it is difficult to create agent-based systems that solve global tasks using only local information. The result, therefore, is an increase in resource and energy usage for the necessary inter-agent communications, a cost avoided by the centralized model presented above. This multi-agent approach clearly incorporates in-network processing techniques for information generation. The “environmental nervous system”, [23] a multi-agent system, is based on this model. It is a small (three node) autonomic wireless sensor network aimed at environmental sensing for intruder detection. This system is described in greater detail in Section 10.2.1.2.

The third model uses mobile agents. Here an agent is able to migrate to a node or nodes, collecting, aggregating and fusing data before it sends a message back to the central processor. A primary advantage here is a reduction in the cost of computation since only one node is active at a time. A second advantage is that communication cost is low since migration of agents only involves one transmit and receive event. The agent in this model is responsible for both collecting data and the itinerary related to the task in operation. The main disadvantage, however, is that if the node to which an agent has migrated fails then all data gathered up to that point is lost. There are, however, techniques that could be implemented to combat this problem, for instance leaving copies of the data on previously visited nodes so that the agent could restore its state if needed. A second disadvantage is the lack of concurrency this approach brings. In large networks, in particular, this could be a problem if the agent takes too long to collect all the data required to perform the given task. The mobile agent-based deployment reported by [17] is based on this model. This network was aimed at ground vehicle classification using acoustic sensors and is described in greater detail in Section 10.2.1.1.

Each of the three models described exhibits both advantages and disadvantages. According to [22] the strengths of the three methods were simplicity in the case of the centralized approach, robustness in the case of the multi-agent approach, and efficiency in the case of the mobile agent approach. The weaknesses were identified as scalability,

complexity and latency. Two hybrid approaches were further proposed and described by [22] that combined the features of the three in ways that attempted to minimize the impact of the disadvantages described.

Much of the reported research in the area of agent-based information extraction mechanisms for WSNs describes variations of the agent approach outlined above. In the following, some examples of mobile and non-mobile agent system architectures are presented.

10.2.1.1. Mobile Agent-based Distributed Sensor Networks (MADSN)

The Distributed Sensor Network (DSN) model was first proposed by [24]. In this model, referred to as fusion architecture, all sensor data is sent to a central location where it is fused [20]. A number of variations followed [25-27], each making some improvement on the DSN architecture but holding to a common client/server paradigm or a centralized approach to information processing.

In a WSN with a large number of deployed sensors, each with its own piece of data, it is impractical to have all that information flowing to a central store. First, it has been identified that communication dominates power usage in sensor networks and that the cost of transmission of data from individual nodes in a network to a base station or sink far exceeds the cost of in-network processing of data before transmission [28, 29, 11]. With resource constraints like limited battery power on the nodes in the network, there is a need to reduce the cost of communication if at all possible. The energy availability problem is further exacerbated by the fact that often not all data that is transmitted is critical to ensuring quality information. There is often a large quantity of redundant or erroneous data. Other problems include network bandwidth limitations, unreliable connectivity and noise in the network. Given these constraints, DSN and other client/server type architectures cannot meet all the challenges of WSNs. Qi, Iyengar and Chakrabarty proposed an improved DSN architecture that used mobile agents (MADSN) [30, 17]. Whereas in DSN, sensor nodes collect data and transmit to the base station node or sink (the central processor), with MADSN the computation code is transmitted to the node where the data resides. The need to transmit masses of data is removed and replaced by transmission of only a small piece of code representing the agent. Additional benefits are that the agents can be programmed to perform fusion tasks to consolidate data increasing the extensibility of the system and the mode of information collection is more stable since it is not affected by fluctuations in connectivity. If a network connection fails, the agent can wait till it is re-established before returning its results.

A comparison between MADSN and DSN showed up to 90 % savings in data transfer for MADSN over DSN due to avoiding transfer of raw data [30]. It is not clear however if this was despite the overhead introduced by having agents created and dispatched. In another study, a model that incorporated collaborative signal and information processing (CSIP) techniques was applied and an analysis of performance between MADSN and

DSN conducted to measure energy consumption and execution time. Again, in that study MADSN outperformed DSN [31].

[32] studied the use of mobile agents for data fusion in a WSN running MADSN and focused on creating an optimum design of the itinerary component of the agent to improve performance and minimize resource usage. [33] showed that MADSN could be successfully applied to the real-world problem of vehicle classification in an unattended ground sensor system. The study showed, however, that classification accuracy was low (about 23 %) when only one sensor was used but improved to about 80 % when a multisensor array was used. (Target classification accuracy with a single sensor was high (about 75 %) only when the target was in close proximity to the sensor.)

10.2.1.2. Autonomic Wireless Sensor Networks (AWSN)

[23] introduced the concept of the autonomic wireless sensor network. They proposed that mobile agents could be deployed in the network to facilitate autonomic computing. Autonomic computing implies the ability to self-manage, self-maintain as well as interact with a user of the network at a policy rather than hardware level. The purpose of interaction would be to allow interoperability with legacy systems if needed. Mobile agents in that context would support cooperation and negotiation in the system via relevant protocols and a suitable agent communication language. The mobile agents would introduce flexibility in terms of the ability to modify agent code through agent migration or agent adaptation. The idea here is that an agent could evolve as system policies evolved. In modeling autonomic behaviour, an AWSN also uses the self-knowledge, self-protection and self-interest characteristics inherent in mobile agents of multi agent systems.

Autonomic behaviour in Marsh's system includes the ability to handle sensors that have been rendered inoperable due to some type of damage, and the system can put into effect strategies to deal with that. For instance, the authors suggest that the system could make an estimate of data lost due to system damage and effect nearby sensors to increase their sampling rate to compensate. It could also mean a reconfiguration of the routing topology to minimize or eliminate message loss. The AWSN concept is closely related to the multi agent system framework and incorporates some of the same ideas including that of entities migrating from node to node while making "intelligent" decisions.

[34] and [35], building on this idea of an autonomic wireless sensor network, proposed an agent-based approach to implementing intelligent power management. In their work they promote the use of agents in intelligently activating or deactivating nodes for power conservation based on interpolation. For example, the following criteria are used to decide whether a node should be put to sleep: a node is considered redundant if the remaining nodes can interpolate the temperature at its location to a desired degree of accuracy. Redundancy in this context means that the node is not needed since the other nodes or the network can operate effectively without it.

In Marsh's scenario, a coordinator agent is appointed and aggregates all calculations performed by other agents in deciding whether deactivation should occur or not. The coordinator agent sends out requests to other agents to calculate the components of the interpolation function used and send that result back to the coordinator. The coordinator then calculates the actual interpolated temperature and then requests the temperature from the appropriate agent. If the two values fall within a given error range the node is put into sleep mode.

It must be noted, however, that the issue of power management is influenced by a number of other network characteristics such as coverage, latency and longevity. Any power management scheme would have to balance the tradeoffs in making decisions on how power can be conserved. In the above example, the inherent autonomic characteristics of the network itself are used for efficiently managing power usage. Although a promising approach, this concept has not yet been implemented in agent-based WSN deployments.

10.2.1.3. Mobile Agent-based Wireless Sensor Networks (MAWSN)

[19] identified that the operation of MADSN was really based on three main assumptions. First, that the network had a cluster-based architecture, second, that each source node (nodes with data) was one hop away from the clusterhead, and third, that the redundant data collected could all be fused into one packet of fixed size [19], [36]. Those restrictions in a real-world context seemed to impose too strict limitations on the applications that could make use of the architecture and excluded many systems that did not boast all of these features. Hence, [19] proposed MAWSN as an alternative architecture to address networks where the features needed for the MADSN approach to work were not evident. Previous work done by [36] showed that in multi-hop environments without clusters, mobile agents could be employed to eliminate data redundancy by employing context-aware local processing techniques, use data aggregation to eliminate spatial redundancy with sensors that were of close proximity to each other, and reduce communication overhead by combining tasks. They were able to reduce information redundancy at three levels: the node level, the task level and the combined task level.

At the node level mobile agents were assigned processing code specifically targeting the requirements of the specific application. The result was that the mobile agent only needed local processing of that data requested by the application thereby reducing the amount of data transmitted as only relevant data was extracted from nodes for transmission. At the task level, the mobile agent aggregated sensed data from individual nodes when visited. Finally, at the combined task level, the mobile agent used the packet unification technique to unify shorter data packets to one longer packet again reducing the number of transmissions. MAWSN built on the encouraging results in [36] and experimentally showed improvements in energy consumption for data packets transmission over the more traditional client-server based WSN.

Concurrently with the above work, [37] proposed a framework aimed at implementing a wide variety of sensor network applications. Their aim was to create an architecture that allows multiple applications to share a network and permits scalable deployments that can evolve over time. In their design autonomous agents are deployed in the network with a particular agent having the ability to be on one or more nodes. Their framework is built atop TinyOS and its associated virtual machine Mate [38] and agents run on Crossbow motes. The agents were shown to use resources only on those motes they visited so resources in the network were used more efficiently. Agents were able to read sensor values, start devices and sent radio packets if required.

In a data collection experiment aiming to find the maximum value in a field of sensor readings, an agent was injected into the network via broadcast. After arriving at a node, it re-broadcasts itself if the reading at the current node was greater than that at the previous node. Agents stopped propagating once the region with the maximum sensor reading was reached. This was done by incrementing a count value when an agent arrived in the area. Once that value went over a pre-defined threshold the last agent to enter the area could issue a notification message to the processor.

Analysis of the experiment showed that as the size of the network increased, a smaller proportion of the network needed to be active. This meant that in large networks agents stayed on nodes only long enough to perform their task and quickly release resources for other task execution. The conclusion of the experiment was that the agent model presented was successful in showing that it could scale up without having to occupy all nodes on the network in performing tasks; however some areas for future work were identified. The first is security to ensure that unauthorized agents are not able to mount energy-depletion and denial-of-service attacks among others. The second involves giving agents the capability to query nodes for information such as processing power and energy. The third area involved giving agents the ability to cache their code on a node to reduce the number of agent transmissions.

In a similar approach [39] describe the agent oriented programming paradigm for development of intelligent sensor networks. They implemented a test application using the Java Agent Development Framework (JADE) [40] aiming to develop an intelligent ground sensor network. The application consisted of an Unattended Ground Sensor Network (UGSN) used to monitor moving targets with agents analyzing the data being acquired by nodes. In the experiment, information from the sensors were correlated and fused by reasoning agents using a fusing algorithm (the majority voting ordinary technique) suited for distributed information fusion. A special agent called the Target Agent (TA) is created on the node where the first sensor trigger occurs. The TA then reasons with its neighbours and after correlating their sensor data, decides what node it should migrate to next. The objective of migration is to get closer to the target being tracked. After each move the TA performs data fusion using the new data available on its host node. Through the TA, the network as a whole, decides when external applications need to be updated and what information should be sent to those applications.

The experiment was successful in its sole aim of showing that agents were suitable for distributed information fusion. Distributed information fusion is the idea, quite similar to aggregation, that data generated by nodes in a network can be efficiently combined in-network instead of transmitting all raw data to a sink for processing. The main advantages of this approach were identified as: reduction in data redundancy since we have agents deciding whether data is important enough to be used; reduction in power consumption in having all data sent directly back to a central location; conservation of communication bandwidth by limiting the exchange of data between nodes as much as possible.

The advantages of agent-based distributed information fusion parallel those exhibited by other systems, particularly query-based systems that incorporate in-network aggregation.

10.2.1.4. Multi-Agent Systems

Although the idea of using agents in wireless sensor networks is widely supported by the literature [27, 31, 41, 32, 36], the actual use of multi-agent systems in WSNs is not as widespread as one might expect [42]. There are of course the big problems of deployment, testing and debugging of such systems on what is essentially a distributed application with minimal interfaces to support user interaction. There have been real deployments using agents [20, 37, 43, 36] but multi-agent system deployments are still rare. [42] propose a methodology for multi-agent deployment. They start with a base station implementation for the agent that leads to a distributed implementation where agents are mapped to nodes, using either a one-to-one, many-to-one or one-to-many mapping. This mapping is used to model the interaction expected between agents and the base station, as well as iron out communication and interaction rules between agents. These first two stages are carried out on a device with more processing power like a laptop or desktop. In the third and final stage, the statements and rules governing agent behaviour are translated to the language of the WSN system hardware that will be hosting the agent. The result of the last stage is a topology in which the load of executing an algorithm is distributed among the agents, which can allow faster response times for complex algorithms. In addition, it can also result in a reduction in the number of transmissions in the network since an agent on a node or on a neighbouring node can now process packets that previously had to be routed to the base station for processing.

In the examples described above, agents are pieces of code injected into the system, gathering data, making decisions and migrating between nodes in some cases. Some research, however, has investigated the concept of having nodes themselves act as intelligent agents. [44] use intelligent agent theory to model sensor behaviour, viewing the WSN as a network of distributed autonomous nodes with the capability of making decisions. They also propose using artificial intelligence strategies on nodes to enable decision making, selecting a rule-based expert system that makes use of locally collected information on a node to make step-wise decisions. The concept, however, has not yet been implemented and deployed.

10.2.2. Agent-based Middleware

In general terms, middleware sits between the operating system and the application with its main function being to provide support for development, deployment, execution and maintenance of sensing applications [45]. There are a number of categories of middleware based on the objectives of the approach and the way in which the wireless sensor network is viewed. With some, the main goal is to provide a dynamic reprogramming capability while others seek to provide a platform-independent model on which sensor network programs can be written and executed. Other approaches try to provide a cross-layer management approach and provide functionality for manipulating other services like routing, etc. Information extraction, which is ultimately the goal of the sensor network applications built using these models, is supported by the services provided by the particular middleware approach. Approaches include the use of virtual machines which in many cases are application-specific but reduce the amount of code that has to be transmitted and therefore reduces the cost of communication, as well as modular programming approaches which allow easier and more efficient reprogramming of sensor nodes [46]. Some agent-based middleware will be described below.

10.2.2.1. Mate

Virtual machines (VMs) provide one middleware solution to the problems posed by wireless sensor networks. In WSNs the greatest drain on energy resources is the cost of communication, so if the traffic needed to reprogram motes could be reduced this could lead to a longer network lifetime as well as the ability to reprogram more frequently. Mate is a middleware approach that abstracts high-level operations into virtual machine bytecodes (see Fig. 10.1). As a result a VM program can be very short, bytes long instead of kilobytes. Mate, therefore, is a virtual machine designed for sensor networks [38] and forms the basis for many agent-based information-gathering applications.

Mate is a bytecode interpreter, running on TinyOS and provides a high-level interface for creating small (< 100 bytes) but complex programs. Mate is focused on energy conservation in transmission of code and breaks code into small sections called capsules (referred to as agents by some), each 24 bytes long. Programmers write applications and the system injects them into the network using certain algorithms that minimize energy and resource usage. The small size of the modules makes it easier to distribute into the network. Mate specifically targets the constraints of limited bandwidth and power by allowing adjustments or reprogramming of capsules before they are issued to the network.

Mate is a general architecture that allows a user to create a wide variety of virtual machines. A user builds a Mate VM in three steps [45]: a) the user selects a language that defines a set of VM bytecodes representing the basic functionality; b) the user selects the execution events. Each event has its own execution content which runs when an event is triggered; c) the user selects the primitives to be used. These are operations that provide functionality beyond that of the selected language.

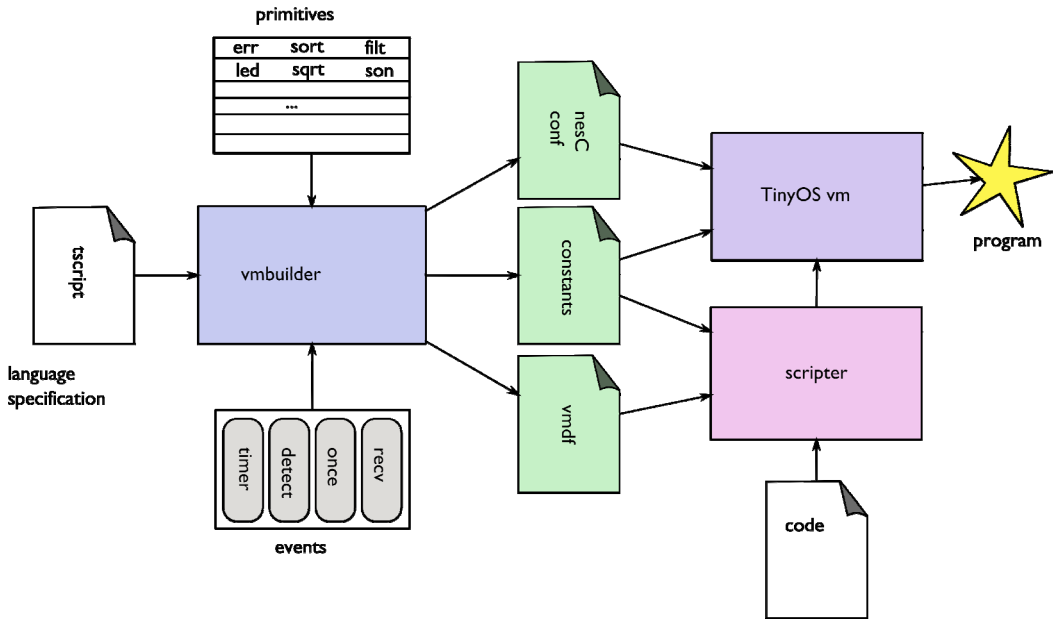

Fig. 10.1. Process of generating a program using Mate.
Revised image used with permission of P. Levis [47].

A set of files is generated after these steps have been executed. Some will build the VM that will run on the motes, others will build a scripter program with information on the language and primitives selected. The user can now construct programs in the scripter that will be compiled down to the VM-specific binary code. These bytecodes are then issued to the network and processed by a VM there, which will execute it.

To reprogram motes, the user needs only add one mote with the new capsule to the network. When a mote hears a new capsule it immediately starts forwarding it until every mote in the network has been updated.

The advantages of Mate, therefore, are the simplicity of programming for the sensor network as well as the ease with which the network can be reprogrammed. Mate's authors have identified future work as being the creation of propagation algorithms for larger programs, security and scripting languages that can be compiled to a Mate VM.

10.2.2.2. Agilla

Agilla is a mobile agent middleware based on Mate that allows quick deployment of adaptive applications [18], [48]. Instead of creating capsules that are flooded throughout the network, Agilla allows the user to create mobile agents that can be injected into the network. Agents are dynamic and intelligent, and can coordinate and collaborate locally or through remote invocation. These agents can clone themselves and move from node

to node as they perform tasks. Agilla affords more flexibility because it allows applications to decide how agents in the system should migrate and spread in the network while maintaining their code and state.

In the Agilla Model, each node can have a maximum of four agents and addressing is done using the geographic location of nodes instead of IDs. Agilla can therefore be used in running applications over geographic regions and to allow geographic routing for any multi-hop interactions [18]. Inter-agent coordination in Agilla is facilitated using a tuple space and an acquaintance list, which are maintained on each node. Each node's tuple space is shared by local agents and remotely accessible. Global tuple spaces are not supported due to the energy and bandwidth constraints, rather separate local tuple spaces are maintained by each node and agents can access remote tuple spaces using special instructions provided in the middleware. Each node also maintains an acquaintance list containing the location of all one-hop neighbours and local agents can access it via a number of provided functions. Fig. 10.2 shows the programming model for the Agilla system.

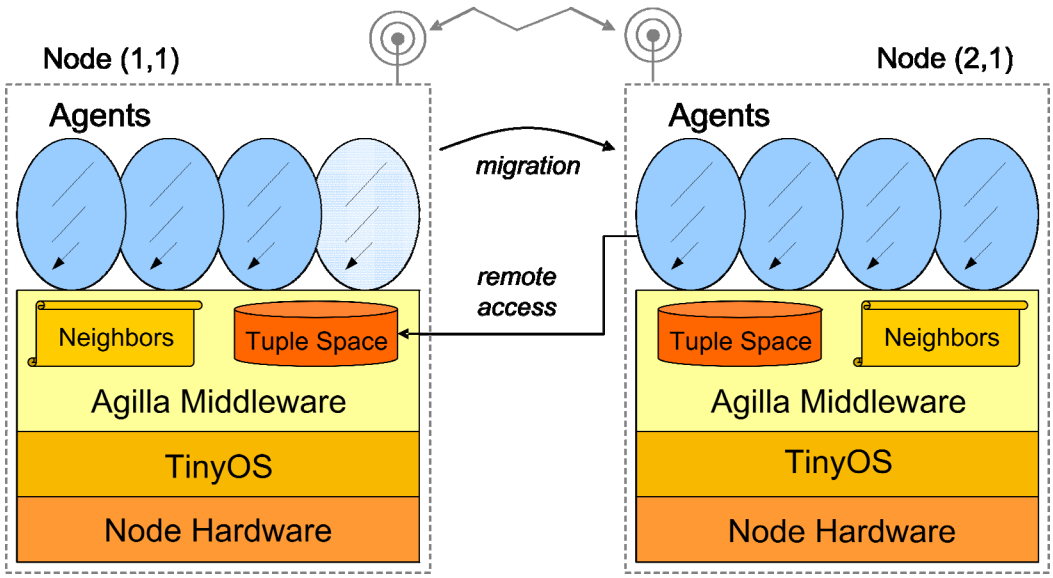

Fig. 10.2. The Agilla programming model. Original image used with permission from C-L. Fok [48].

Fok [48] used Agilla to successfully deploy fire and tracker agents in a fire tracking application and demonstrated the reliability and efficiency of the exercise in this case study [18]. The scenario is as follows: a fire starts in a region of the network and as it spreads tracker agents crowd round it repeatedly cloning themselves until a perimeter is formed. Once that perimeter is formed a fire fighter is informed. The fire fighter injects a guidance agent who is responsible for guiding the fire fighter along a safe path to the

fire. The study focused on the tracker agent, leaving the guidance agent and the development of a safe-route algorithm to future work.

Two types of fire modeling agents were used, static fire agents and dynamic fire agents. Static agents simply insert a fire tuple into the local tuple space and then repeatedly blink a red LED (visual indicator of network state). These agents are used to create fires of different shapes. The dynamic agent models a spreading fire. It inserts a fire tuple upon arrival at a node, blinks the red LED a number of times, clones itself onto a random non-burning neighbour and repeats the blinking. The cloning and blinking process is repeated until every node is on fire. The rate of spread can be controlled by the number of times a node blinks between cloning operations.

A fire-tracking agent is responsible for discovering a fire and forming a boundary around it. The tracker agent inserts a tracker tuple when it arrives at a node. Other tracker agents will use this tuple to check whether a neighbouring tracker agent is still present on the node. If the node a tracker agent is on catches on fire the tracker agent dies. A reaction is registered when this occurs, the tracker agent will turn off all LEDs, remove its tuple and stop functioning.

The life cycle of a tracker agent is as follows: it repeatedly checks whether any of its neighbours are on fire. If there are none, it moves to a random neighbour and repeats the check. If, however, a neighbour is on fire it enters a tracking mode and lights up its green LED and repeats the following. It determines the locations of all neighbours that are on fire and for each non-burning neighbour within a certain distance of the fire clones itself to those nodes. This process is repeated until the fire dies. Periodically checking for neighbours close to the fire allows the agent to adjust the perimeter. Once a fire was started, a tracker agent was injected next to it. This allowed the investigators to focus solely on the efficiency of perimeter formation and not on fire discovery as well.

The experiment was successful in a number of ways. First, it demonstrated the use of Agilla in deploying complex applications. Second, it showed that multiple applications could share the network at the same time (fire-simulation and tracker application). Third, it showed that mobile agents can be used to successfully program WSNs. Experiments on a 26-node MICA2 network showed that tracker agents each 101 bytes long were able to form a perimeter around a static as well as a dynamic fire. It was also evident that the efficiency of perimeter formation, in the case of static fires, depended to a great extent on the amount of agent parallelism in the system. The authors have identified areas of future work aimed at allowing new instructions to be added to the network after deployment and also allowing the instruction set to be customized depending on the agent being deployed. [18] describes the experiment and its results in greater detail.

10.2.2.3. Impala

Impala can be considered another mobile agent-based middleware approach. Its middleware architecture allows a user to create modular, adaptable and maintainable

applications for WSNs. The adaptation facilities allow changes to be made at runtime in an effort to improve energy-efficiency, reliability and performance of running applications. The idea is that given the need for long-term management of a sensor application, a middleware layer that can update and adapt applications dynamically, switch to new protocols easily at runtime is a big advantage. Impala, therefore, is a middleware layer that acts as an operating system, resource manager and event filter upon which applications can be installed and executed [49]. It was specifically designed for a wildlife-monitoring project, ZebraNet, also detailed in [49].

The system architecture is depicted in Fig. 10.3. The architecture's upper layer contains all application protocols and programs for ZebraNet. In the lower layer are three middleware agents: Application Updater (AU), Application Adapter (AA) and Event Filter (EF). The AU receives and transmits software updates to the node via the wireless transceiver, the AA adapts the protocols to different conditions at runtime in a bid to improve energy-efficiency and performance and the EF collects and sends events to the system for processing.

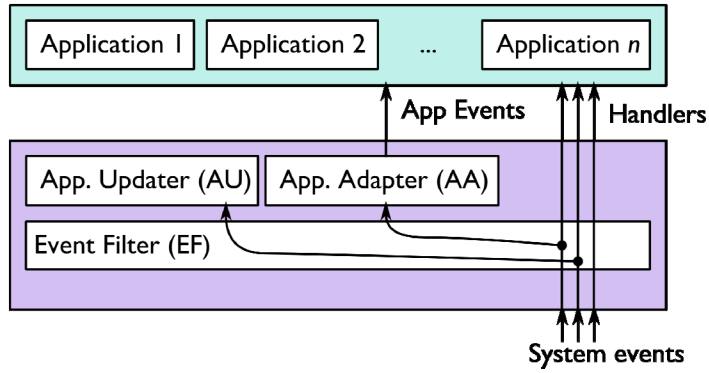

Fig. 10.3. Architecture of the Impala system [49].

A program module is compiled into binary code before it is injected into the network. Linking is performed by the updater on every node and when all modules in an update have been received the module is linked to the main program. Modules are linked independently. Agents work autonomously making the network more fault-tolerant and better able to self-organize. It is not suitable, however, for resource-limited systems.

Impala uses an event-based programming model where the AA, AU and applications are programmed as a set of event handlers called by the EF when appropriate events occur. The EF controls various operations and starts a number of processing chains including timer, send, done data and device events. Based on different scenarios such as improving energy efficiency or performance, the AA will adapt a particular application. The AU is then responsible for effecting software updates while being cognizant of such issues as resource or bandwidth constraints and node mobility. If, for example, the radio

transceiver on a node was to fail, the EF would call on an appropriate handler in the AA. The AA would determine the impact of that failure and decide whether an updated or new application that takes into account these issues should be issued to the network. The AU would then be responsible for carrying out the software update itself.

10.2.2.4. SensorWare

Although not strictly considered mobile agent architecture by some, SensorWare [50] is based on a scriptable lightweight run-time environment suited for nodes with energy and memory constraints. It is formally called an active sensor framework (ASF) where mobile scripts less than 180 Kbytes are used to make the network programmable and open to users.

The difficulty in designing an ASF has been identified as determining how to accurately define the abstraction of the run-time environment. Any abstraction would have to result in compact code, resource sharing, multiple users being able to access the system and portability to different platforms. With these in mind they designed a node abstraction that allows multiple users access to the modules on a node while still being able to create new modules.

The authors focused on defining a framework that allowed the description and deployment of distributed algorithms for wireless ad-hoc sensor networks. SensorWare's language model can effectively express these algorithms while simultaneously shielding the user from low level details while allowing sharing of node resources among several concurrently running applications or many users. The language model focuses on the properties of efficient algorithms for sensor networks while application development for a real network is underway. Fig. 10.4 shows the architecture of the SensorWare system.

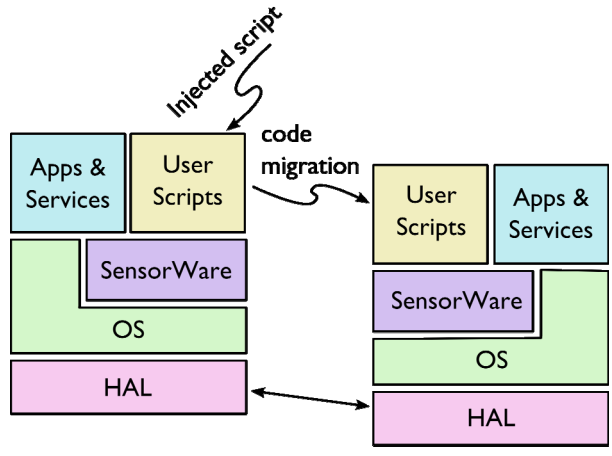

Fig. 10.4. Architecture of sensor nodes running SensorWare [50].

SensorWare is also quite effective at dynamically deploying the distributed algorithms represented in the language. Given that the nodes are memory-constrained and cannot store every application in local memory the method of dynamic deployment has to be effective. Rather than each node being programmed by the user the SensorWare approach gets the nodes themselves to program their neighbours. The user injects the program into the network and the program is autonomously distributed to the nodes that should receive it. SensorWare is described in detail in [51].

10.2.2.5. TinyLIME

Unlike many of the examples described above that use a single controlling processor or sink node for data collection and processing, TinyLIME instead promotes a distributed approach. Multiple mobile clients or agents are distributed with the ability to receive data only from the sensors that they are directly connected to. The clients can share this locally collected data through their wireless interconnections [52]. TinyLIME is built on the original Linda in a Mobile Environment (LIME) architecture [53] and deployed on Crossbow motes. All base stations run an instance of moteAgent, a LIME agent that among other functions maintains data freshness and historical information on recent sensed values. TinyLIME also supports data aggregation both over values sensed by multiple sensors and those sensed by a single sensor.

10.2.3. Remarks

With respect to agent-based approaches to information extraction, it is forthcoming from the survey that many of the described techniques and architectures supporting them have not yet been sufficiently developed and evaluated other than theoretically and are, hence, far from being deployable (or even successfully implementable) on resource constrained nodes such as those found most commonly in WSN applications.

It is, however, encouraging to see the wealth of research and proposals related to agent, multi-agent and mobile agent systems, leading, no doubt, in the future, to energy efficient, hardware independent, robust methods of relaying information drawn from large scale WSNs, to the user.

10.3. Query-based and Macroprogramming Approaches

Having examined agent-based approaches in the last section, this section surveys both query-based and macroprogramming-based systems, with a view to first identify which of the methods proposed in the literature have been evaluated at implementation level. (This analysis is needed as a large proportion of the research effort is at theoretical level and not readily suitable for practical deployments.) Second, looking out to common constraints shared by WSN systems (such as limited energy and communication resources for example), it is aimed here to identify which of the methods proposed in the

literature make use of or support in-network information extraction. (This is important, as approaches that utilize in-network processing like aggregation and filtering have been shown to be more energy efficient and therefore more desirable).

10.3.1. Query-based Information Extraction

Many researchers have taken the view that the sensor network can be considered a database from which information has to be requested and retrieved. The reasons for this are as follows: first, the network is a collection of data albeit that data is dispersed across multiple nodes. Second, data needs to be retrieved from the network, and like a conventional database one needs to be able to query the network for what is desired and get a response. Third, the nodes in the network are of interest primarily in as far as the data they generate, this can also be termed a data-centric view of the WSN.

Many of the sensor network databases use a query language similar to the Structured Query Language (SQL) for constructing queries.

SQL is the most popular language used for creating, modifying and retrieving information from relational databases. It consists of a number of clauses including SELECT - FROM - WHERE and GROUPBY clauses that allow selection, join, projection and aggregation respectively. SQL-type queries constructed for WSNs are quite similar to those of traditional SQL both syntactically and semantically. The FROM clause, for example, can be used to either specify sensors or data stored in tables. The GROUPBY clause allows a SELECT statement to collect data across multiple records and group the results by one or more attribute values. In the WSN querying context, for example, records can be grouped by the node's id or by locations, etc. if these are attributes retrieved in the query.

In traditional database systems, data is accessed via an application or directly via a front end. Such applications insulate the user from the inner workings of the database system while allowing easy and meaningful queries to be applied to it in retrieving the necessary information. In a similar way, the sensor network can also be interfaced using a suitable application. The application would act as a query processor-type interface to the network taking into account the power and computation resource limitations on the devices in the network. According to [54] the challenge in query processing is not in processing data as quickly as possible but rather in figuring out a way to effectively respond to queries while transmitting as little data as possible. Several researchers have noted the benefits of this query-processor interface approach.

[55] describe a view of the network as a database with two methods for processing queries issued to the network: warehousing and distributed approaches. With the warehousing approach the processing of a query is separated from the interaction with the network itself. Essentially, the data is extracted and stored at a centralized location for processing. Queries in this case are pre-defined and usually ask for aggregate historical data, for example, "for each rainfall sensor, display the average level of

rainfall for 1999" [57]. This model really mirrors a client/server approach to data collection as is found in a traditional distributed network [56].

There are two main disadvantages to this approach, however. First, it is not well suited to requests for continuous data, either because it is not possible to retrieve the required data from the node or because data is not retrieved frequently enough within the network to be able to answer a query [57]. Second is the high-energy consumption that occurs when transmitting large quantities of data from the node to the centralized database and even more so in the case of a continuous stream of data, making the process very energy-inefficient [57, 55].

Alternatively, [57, 55] propose a view of the network as a set of distributed databases where the composition of the query in effect determines what data is retrieved. Such queries have the advantages of efficiency because only what is required is actually retrieved, as well as flexibility since various queries can be formulated depending on what information is needed. This approach also takes advantage of the processing power on the node itself and where possible processes queries or parts of queries on them.

[58] note that data generation and routing in the sensor network can be seen as similar to the concepts of data storage and query processing in traditional databases and so also view the sensor network as a database. They examine the challenges in implementing the network as such and identify the need for robustness of data access given the possibility of node failure and noise that can affect readings. The idea is that by creating a standard querying interface that allows less restrictive query semantics, approximate results can be obtained which can help in producing an energy-efficient implementation of the "sensornet database application". [58] are also of the view that query optimization is closely linked to routing and they propose an adaptive approach that takes into account the volatile nature of data and communication in the network.

Several major query-based systems are described in detail below.

10.3.1.1. COUGAR

The COUGAR approach has been proposed in [57, 55, 58] and more fully described in [59]. It builds on the Cornell PREDATOR object-relational database system [60] and models each sensor in the network as an abstract data type while signal-processing functions are modelled as abstract functions returning sensor data. The network is viewed as a system where each node is a "mini database" holding part of the data contained by the whole. COUGAR allows the issuing of declarative queries for information requests and uses a query optimizer to plan an in-network processing strategy. Queries in Cougar are SQL-based and are posed to a virtual table called sensors with one row per node per instant in time and one column for every attribute. (The same model is used in TinyDB and is described in Section 10.3.1.5.).

A query proxy layer is placed on each node and interacts with both the routing and application layers in the system. A query optimizer is placed on a gateway node and distributes the query processing plans after it receives the queries from the user. COUGAR uses catalogue information and query specification to create the query plan that describes not only the flow of data between sensors but also how information is to be computed on individual sensors. Once the plan is complete it is sent out to all relevant nodes. During execution, data is returned to the gateway node. In addition, a query proxy can inject queries into the network from arbitrary or specified nodes as it performs high-level services.

Fig. 10.5 shows how the system executes a simple aggregate query. COUGAR is especially useful for executing continuous queries.

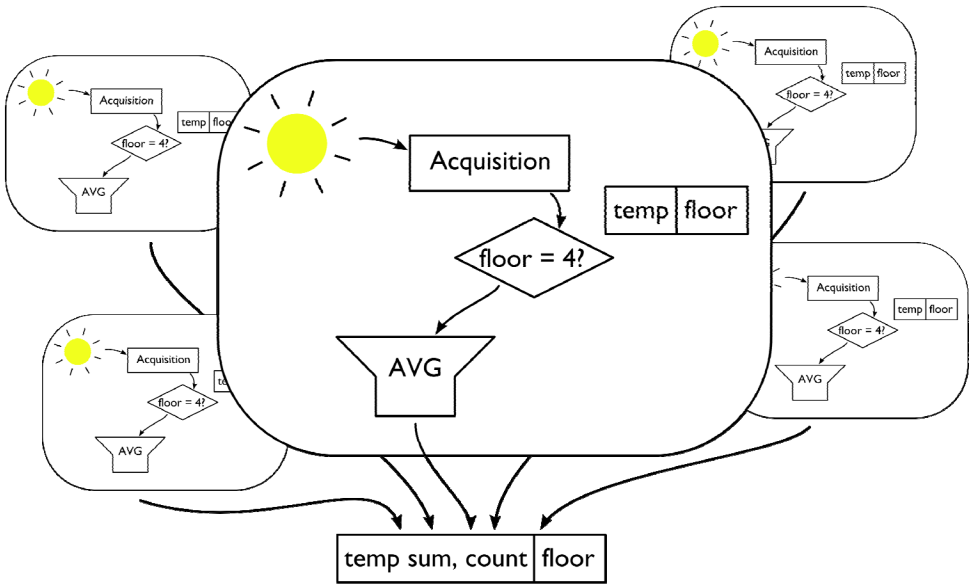

Fig. 10.5. A sensor network running the Cougar system executes a simple aggregate query [61].

A big advantage of COUGAR is that it shields the user from needing to have any knowledge of the underlying network, or how data is generated and processed. A second advantage is that it increases efficiency in terms of energy consumption since in-network processing is allowed thereby reducing the amount of data that has to be sent back to the gateway node [62]. There are other architectures that promote this distributed approach to query processing, for example, IrisNet, described in [63], SINA and TinyDB. Both TinyDB and Cougar follow a very similar query processing model and address the

resolution of both simple and aggregate queries. Neither, however, facilitates the processing of other types of complex queries such as spatio-temporal or queries that target only a subset of the network.

10.3.1.2. Sensor Information Networking Architecture (SINA)

Like COUGAR, the SINA architecture described in [64] promotes a distributed database query interface that emphasizes in-network processing and reduction in power consumption in the network. It views the network as a collection of datasheets (a spreadsheet) with each datasheet containing the attributes relevant to the node it describes. A cell in this scheme represents an attribute and the collection of datasheets form what is called an associative spreadsheet or spreadsheet database representing the network. Each cell is therefore referred to using an attribute-based naming method.

SINA also makes use of clustering of low-level information in the network to increase efficiency. Clusters are formed from aggregations of sensors based on their power levels and proximity. Recursive aggregation can also be used to produce a "hierarchy of clusters". A cluster head within a cluster can then be elected to perform key functions like information filtering, fusion as well as aggregation [64]. Fig. 10.6 shows a model of a network with the SINA middleware.

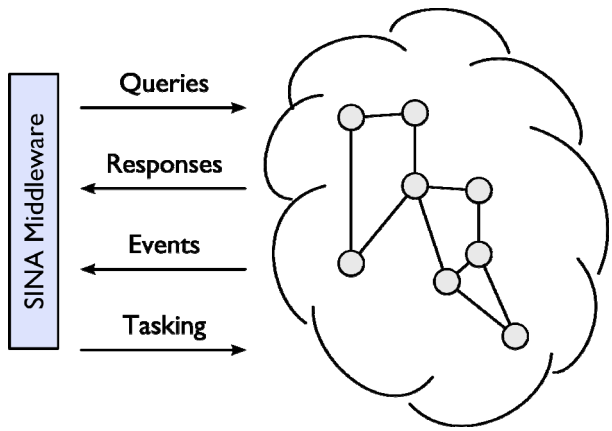

Fig. 10.6. Model of a sensor network and SINA middleware [64].

SINA uses the sensor programming language Sensor Query and Tasking Language (SQTL), which serves as the programming interface between applications and the middleware. SQTL is a procedural scripting language built from Structured Query Language (SQL) and messages written in the language can be interpreted and executed by any node in the network, although messages can target specific nodes if required. SQTL also supports the generation of information as a result of the occurrence of, or change in, some phenomena (events). Three types of events are supported by SQTL. The

first is an event generated when a node receives a message, the second is an event triggered periodically by a timer, and the third is an event triggered by the expiration of a timer.

SINA modules run on each node in the network and allow querying as well as monitoring of events. Nodes are aggregated to form clusters and elected cluster heads perform filtering, fusion and aggregation tasks. The user issues a query and the SINA architecture selects the most suitable methods for information gathering and distribution based on the type of query issued as well as the current status of the network. A node receiving the query will interpret it and request information from neighbouring nodes in evaluating it.

A number of information gathering mechanisms are employed to help reduce resource consumption and increase response quality, as follows:

1. Sampling Operation - for some applications a query may need to target the entire network in eliciting information of interest. Responses from all nodes in a network may result in response implosion [65], however, the effect can be reduced if nodes are able to make decisions on whether they should respond or not. SINA implements this "decision-making" capability by assigning each node a response probability that determines whether they respond or not.

2. Self-Orchestrated Operation - in networks with a small number of nodes it is sometimes necessary that all nodes respond in order to improve the accuracy of a result. SINA uses what is called self-orchestration in an effort to reduce the problem of response implosion mentioned before. Here each node suspends its response transmission for a particular period of time to help reduce the likelihood of collisions occurring as could happen if nodes transmitted simultaneously.

3. Diffused Computation Operation - this method is used to implement aggregation functionality in the network. Each node is first assumed to have knowledge of its immediate neighbours and aggregation logic already programmed into the SQTL scripts is used to perform aggregation before routing the result to a designated node.

SINA is therefore particularly suited for aggregate queries for replicated data in a cluster-based network configuration but does not address other complex queries like spatial or temporal queries which may also be suited for processing in a cluster-based configuration.

10.3.1.3. Data Service Middleware (DSWare)

Like SINA and Cougar, DSWare [66] is a data-centric middleware approach to providing information retrieval services within a WSN. DSWare, however, is aimed at handling real-time events and integrates a number of real-time data services while providing a database-like abstraction. In this way it can be considered another database-

like approach adapted to sensor networks. It provides services for reliable data-centric storage and implementing alternative methods for improving real-time execution performance, as well as reliable data aggregating and decreased communication that is aimed at conserving the limited resources available within the network. Its architecture separates the routing layer from the DSWare and network layers as DSWare provides components for improving power-awareness and real-time awareness of routing protocols.

There are six services/components available in this middleware approach (Fig. 10.7):

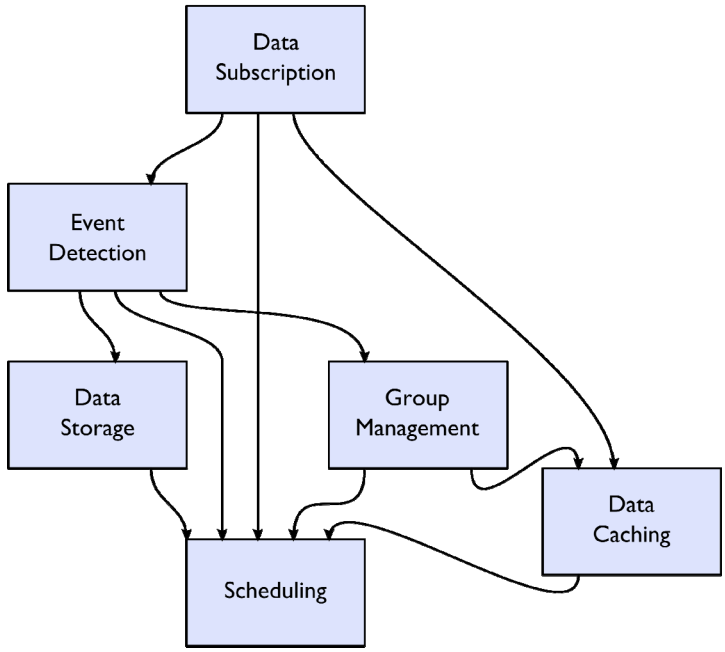

Fig. 10.7. The DSWare Framework [66].

1. Data Storage - this component allows storage of location-linked data so that future queries for similar data would not mean having the query re-issued or flooded through the entire network. In DSWare data lookup is implemented using a hashing function that maps data to physical storage nodes via a unique identifier while robustness is facilitated by using replicating needed data in several physical nodes which can then be mapped to a single logical node. Queries targeting that data can be directed to any one of these nodes in an effort to avoid collision and sustained overload on a single node.

2. Data Caching - this service provides copies of data that is requested often and spreads it through the network using the routing path. This serves to accelerate execution of the query while reducing communication aimed at accessing data for query processing.

3. Group Management - this component facilitates cooperation among nodes to accomplish more complex tasks. In some cases, for example, requested information requires multiple sensors combining their values to calculate a result. The group facility therefore can create a group when a query is issued. Nodes receiving some information on this group can decide whether they match the particular criteria described for that group. The group then manages the values of relevant sensors in accomplishing the given task after which the group is dissolved. In some cases, the query itself may expire before the group can be dissolved.

4. Event Detection - DSWare is based on event detection, an event being an activity that can be monitored or detected in the environment and is of interest to the application. An event is pre-registered depending on the application and can be classified as either atomic events (events that can be determined based on the observation made by sensor) or compound events (events that cannot be determined directly but rather need to be inferred from other atomic "subevents"). Like COUGAR, DSWare uses statements in a SQL-similar language for registering and cancelling events. After parsing the statement defining the event, DSWare generates the query execution plan and calls on the methods required to register, execute and cancel the event.

5. Data Subscription - this service is used to aid in more efficient data dissemination. If multiple nodes request the same data, the subscription service puts copies at intermediate nodes that can then be accessed by the requestor nodes. This helps in reducing communication and helps conserve resources within the network.

6. Scheduling - this component provides a real-time scheduling mechanism as the default method with the option of adding on an energy-aware mechanism if the first has already been successfully implemented. All components of DSWare are scheduled using this component.

DSWare, because of its emphasis on providing event detection services and the mechanisms it provides for data caching, group management and data subscription, is particularly suited to aggregate queries as well as queries for replicated data. These aggregate queries, however, are simple aggregates over an attribute and do not include more complex aggregate queries where data from multiple clusters, for example, could be combined to resolve such queries.

10.3.1.4. Framework in Java for Operators on Remote Data Streams (Fjords)

The Fjords architecture presented in [67] shows how processing multiple queries can be managed over multiple sensors while allowing more efficient resource usage and keeping query throughput high. The focus in the research is on creating the underlying architecture that will support the processing of multiple queries over sensor data streams.

The architecture comprises operators configured for streaming data and Fjord operators called sensor-proxies whose function is to serve as a mediator between the query processing plan and sensors. Data is allowed to flow into the Fjord from the sensor and then pushed into the query operator. Query operators are not active pulling in data but rather wait till data is sent to them from the sensors. The fjord, in addition to combining streaming and fixed data also processes multiple queries and combines them into a single plan.

The Fjords architecture is therefore suited to continuous queries and addresses a number of issues important to query processing. It does this in two ways. First, by using proxies, non-blocking operators and query plans which allow streaming data to be pushed through operators that pull from data sources, it allows merging of the both stream data and local data. Second, by letting proxies serve as mediators between query plans and sensors it allows query processing while at the same time taking into account power, communication and processor restrictions in the network.

The work here is similar to the COUGAR project in that they both focus on processing of streams of sensor data, however, COUGAR although it does incorporate in-network processing in the form of data aggregation, it does not focus specifically on energy efficiency and the resource constraints on sensor devices but rather on modeling the streams of data using abstract types. Fjords, however, look more closely into improving efficiency for processing of data streams. Although a viable approach for continuous queries it does not address complex queries or even aggregate type queries to any great extent and is more concerned with managing simple query processing over multiple sensor data streams.

10.3.1.5. TinyDB

All the approaches to query processing described above have one thing in common: they view query processing in the network as a modified version of that in traditional databases. In essence, each node in the network is seen as a generator of named data against which queries can be issued. [68] describe a distributed database approach that attempts to improve energy consumption by applying varying techniques for aggregation and optimization within the network. This aggregation service is called Tiny Aggregation (TAG) and is designed specifically for TinyOS [69] motes. TinyDB has been one of the more widely used and popular query processing systems and considerable work has gone into extending its capabilities both in terms of the types of queries that can be issued as well as improving its operation by integrating different routing protocols. TinyDiffusion [70] is one such example. It has been widely used in a number of real-life deployments as well as inspired the research into other query processing systems and hence will be discussed in full in this section.

TinyDB uses acquisitional query processing (ACQP) as described in [71] for in-network query processing. In terms of practical applications ACQP focuses on the location and cost of accessing data as a way of reducing power consumption. There is no assumption

made that data exists at a particular location rather queries are only issued where it is known that data exists.

TinyDB can therefore be considered an ACQP engine, a distributed query processor that runs on all nodes in the network. TinyDB was designed to be deployed on the Berkeley Mica mote platform, which runs the TinyOS operating system (Mica motes are arguably one of the most popular of WSN hardware platforms.)

Query Syntax: Queries in TinyDB are quite similar to SQL with the FROM clause being used to either specify sensors or data stored in tables (called materialization points).

Sensor tuples are stored in a table (sensors) with one row per node per instant in time and one column representing each attribute. An attribute could be light, temperature, sound, etc. Records in this table are stored for a short period usually and only "acquired" when needed to resolve a query. An example of a query could be:

```
SELECT nodeid , light

FROM sensors

SAMPLE PERIOD 1s FOR 10s
```

This query states that each node should return its own id and light reading from the sensors table once per second for ten seconds. The results are either returned to the user or logged. The sample period is used to indicate when data collection should begin. The sensors table is an unbounded, continuous data stream of values and can only be sorted or used in joins via a specified window.

Time Synchronization: TinyDB adopts the Timing-Sync protocol presented by [72] which attempts to provide network-wide time synchronization in the sensor network. The algorithm proceeds in two stages: the Level Discovery Phase establishes a hierarchical topology for the network and occurs at the point of network deployment. In the second phase, the Synchronization Phase, a two-way message is used to synchronize two nodes. The time synchronization process begins with the root node first sending a time-sync packet after which receiving nodes initiate the message exchange with the root node as described above in the synchronization phase. Once the nodes receive acknowledgment from the root node they adjust their clocks to that of the root node. This process is carried out until all nodes are synchronized with the root node.

Aggregation Queries: TinyDB supports aggregation of data, which in turn allows reduction of data prior to transmission. The main difference between such queries in TinyDB and SQL is that the output to a query is a stream of values for the former while it is an aggregate value for TinyDB. The aggregate operators allowed are the same as that allowed in a normal relational database system, for example AVG, MAX, MIN which compute the average, maximum value and minimum values respectively for the attribute they are applied over.

Temporal Aggregates: The TinyDB system acknowledges that aggregates over values within a common sample interval are not the only type of aggregate values that a user may want. In some cases a user will need to perform some type of temporal calculation. For example, users of a habitat monitoring system may want to monitor the temperature as a frost moves through a geographical area. This can be done by measuring the maximum temperature over a period of time and reporting that temperature at set intervals. In TinyDB this can be implemented as a sliding window query, for example:

```
SELECT WINAVG(TEMP, 60s, 5s)

FROM SENSORS

SAMPLE PERIOD 1s
```

The above query would report the average temperature over the last 60 seconds every 5 seconds with a sampling rate of once per second.

Event-Based Queries: TinyDB also supports event-based queries which are generated either by another query or by some part of the operating system, the language provides an EVENT clause for that purpose and the code that generates the event is compiled into the node [73].

Continuous Queries: Continuous or lifetime-based queries are also supported using the LIFETIME clause as well as nested queries and offline delivery queries which allow data to be logged for non-real time delivery. Materialization points are used to implement offline logging.

Query Optimization: Queries in TinyDB are first parsed at the base station and then optimized to select the ordering of joins, selections and sampling. The optimizer determines the lowest power consumption that takes into account not only the cost of data acquisition but also processing and radio communication. Data acquisition, however, is the main source as it includes actions like sampling of sensors and transmission of query results. Query optimization, therefore, is focused on reducing the number and cost of data acquisition. Once a query has been optimized it is issued into the network in a binary format, where it is instantiated and executed.

Query Dissemination and Routing: TinyDB uses semantic routing trees (SRT) to help determine whether a query is forwarded or not.

Query Processing: after a query has been optimized and disseminated the query processor executes it. Execution follows a simple sequence of sleep, sampling, processing and delivery. Nodes sleep for as much time in an epoch as possible to conserve power and wake up to sample sensors and deliver results. The nodes are synchronized so parent nodes can ensure that child nodes are awake to process

messages. Results are sampled and filtered based on the plan provided by the optimizer and then routed to the aggregation and join operators further up the query plan.

In summary, the TinyDB approach does allow quite a good deal of flexibility in the types of queries that can be processed and shows increased efficiency in terms of power consumption using a number of query optimization techniques. Some further work needs to be done on optimization of multiple queries, as these are not addressed as well as implementing more sophisticated data delivery prioritization techniques. The authors stress, however, that with TinyDB using the ACQP method it is essential that data delivery, query language and optimization techniques take into account the underlying hardware capability and semantics in order to increase the likelihood of successful deployments.

Add-ons aimed at simplifying the deployment and development of applications for wireless sensor networks, are also being developed. The Tiny Application Sensor Kit (TASK) is built on top of TinyDB for that purpose [74]. This kit contains a relational database used for storage of sensor readings, a server to act as a proxy for the network on the Internet as well as a front-end that facilitates data selection and recording. Other architectures with a similar purpose also exist, for example jWebDust described in [75].

Although useful, like the other query-based approaches in use TinyDB has its drawbacks. The queries that can be issued are limited to those that target “low level” sensed values on nodes in the network or those requiring the computation of simple aggregates like average, maximum, and minimum over some attribute. It does not allow the creation of nested queries or even nested, aggregate queries. In addition, the query language used cannot easily express spatio-temporal characteristics which are an important aspect of the data generated in WSNs. Therefore, there is still room for improvement and expansion in a query processing system that follows the same model but able to issue and process more powerful queries.

10.3.1.6. Active Query Forwarding (ACQUIRE)

Common to all of the query-based approaches described above is the clear distinction between the dissemination phase when the query is sent out and the response stage when results are sent back to the querying node. ACQUIRE described in [76], does not distinguish between these two stages but rather issues what is called an active query.

An active query, in addition to the simple queries for an attribute or attributes, also includes nested queries. Each subquery can be a query of interest in a different variable or value. The active query is propagated through the network node to node. At any point in time, an active node (the node carrying the active query) uses data from a set of neighbouring nodes within a look-ahead of x number of hops and tries to resolve the query or part of the query if it can. The number of hops used can vary, but has been analyzed experimentally to determine what the most effective value is. The experiments in effect measured the average number of nodes from which new information is obtained

during query forwarding based on different values for x and set that as the effective look-ahead. Fig. 10.8 gives an illustration of the ACQUIRE mechanism at work.

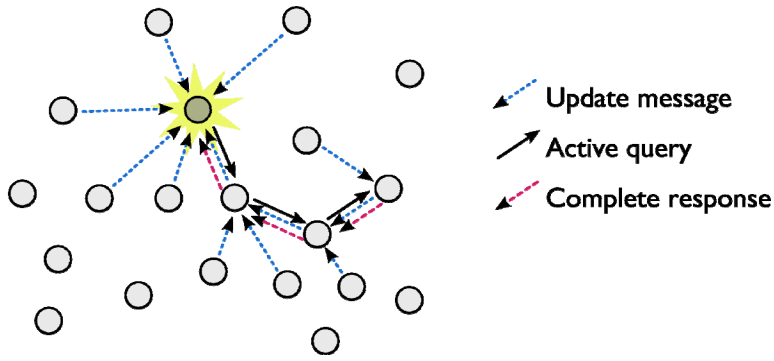

Fig. 10.8. Illustration of the ACQUIRE mechanism given a one hop look ahead. At each stage of the active query propagation the node carrying the query uses information gained to partially resolve the query [76].

Analysis of ACQUIRE has been restricted to networks exhibiting a regular grid topology. Given these restrictions, however, ACQUIRE worked better than flooding-based querying (FBQ) mechanisms like Directed Diffusion as well as had a 60-75 % savings in energy consumption when compared to Expanding Ring Search (ERS) [76]. ACQUIRE is most suited to complex one-shot queries for replicated data and needs to be analyzed on more realistic networks where the topology is irregular or dynamic. The reported evaluation of ACQUIRE is further limited as it uses mathematical modeling to analyze the performance of the mechanism in terms of energy costs. It does not address implementation. Further, the complex queries although including nested queries do not include queries where spatial or temporal characteristics may be of interest.

10.3.2. Macroprogramming Approaches

In the applicative query approaches described above, the user must have some knowledge of what type of queries or operations are to be issued as well as what raw data needs to be targeted in order to retrieve the information. Although information is retrieved, in many cases, the user still has to make sense of what it means and whether it is relevant or not. For example, a user could issue a query in TinyDB requesting the average temperature over the network for a particular period of time. Once the results are received, however, the user would then have to look at this “information” in deciding whether it indicates that a fire has started. If it does, the user then has to decide what to do next, for instance, issue additional queries. The ability to create a “program” where events are anticipated and response (further queries) issued is not allowed in the system.

Macroprogramming approaches to information extraction aim to treat the wireless sensor network as a whole by directly specifying global behaviour instead of programming individual nodes. It attempts to address the issues raised above in the form of a global program that is defined in a high level language that can capture operations going on in the network at a global level. A number of systems have been developed which incorporate global abstractions and in some cases couple them with node-level abstractions in creating useful macroprograms. In this Section some of the more popular macroprogramming systems will be examined.

10.3.2.1. Node-level Abstractions

The underlying principle of macroprogramming is the creation of a global program that can capture network level operations. These macroprograms ultimately have to target node level data in order to function globally. A number of node-level abstractions have been proposed in the literature for modeling node level data and are used by macroprograms to target the data within the network. The data is not accessed on a low level, node-by-node basis but through these abstractions that provide access via logical "collections" of nodes. The logical node-level abstractions proposed are usually based on one or more attributes within the WSN. This can be a dynamic attribute like a sensed value (temperature, for example) or a static attribute like geographic location (in the case of stationary nodes). Following is a description of some of the more popular node-level abstractions put forward in the literature.

Hood: Whitehouse et al. [77] describe a neighbourhood abstraction called a hood (which is defined by a set of criteria for selecting neighbours) and a set of variables to be shared within the hood. Hoods are mainly defined geographically with nodes being included in hoods based on the number of hops from a node. For example, a hood could be a one-hop or two-hop neighbourhood, over which temperature readings are shared or a one-hop neighbourhood over light and temperature readings. A node can therefore define multiple neighbourhoods over different variables.

Whitehouse et al. use a broadcast/filter mechanism to allow data sharing and neighbourhood discovery. Shared attributes are broadcast and receiving nodes cache any attributes of interest from valuable neighbours. A neighbour's value is based on how the neighbourhood has been defined. For example, a node may define a routing neighbourhood that caches location information of valuable routing nodes [77]. Once a neighbourhood has been defined and attributes broadcast, interested observers add the broadcasting node to its neighbour list and cache the attribute or attributes of interest. Fig. 10.9 shows a model for an application with two neighbourhoods and two shared attributes.

Abstract Regions and Region Streams: Welsh and Mainland [78] propose abstract regions, spatial operators used to hold local communications in regions which can be defined in terms of radio connectivity or geographic location. A region in that sense would serve as an identifier for neighbouring nodes who can then share data, identify

each other, and reduce data among them. Nodes could then be manipulated via their common interface. An abstract region could therefore be defined as a relationship between a node and its neighbours and the main aim of this work is to move away from the need to construct low-level mechanisms for routing and data collection and instead focus on a higher level interface that is flexible enough to allow a user to implement queries. Such a system would not, for instance, have to consider routing, data dissemination or state management in order to function effectively, however it would still allow a programmer to create applications that can adjust to changing network conditions as well as adjust energy consumption to improve accuracy and latency.

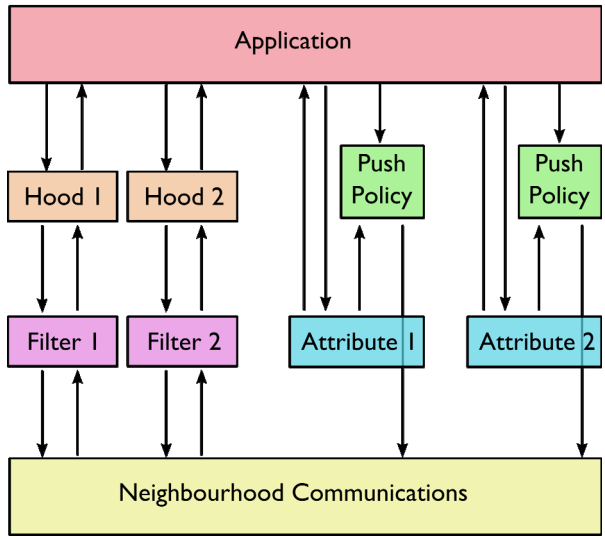

Fig. 10.9. Component model for an application with two neighbourhoods and two shared attributes [77].

10.3.2.2. Semantic Streams

Semantic streams, a system proposed by [79] allows a user to issue queries over semantic values directly without identifying what data should be targeted or what operations should be used on that data. It allows the user to interact with the sensor network using declarative statements, for example, "I want the ratio of cars to trucks in the parking garage". There is no need to construct code to actually check the data for the existence of cars or trucks rather components referred to as inference units are used to facilitate interpretation of sensor data. This is in direct contrast to other approaches that issue queries over raw sensor data like TinyDB and Cougar, and is of interest as it is similar in principle to the work proposed on complex queries in this research.

The Programming Model: The semantic streams framework uses the semantic services programming model, which contains two main elements, inference units and event streams. An event stream is simply a flow of asynchronous events, each of which

represents some real-world entity (for example, in the car park query given earlier, a car detection has properties such as the location at which it was detected, its speed and direction). An inference unit is a process that operates on an event stream. It infers semantic information about its environment and either adds it as a property to an existing event stream or generates a new one. Fig. 10.10 shows the semantic streams model within its service-based architecture.

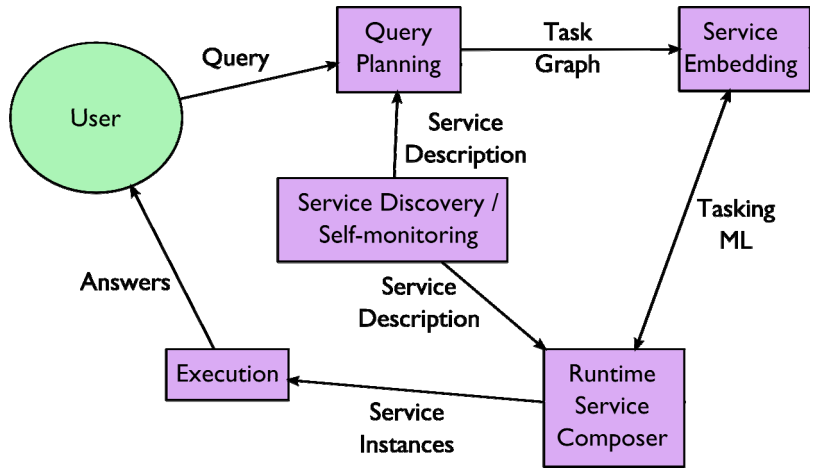

Fig. 10.10. The semantic streams query processing model [79].

As a stream moves from a sensor through inference units its events acquire new semantic properties. A more detailed example, a vehicle detector service, follows.

The Query Language: Each inference unit is specified using a first-order logic description of the semantic information it needs in its input stream and that it adds to its output streams as well as any relationships between the two. A mark-up language (the Semantic Streams mark-up and query language) is used to construct a logical description of that semantic information. A number of predicates are used to describe sensors and services: the sensor predicate defines the type and location of each sensor; the service, needs and creates predicates describe a service, the semantic information that it needs and creates. In query processing these are treated as rules and their pre- and post-conditions; the stream, isa and property predicates describe an event stream and the type and property of its events. An example of a sensor declaration is as follows:

sensor(magnetometer, [[60,0,0], [70,10,10]])

This defines a magnetometer that covers a three-dimensional cube defined by the pair of 3-D coordinates.

An example of a service would be:

service(magVehicleDetectionService,

needs(sensor(magnetometer, R)),

creates(stream(X), isa(X, vehicle),

property(X, T, time), property(X, R, region)))

This vehicle detector service uses a magnetometer to detect vehicles and generates an event stream with the time and location in which the vehicles were detected.

Once a set of sensors and services has been declared the user can begin to issue queries. An example of a simple query would be:

stream(X), isa(X, vehicle)

This query would result in true if a set of services could be composed to generate events X that are known to be vehicles. All known possible service compositions would be generated. To constrain the result more predicates could be added. For instance,

stream(X), isa(X, car)

property(X, [[10,0,0], [30,20,20]], region)

This would restrict the result to cars found in the region described by the pair of 3-D coordinates given.

Query Processing: An inference engine is used to determine which sensors and services will provide the required semantic information. The inference engine employs the backward-chaining algorithm [80] using the pre-conditions and post-conditions of the services. An attempt is made to match each element of the query to the post-condition of the service; if this is successful the pre-conditions are added to the query. Once all pre-conditions have been matched to actual sensor declarations (without pre-conditions) the process terminates. In other words, the process terminates once physical sensors have met all inference units' requirements. A description of a real-world application of the semantic streams framework is described fully in [79].

In the semantic model described, all services and their descriptions are maintained in a central repository or query server. Resource usage is optimized because the inference engine reuses services when possible and the query language used allows great flexibility in how the query processor executes queries. The language also allows the user to specify constraints. When a user issues a query as an event stream the query-planning engine generates a task graph, which is then assigned to a set of nodes in the network. The service runtime on each node then accepts the graphs and instantiates services as required, resolves any conflicts between tasks and available resources and then executes the query.

One major limitation of the system is the semantic mark-up language used. Although suitable for simple examples it cannot capture more complex applications. Another drawback of the system is that the query processor cannot reason at runtime and so two applications that have to access the same device, for instance, would not be able to run simultaneously. It does not address sensor network issues that are not semantic transformations. For example, routing data between nodes in a sensor network would not change the semantics of the data being routed. Semantic Streams cannot differentiate between different routing algorithms and so cannot take advantage of their features if needed.

10.3.2.3. The Regiment Macroprogramming System

Regiment, proposed by [81] is a functional programming language for sensor networks. Its data model is based on the concept of region streams, where sensor nodes are represented as streams of data that can be grouped into regions for the purpose of in-network aggregation or event detection. Collections of nodes, therefore, can be represented both spatially and temporally. A region stream is used to describe a collection of nodes with a logical, topological or geographical relationship. For example, a programmer may be interested in nodes that fall within certain 3D coordinates. The corresponding region stream would represent all sensor values for those nodes.

Operations allowed on region streams include fold, which aggregates values across nodes in the region to an anchor node, and map which applies a function over all values in a region stream. A fold requires inter-node communication while a map does not. Regiment, as is the case with other functional programming languages, allows functions to take functions as arguments.

The Regiment compiler converts the high level program into node-level code that targets an abstract machine model called the token machine. This is an intermediate language that links token handlers to a token name which represents a task to be executed by a sensor node once it receives a token of that name. A token may be generated locally or through a radio message. The token machine language described by [82] elaborates on these concepts and defines the Token Machine Language, which is based on an abstract machine model called Distributed Token Machines (DTMs). Typed messages with a small payload (a fixed size buffer) are referred to as token messages and used by DTMs for communication. Each token has an associated token handler, which is executed when a token message is received by a node. DTMs, in addition, allow for concurrency, state management and communication. The aims of TML are to provide abstractions for key requirements in the sensor network including data dissemination and communication and provide a framework within which complex algorithms, such as those that allow in-network aggregation, can be implemented.

[16] describes the Regiment Macroprogramming System, which extends the work done on the Regiment language and token machine interface. It uses “signals” which represent attribute values for nodes, for example, the temperature read by a sensor, and

groups signals into regions that are then used as a programming abstraction upon which different operations can be performed (aggregation for instance). Region membership is not static and may vary over time, due to changes in sensed values, node or communication failure or new nodes being added to the network. Key here is that the system abstracts away details of storage, communication and data acquisition from the user and allows the compiler to map global operations on regions and signals to local network structures [16].

The Regiment system has been evaluated in a number of simulated chemical plume detection macroprograms with a network size of 250 nodes over an area of 5000 x 5000 m. The results demonstrated the flexibility of the system in maintaining good communication performance [16].

10.3.2.4. Kairos

In contrast to some programming approaches that focus on providing high-level abstraction for local node behaviour in a distributed computation, the Kairos macroprogramming system [83] focuses on abstractions for specifying global behaviour. This distributed computation encompasses the entire network but uses a centralized approach. The abstraction considers the sensor network as a collection of nodes that can be tasked at the same time from one program. The programming model is based on shared-memory-based parallel programming models over message passing architectures and three programming abstractions are made available in the system. Fig. 10.11 shows the architecture of the Kairos macroprogramming system.

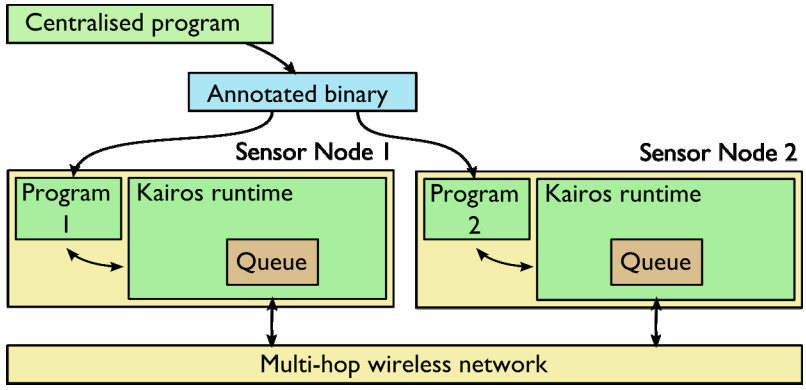

Fig. 10.11. The Kairos programming architecture [83].

Programming Abstractions: Kairos provides three abstractions for manipulation by the programmer. The first is the node abstraction where programmers are allowed to manipulate individual nodes or lists of nodes. Nodes have integer-based identifiers and are represented by a node data type which makes available operators like equality,

ordering and type testing. There are also functions made available for manipulating lists of nodes.

A second abstraction is the set or list of all one-hop neighbours of a specified node, which is made available using a get_neighbours() function. This abstraction is quite similar to the region and hood abstractions and it reflects the natural construct of radio neighbourhood of the nodes in the network.

The third abstraction is remote data access, which allows a programmer to read from variables at a named node using a variable node construct. In Kairos only a node may write to its variable although multiple nodes may have read access.

Programming Mechanism: A distinguishing characteristic of the Kairos system is that a programmer writes a single centralized program representing the distributed computation. First, the program is pre-processed to produce annotated source code that is then compiled into a binary. During this stage, the Kairos pre-processor identifies and translates remote data references to calls to the Kairos runtime. The binary can then be distributed to all nodes in the network using a code distribution facility. Although the initial program is a global level one, after compilation a node-level version is created that details what an individual node's functions are, when and what data remote and local it manipulates.

When a copy of the node-level program is instantiated and executed on a node, the runtime exports and manages variables that are owned by a particular node but are referenced by remote nodes.

Kairos has been used to implement programs for a variety of problems from routing tree construction to vehicle tracking [83].

10.3.2.5. Knowledge-Representation for Sentient Computing

An interesting approach is proposed by [84] called a scalable knowledge representation and abstract reasoning system for Sentient Computing (See Fig. 10.12). Sentient Computing promotes the view that an application can be made more aware by perceiving the environment and reacting to changes in it. A gap exists, however, between the level of abstraction in the knowledge of the sentient world such a system requires in order to function and the low-level data produced by sensors in the environment. [84] propose the creation of a deductive component that would interact with the low level data, perform some type of reasoning function in order to deduce a higher-level abstract knowledge. The application layer can then use this knowledge. The deductive knowledge base layer therefore is really a domain specific language component positioned as a type of middleware layer between the application and hardware layers.

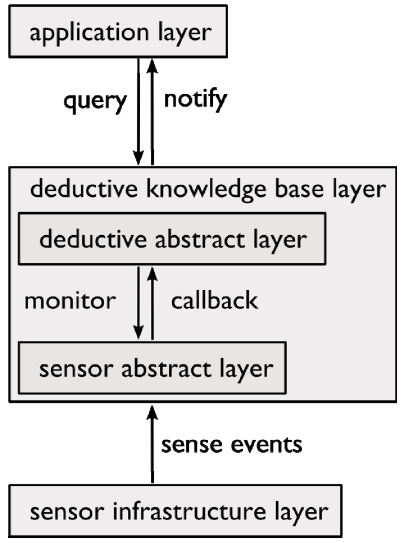

Fig. 10.12. System architecture for Sentient Computing [84].

Knowledge Representation: An event in this context is defined as an occurrence of interest taking place instantaneously at a specific time. The Sentient environment refers to the physical environment. The current logical state of the Sentient environment includes all known facts about the sentient environment from an initial event to a terminal event. These events can include any events of interest to the Sentient Application layer. The Sensor Abstract Layer (SAL) keeps a low-level view on the current logical state as sensors in the environment update it through sensed events. The Deductive Abstract Layer (DAL) maintains a high-level or abstract knowledge of the current logical state through its interaction with the SAL.

The two layers use a monitor-callback communication scheme to interact. The application layer issues a monitor call that causes the SAL to filter through to the DAL those low-level changes that affect the abstract knowledge in the DAL. The DAL, therefore, does not have to monitor all the data and is updated at a much lower rate than the SAL. This is depicted in Fig. 10.13.

Both DAL and SAL are described using first order logic and the application layer retrieves the stored knowledge about the Sentient environment using queries, which are quite similar to SQL SELECT statements. Experiments conducted with a prototype implementation showed that the two-layered architecture where the low-level and high-level interactions are separated (SAL and DAL) was more efficient than a single-layered one for a number of reasons [84]. First, queries executed in the DAL were of a simpler form with fewer conditions so used less computational resources. Second, the knowledge update rate triggered by assert or retract commands in the DAL is lower than that of the SAL making DAL more computationally efficient. Experiments with more complex queries have not been carried out.

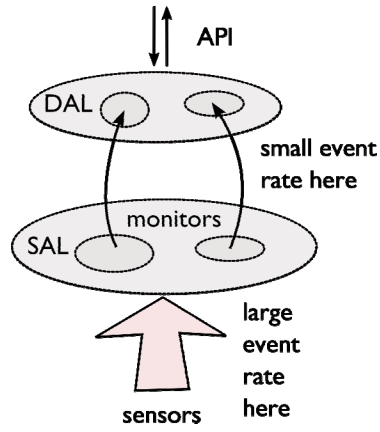

Fig. 10.13. Interaction between DAL and SAL, derived from [84].

10.3.3. Remarks

The three approaches to information extraction surveyed over Sections 10.2 and 10.3 show systems and techniques with varying degrees of ease of use, flexibility, informational complexity and expressiveness. While the querying approaches are easier for a user to manipulate they are restricted by the lack of expressiveness of the query languages they provide for requesting information. This in turn restricts the type and level of information that can be retrieved. Although it is clear that simple queries for attributes and simple aggregates are possible and extensively demonstrated in the deployments to date, informational query-based systems that go beyond that (for example spatially and temporally aware queries) are practically non-existent. Query-based systems are also limited as they do not allow users the flexibility to target smaller areas of interest within the network but in effect force the user to task the entire network in order to get a response, an issue in terms of energy efficiency. An advantage however is that these systems have been extensively used, the languages used familiar to users of sensor network systems and are relatively easier to design and implement when compared to other information extraction mechanisms.

In contrast, the agent-based approaches are more expressive and flexible in terms of what functions can be performed and the ability they incorporate to make decisions while in the network. Agents can act autonomously, multiple agents can run on a node at the same time, and multiple applications can co-exist in the network (this can be a problem with many query-based and macroprogramming systems). Mobile agents can move, clone themselves and can act to deal with unexpected changes in the environment. Agent-based systems, however, are hindered by the difficulty they present for non-expert users to design and program. It is also apparent that despite the many agent-based techniques and models proposed in the literature, actual deployments have been minimal and restricted to very small networks (less than 10 nodes) with the agents being, at most, just pieces of mobile code with minimal autonomy or decision-making capability.

The Macroprogramming approach appears to be one solution to the problem of the limitations of query-based systems while at the same time promoting the expressiveness inherent in agent-based systems. Some researchers consider macroprogramming an extension to query-based applicative approaches and attempt to adhere to the ease of usability mantra of query-based systems, while at the same time providing users the ability to access higher level information. The reality, however, is that the systems are not nearly as intuitive as many of the SQL-based querying systems and require a learning curve for the programmer. In addition, the approach seeks to eliminate the need for low-level programming on the node but in reality has to provide node-specific abstractions, which undermine rapid program development. Finally, the high abstraction level although advantageous on the one hand in terms of shielding the user from the underlying workings of the network (radio communication and network topology, for example) presents a challenge when constructing compilers that need to cater not just for computation but node level communication as well. Macroprogramming therefore presents similar challenges to the agent-based approaches.

Overall, the approaches to information extraction examined highlight that trade-offs must be made between ease of use and expressiveness coupled with flexibility (for instance in terms of decision-making capabilities within the network) in selecting a suitable information extraction system. Clearly, the query-based and macroprogramming approaches are useful in a number of applications but raise a number of challenges as outlined above. There is a definite gap between the ease of use provided by the more popular query-based mechanisms and the expressiveness of the global and node level abstractions provided by macroprograms as well as the flexibility of agent-based systems. For purposes of any applied research in WSNs, the idea of usability and utility are paramount. Future research directions and open issues are presented in Section 10.4. To that end, the authors here put the development of a declarative query-based system forward as being a viable option. Given the limitations already described above, a number of features have to be incorporated into such a query system. In order to make the querying approach more attractive, the development of a query language that allows a user to more richly express high level information requests which can target the full breadth of collectable information presented by sensors in the network is promoted. Complex queries could be suitable constructs that can allow the expression of spatio-temporal characteristics and requests requiring more involved in-network interactions to generate information. Incorporating within this query processing system useful elements that will aid the processing of these complex queries, such as abstractions already employed in some macroprogramming approaches to facilitate resolution of this higher-level information requests would be necessary. Such a hybrid approach, it is anticipated, will retain the simplicity and ease of use of the traditional query-based approaches while simultaneously allowing the inclusion of useful logical abstractions provided by macroprogramming approaches to facilitate dissemination, processing and resolution of more powerful queries than those available in query processing systems today.

10.4. Towards a Hybrid Approach

10.4.1. Introduction

In this Section, the case is made for a hybrid approach that retains the simplicity and ease of use of the more traditional query-based approaches while allowing the inclusion of useful logical abstractions provided by macroprogramming approaches to facilitate construction and resolution of more powerful queries. Such an approach would need to incorporate some of the principles of agent-based systems, such as collaboration and decision making in the network, in assisting in query resolution. In this context it can be hypothesized that end-users of WSN applications could be provided with the ability to construct and pose higher level information requests instead of simple queries requiring the collection of raw sensed values or the calculation of simple aggregates over the entire network [85].

Complex query type constructs that allow the expression of phenomenological spatio-temporal characteristics would provide the means for exploiting the networked sensing concept at large scale. Moreover, providing the end-user with a system that produces responses to these higher level requests for information within the network instead of as a result of post-collection analysis of all data would most certainly demolish one of the largest road-blocks of the WSN technology in its route to adoption: ensuring user acceptability. In the quest for testing the above hypothesis, this Section looks at some of the open research issues the community faces with respect to information extraction, raises research question relating to the usefulness of in-network processing from an informational viewpoint and concludes the chapter.

Section 10.4 is structured as follows: the first subsection identifies a family of applications which would benefit from a WSN-integrated higher level information extraction and delivery mechanism and surveys the state of the art of deployments in the identified application area. Next, the requirements for an integrated high-level information extraction mechanism are described and reviews of prior art in complex querying and in-network processing are provided. The following subsection identifies WSN topologies, architectures and protocols suitable for advanced informational systems; this is followed by the description of a new, hybrid approach, termed a Distributed Complex Query Processor.

10.4.2. Information Extraction in Monitoring Applications

Military applications were the motivation for much of the initial research into WSNs with the Defense Advanced Research Projects Agency (DARPA) providing funding for a variety of projects from early 1990s to the present day. Today, however, the use of WSNs has expanded to encompass a large number of application domains. A particular group of WSN applications, denoted as monitoring applications, make up a large proportion of the WSN systems being deployed today. Though most monitoring systems are generally characterized by the need for attributes of interest to be observed, sensed

and then relayed to a user, most WSN applications of this type show very little genericity in design. Rather, the WSN systems deployed are application-specific, with designs intimately linked to the particular application requirements. This section will identify some of the common types of monitoring application and describe examples of each, highlighting the information gathering model used in each example. An application acting as a motivator for higher-level information extraction procedures is also described.

10.4.2.1. Habitat and Environmental Monitoring

Habitat and environmental monitoring form a particular set of monitoring applications that have grown over recent years. This is in part because of the great benefit they claim to provide to science and education as well as the fact that funding for improving science education is a priority. There are a number of WSN deployments in this category.

On the environmental side, the ARGO project [86] uses a sensor network to monitor the salinity and temperature of the upper ocean. The aim of the project is to generate a quantitative model of the changing state of the upper ocean while monitoring ocean climate patterns over the short to longer term. The data gathered is expected to act as input to ocean and ocean-atmosphere forecast models for data assimilation and model testing. Nodes are dropped from aircraft and cycle between the surface and a depth of about 2000m every ten days collecting data. When nodes rise to the surface, data collected by nodes are individually transmitted to a satellite.

The Great Duck Island project [85] is an example of a habitat monitoring application where a WSN was deployed to monitor the breeding behaviour of small birds called petrels. Scientists are interested in the usage patterns of nests as well as the environmental changes in and outside of nests during the breeding season. Measurements are taken of humidity, temperature, pressure and light level. Nodes are clustered with each cluster connected to a base station via a long-range antenna. Sampling is done every minute and readings are sent directly to the base station, which connects to a database via a satellite link.

Another environmental monitoring application is the GLACSWEB project [87], which uses a sensor network to monitor sub-glacier environments in Briksdalsbreen, Norway. Holes are drilled at different depths in the ice and sensor nodes carrying pressure, temperature and tilt sensors are deployed in them. The nodes communicate with a base station on top of the glacier that measures supra-glacial displacements. The collected data is transmitted via GSM with no information processing occurring within the network itself.

For all of the applications above, raw sensed values are sent via base stations or servers to a user and processing occurs at that point.

10.4.2.2. Agricultural Monitoring

Increasingly, WSNs are being used to monitor conditions that affect plant growth. These networks are used in precision agriculture, which aims to improve crop yields, reduce pollution and monitor the general health of crops. Variables monitored include temperature, soil moisture, humidity and light and are usually referred to as microclimate variables. The Climate Genie system [88] is one such commercial example and is used to monitor vineyards. Nodes are spread over the vineyard in a wireless mesh configuration and are equipped with sensors for measuring temperature, moisture and light. The data gathered is summarized (aggregates that reflect grape quality and vitality) within the network and sent via Wi-Fi, cellular or satellite to servers for viewing anywhere using a web browser. This system therefore incorporates some level of in-network processing although limited to the calculation of aggregates that are used in decision making by the observer.

Another system was developed to monitor in real-time field conditions including leaf moisture, soil temperature, soil moisture and CO_2 [89]. Field monitoring servers (FMSs) similar to web servers were deployed in rice paddy fields, collected data automatically and transmitted the data for permanent storage in publicly accessible databases. Real-time data was made accessible via a web browser. Like the habitat and environmental monitoring applications described in Section 10.4.2.1, raw sensed values are sent to the databases for storage and analysis occurs from that point on.

10.4.2.3. Structural Health Monitoring

Structural health monitoring (SHM) refers not only to the state of health of a given structure whether a building or bridge, for example, but also the detection of changes that may affect the structure's health in the future. There are two types of SHM systems: systems that monitor disaster response after some catastrophe has occurred, for instance, an earthquake and dedicated to continuous health monitoring which may check for signs of stress, monitor vibrations, wind, etc.

Disaster response systems are still a young area of WSN research. [90] describe a centralized WSN for structural-response data acquisition called Wisden. Wisden collects structural response data from a multi-hop network and relays and stores it in a base station. [91] also proposed a wireless monitoring system aimed at detecting damage to civil structures after a disaster (such as an earthquake) has occurred. In other research, technology developed by the CodeBlue project [92] is being used in the AID-N project [93] at Johns Hopkins Applied Physics Laboratory in developing systems aimed at disaster response.

For continuous, structural health monitoring applications the focus is on high sample accuracy with minimal distortion, high frequency sampling, time synchronization of readings and efficient data collection as opposed to energy efficiency through reduced power consumption. [94] describes a SHM system deployed on the Golden Gate Bridge.

64 nodes were deployed over a 4200 ft long length and measurements taken of ambient structural vibrations. All collected data were relayed to a base station for analysis.

10.4.2.4. A Motivating Scenario

A WSN application frequently put forward is that of forest fire detection and monitoring [95]. This application is attractive because forestry is a major industry in many parts of the world, and forest fires are a major cause of loss of wood (in the USA the annual average loss to fire is 17,000 km²). Early warning of a fire event and the manner in which it spreads are hence necessary.

Theoretically, any practical forest fire detection system is likely to exceed the scale of present WSNs by a considerable margin, with an expectation that hundreds of thousands of nodes would be needed for detailed monitoring and precise fire detection and localization. Considering a network of this scale, it is clear that real-time data searches using conventional centralized query mechanisms are not an option. For each fire detection cycle, hundreds of thousands of readings must be returned to the sink and processed. Following detection of the fire, new queries must be generated and directed to the nodes in the area, which would be, ideally, geographically mapped. Finally, those nodes need to return the infrared data required for the map. A more efficient proposition would be for the initial event to be detected within the network, with the subsequent queries for the map data being generated locally, without returning data to the host.

A number of deployments have tackled some of the very issues described above but for practical reasons, the deployments have taken very different approaches to those proposed in the scenario described above. The FireWxNet system [96] a wireless sensor system aimed at monitoring weather conditions in woodland fire environments is one such example. The system monitored a variety of weather conditions that influence fire behaviour with an aim to using them to predict fire behaviour. The application, as is the case with most monitoring applications, was driven by a list of requirements acquired through consultation with both fire fighters and fire researchers. The implemented WSN system was deployed in the Bitterroot National Forest in Idaho (USA) and consisted of 3 sensor networks totalling 13 nodes, 5 wireless access points, 2 web cameras and 5 long-range links.

The deployment was distinguished by the rugged environment within which the WSN had to function as well as the sparseness of the deployment itself. The authors note that sparse coverage was a deliberate choice aimed at strategically placing nodes to cover as much meaningful terrain with as few nodes as possible. Rather than scale, the focus was on creating an extremely robust design, which included not just robust equipment but robust routing protocols as well. Once the system was launched, a number of challenges were encountered particularly given the harsh deployment environment but the system overall was a success. During its operation, over 80,000 measurements were streamed in real-time, with operations applied post-collection to transform that data into usable information.

This real-life deployment presents a marked contrast to the motivating scenario and approach described above: the problems are well characterized given the input of domain experts; there is quite detailed knowledge of the application domain and what the corresponding WSN application requirements are. However, on-the-ground challenges usually mean compromises have to be made and as a consequence real-life monitoring systems so far, rarely aim to deploy such large quantities of nodes as put forward in the motivating scenario here.

The FireWxNet example highlights the limitation of the vast majority of monitoring applications today. They are limited in that they are conceptualized, designed and implemented as primarily data collection systems. In-network processing is absent and the systems can be considered automated to the extent that there is little user interaction. In essence, events are defined, queries may be deployed for sensed readings and all data is simply relayed to a centralized location for further analysis. Very few systems look beyond this simple sense-and-send model to incorporate some sort of analysis,

in-network, prior to communicating results. Granted, with some systems it is difficult to define beforehand what events or processes may be of interest and these applications by their nature have to be more exploratory at least at the pilot deployment stage, with feedback influencing subsequent iterations. However, with some applications, as in the FireWxNet example, the problems are better characterized and therefore more amenable to some more sophisticated analysis or processing within the network. This could make the application not only more useful but more efficient as well.

Hence, the view put forward here is that in many of monitoring applications (the fire monitoring scenario being only one example) it is possible to define high-level information requirements that could incorporate in-network processing techniques in resolving the requests. Besides the efficiency benefits in terms of reduced transmissions, this approach could transform a data collection system to an information generating system, extending the application scope while still fulfilling the basic application requirements.

As an example, an informational fire monitoring system would allow the processing of high-level queries aimed at tracking the fire. The queries would need to allow the expression of spatial and temporal characteristics and the querying system would need to support a level of autonomy in terms of some in-network decision-making by the nodes given the impracticality of streaming all data back to the sink. Such a system would perhaps allow a user to zoom in on particular problem spots and query for even more detailed information on-the-fly. Complex queries are proposed as extremely suitable for use in the scenario above as well as other monitoring application scenarios and a degree of autonomy is introduced through the implementation of in-network logical abstractions for query processing and resolution.

10.4.3. Requirements for a Higher Level Information Extraction System

With a view of the above, a higher-level information extraction system should to be able to:

- Enable the user to construct and disseminate complex queries;
- Allow a user to program the sensor network in a similar way to current applicative approaches;
- Enable a means for in-network distributed query processing that allows information to be generated within the network;
- Be implementable on constrained resource nodes.

10.4.3.1. Catering for Complex Queries

Towards the requirements above, some research has addressed specifically the need for complex queries and for the ability to process these queries within the network. [113] for example, define a complex query as a query consisting of one or more subqueries that are combined by conjunctions or disjunctions in an arbitrary manner. In their work on the Active Query Forwarding mechanism (ACQUIRE), they promote the use of these "nested queries" and describe a mechanism that seeks to resolve the query in-network, generating information as a response. The work, however, does not address the implementation of the mechanism and instead presents a mathematical model that is used to analyze the performance of the approach in terms of energy cost.

[98] also identify the need for nested queries (which they too call complex queries) and highlight the problems with evaluating such queries especially in cases where aggregation dependencies exist between the nested queries. They put forward the idea of the query itself supporting abstractions that can then be used in query resolution. In their example the abstractions are geographical regions. Their research resulted in a qualitative study of the requirements of such a system and did not extend to implementation.

Beyond the work reported above, a complex query, in the authors' view, is a query which:

- Consists of one or more subqueries (nested queries) and/or
- Contains multiple operations such as aggregates and/or
- Contains spatial and/or temporal elements.

The query-based systems currently in use neither provide the facility to construct these complex queries nor give users the ability to process them. Such queries, for instance, queries with dependencies (nested queries) would require more complex in-network interactions than those supported by current query-based systems. One example of a complex query, which forms the object of the research here, would be:

"What is the average temperature in those areas in the network where the humidity is greater than 95 and the air pressure is between 900 and 1000 mbar".

This query exhibits a number of complex elements. First, the query language would have to be able to accommodate the expression of the spatial elements described as "areas" in the example above. Second, the query is a dependent or nested query and can only be answered after some reasoning within and between the defined spatial entities. In effect, parts of the query depend on a previous question being answered and only become relevant if a particular answer is obtained.

With the primary goal of creating a system that exhibits simplicity and usability, using a declarative language already familiar to users of existing applicative query mechanisms as well as traditional database systems is considered a worthwhile approach. Identifying and investigating existing SQL-based query languages towards creating a language capable of expressing the complex query requirements described above is essential.

10.4.3.2. Catering for In-Network Complex Query Processing

As will be shown in Section 10.4.4.5, a number of in-network processing techniques have already been proposed in the literature and used in existing information extraction systems. These include techniques like aggregation, fusion and filtering which have been shown to improve energy efficiency in WSN systems and have been incorporated extensively in both query-based and agent-based systems. In addition, a powerful feature of many of the macroprogramming systems has been the creation and use of node or network level logical abstractions to facilitate in-network information processing. The literature has shown that so far logical abstractions have not been considered for application in query processing systems. The authors believe that these logical constructs can provide a more powerful means for processing the complex queries identified as being of interest to monitoring applications. The usefulness of abstractions that can be constructed logically within the network and used in conjunction with the in-network processing techniques mentioned above is examined in Sections 10.4.5.5 and 10.4.5.6. The abstraction proposed in Section 10.4.5 can be based on two types of attributes: Static attributes that do not vary over time (such as, the type of reading a node provides) and dynamic attributes which do vary over time (such as, the current sensor reading). The key idea here is that the abstractions would be a component of the query itself and constructed prior to the dependent query being posed. These abstractions will drive the manner in which the query is both disseminated and processed within the network.

Consider an example where a user is interested in monitoring the soil acidity and relative humidity in "hot patches" of the vineyard. The query posed would first need to define what the "regions of interest" are. In this case, regions would be logically constructed over areas where the temperature level registers above a given threshold. Once these regions have been defined, the body of the query, aimed at retrieving soil acidity and humidity readings, will be disseminated to the relevant nodes via the region construct and not to any node within broadcast range, for instance. The core concept is that the regions are queried rather than individual nodes. Another key feature is that query-dependent logical abstractions will be used to facilitate query dissemination and

processing in the network. The following sections review and identify suitable WSN topologies and adequate architectures and protocols that might enable the implementation of such an information extraction system.

10.4.4. WSN Topologies, Routing Protocols and Architectures

10.4.4.1. WSN Topologies

The most basic WSN topology is the centralized, sink-based topology, sometimes referred to as a flat or single-tier architecture where nodes in the network are homogeneous. That is, nodes are identical in terms of hardware complexity, battery power [99, 100], and bandwidth management.

Data collected by the nodes are directed toward the sink or base station (usually the only "node" more powerful both computationally and in terms of energy capabilities) using single or multi-hop communication [102]. Fig. 10.14 shows a diagram of a typical flat WSN topology. This configuration brings a number of challenges particularly if there are a large number of nodes. These include management of energy consumption, energy optimization, routing, information gathering and general management of the sensor nodes themselves [99], [100]. The University of California at Berkeley's Redwood forest deployment [103] is one example of a WSN system exhibiting a flat architecture. The 33-node deployment used TASK [111], a self-contained sensor network system based on the TinyDB query processor [104] to monitor the microclimate (temperature, relative humidity and solar radiation) of a redwood tree over a 44-day period.

While centralized, sink-based topologies are relatively simple to support and implement, they do not fulfill a topical requirement of WSN designs: scalability. Scalability has been described as one of the key design requirements for both conventional communication networks and wireless sensor networks. As the number of nodes increases with a flat topology, however, the sink node may become overloaded leading to increased latency as well as severe energy usage [104]. As a result of the apparent problems posed by flat networks, hierarchical heterogeneous architectures were proposed. The simplest example of this type of network consists of two layers: the first contains groups of homogeneous nodes, called clusters, connected to a dedicated micro-server or cluster head [102]. Cluster heads are sparsely distributed and serve as aggregators of data and managers of nodes within their individual clusters. They also serve as communicators both to a gateway or sink node and with other cluster heads in accomplishing application goals. Fig. 10.15 shows an example of a hierarchical architecture.

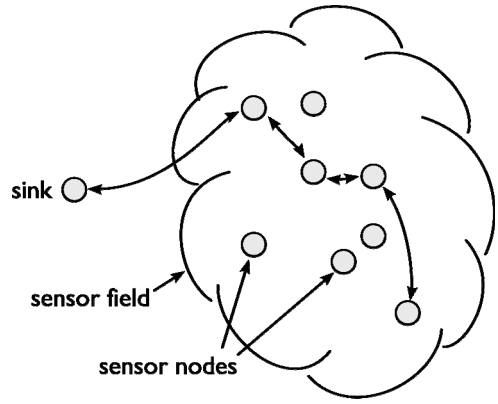

Fig. 10.14. A typical, flat WSN architecture.

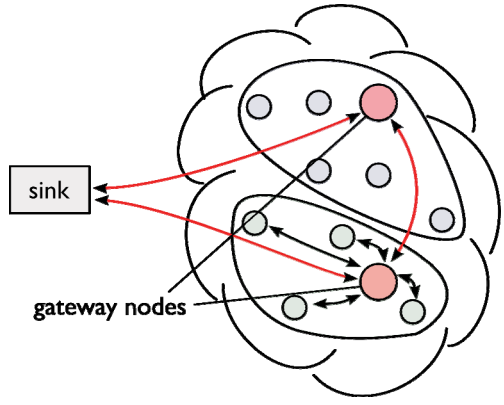

Fig. 10.15. A hierarchical, heterogeneous WSN architecture where sensor nodes are arranged in clusters with a cluster head or gateway node managing the cluster and performing all communication external to the cluster.

The introduction of cluster heads with special functionality (usually supported by enhanced hardware) gives the hierarchical structure some advantages over the flat topology particularly with respect to energy consumption.

Note: In this respect, multihop communication has been identified as a more favourable strategy over single hop, given that energy consumption is directly proportional to the square of the distance [106]. Hence, there is potentially more energy conserved with multiple short hops from node to sink, for example, as opposed to a direct long hop from a node to the sink [107]. A problem occurs, however, with the nodes closest to the sink since they are under heavier traffic and more likely to have their energy drained more quickly. High-energy consumption can also be a problem if clusters where multihop communication is used are large.

Cluster heads provide additional functions like caching and forwarding of data to the required destination as well as performing data aggregation and fusion in order to decrease the number of transmissions to the sink or gateway. This is of particular importance in data gathering networks. It has been shown that for some applications, using a hierarchical structure can bring about significant improvement in performance in terms of reliability, longevity and flexibility of the network [102]. The Great Duck Island habitat monitoring application [85] is one example of a deployment that used a hierarchical architecture.

Building on the basic two-tier hierarchical model, varying multi-tier architectures have been put forward to capitalize on the apparent advantages. Examples are those proposed in [108] and [109]. Finally, the SENMA architecture [110] proposes a novel two-tier architecture where the upper layer consists of mobile access points that are used for data acquisition from homogeneous sensors within the sensor layer. Sensors communicate with mobile agents who periodically move within radio communication range eliminating the need for multihop communication. This architecture, because of the reduction in multihop communication, showed a significant improvement in energy efficiency.

10.4.4.2. Routing Protocols for WSNs

Given the unique characteristics of WSNs the underlying routing protocol used is an important consideration when looking at information extraction. The efficiency of the information extraction mechanism mainly depends on the routing method used. In some cases the routing mechanism itself includes techniques for query dissemination and data aggregation as in Cougar [97] and ACQUIRE [113]. Routing is therefore an important aspect of information extraction in WSNs.

Many routing protocols are currently being used in a variety of wired and wireless networks [112]. Although protocols can be classified in different ways, for example, based on the network structure or perhaps on the protocol operation [113] many follow the address-centric (AC) model where routes are found and followed between pairs of addressable nodes. In mobile ad-hoc networks, AC protocols are used widely (MANETS) [114]. Here each source independently sends data along the shortest path to a sink. This path is usually based on the initial route a query took to get to that source node. Although MANETs are similar to WSNs in that they both involve multi-hop communication, their routing requirements are quite different for a number of reasons.

First, communication in a WSN comes from multiple data sources to the sink instead of just between a pair of nodes. Second, redundancy in sensed data is common in WSN since multiple sensors may be sensing the same phenomena; a fact that data-centric protocols take into account in devising optimal routes. This is not a major consideration in MANETs. Finally, in WSN the major constraint is energy making it essential that data communication rates be made as efficient as possible, much more so than in MANETs.

Given these reasons among many others, the traditional end-to-end protocols used for MANETS are not appropriate for WSNs. Some alternative protocols propose a different model, the data-centric model, where sources send data to a sink, but the content of the data is examined and some processing (whether aggregation or fusion) is executed on that data en-route to the sink. The result in using such protocols is that a better transmission/information ratio is achieved. In the next section some data-centric routing protocols will be examined in more detail.

Although protocol development is usually seen as being beyond the scope of information extraction research, it is important to have an awareness of what protocols are used within the query processing systems in use and their impact on the information extraction mechanism. Data routing, for example, is an important consideration when looking at information extraction as it affects the overall efficiency of the mechanism. For purposes of building advanced informational systems fitting the requirements in Section 10.4.3, identifying a suitable cluster-based routing protocol that facilitates the implementation and testing of the logical abstractions is key.

The choice of architecture for a WSN system appears to depend strongly on the application requirements. Pure data collection applications tend to work with hierarchical topologies with cluster heads aggregating and relaying data as it is generated. In such systems information-processing requirements are simply sense data collection or at most calculation of aggregates. This is not always the case, however, as some WSN systems like the Redwood Forest deployment [103] which for the most part exhibit a flat topology are also used for data collection. In such systems energy conservation is either not a major concern or techniques for optimizing query processing are incorporated to minimize energy consumption.

Again, cluster-based topologies (single or multihop) appear to be more amenable to logical region-based query processing approaches. Region leaders can be considered somewhat similar to cluster heads in terms of functionality therefore enabling a mapping of the logical construct to the network construct. It is expected that regions, in this type of architecture will be easier to implement and test experimentally.

A review of candidate query processing architectures is given below.

10.4.4.3. Query Processing Architectures

Database-Type Architectures: Database-type query processing systems are some of the most popular put forward in the literature. [115] propose a generic database-based query processing architecture for sensor networks (see Fig. 10.16) and describe the components they feel are required at each layer to make the architecture practical.

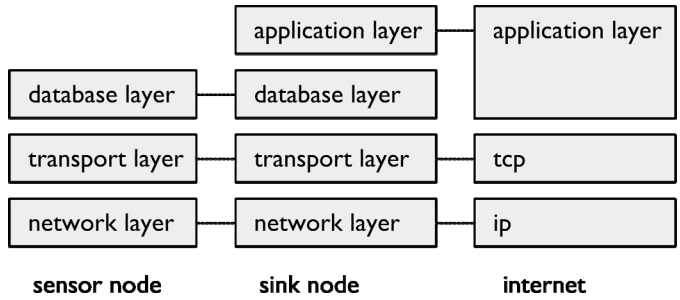

Fig. 10.16. A database-based query processing architecture based on [115].

Application Layer: at the base station this layer provides an interface for posing of queries to the sensor network. These queries can be represented either as an SQL-type message or even a SOAP-like web service. The application layer in the base station is designed to communicate with the TCP/IP stack and so an Internet-based host can easily access query results.

Database Layer: on the base station, the database layer serves the application layer by receiving queries from it and returning results to it. The database layer on the base station contains a number of components including: a Parser – creates the query tree based on the query received from the application layer; Catalog Manager – maintains the relational schema, location and distribution of sensors and monitors nodes' status, for instance, node power levels and node connectivity; a Query Optimizer - generates the optimized query execution plan.

Due to resource limitations, the database layer's function at node level is much more limited. It receives the query tree either from the base station or another node and may need to further optimize the tree based on local catalog information. The database layer returns results in the form of relations.

Transport Layer: this layer provides for efficient end-to-end communication. At the base station it should be able to communicate with TCP entities on Internet hosts although alternative methods may be needed for nodes since nodes are address-less.

Network Layer: this layer should provide energy-efficient and data-centric routing algorithms. The routing decisions made should be based not only on the current network conditions but also the given query execution plan.

10.4.4.4. Example Systems

Query processing systems like TinyDB [104] and Cougar [97] are just some of the database-type systems put forward in the literature. They all follow to varying extents the architecture put forward by [115]. The query processing architecture is similar for both, consisting of server-side software running at a base station and responsible for

parsing and delivering queries into the network, as well as collecting the results as a stream out of the network.

The architecture of the TinyDB system, both server-side and node-side, is illustrated in Fig. 10.17. Architecturally the query layer sits between the application and network layers in the case of the server side component (base station) and sits on top of the network layer at node level. As in Fig. 10.16, the application layer allows the construction and posing of queries with the database layer, here referred to as the query layer, below parsing the query, constructing query execution plans and delivering the results to the network. The query layer at the server side also collects the results for presentation to the application layer.

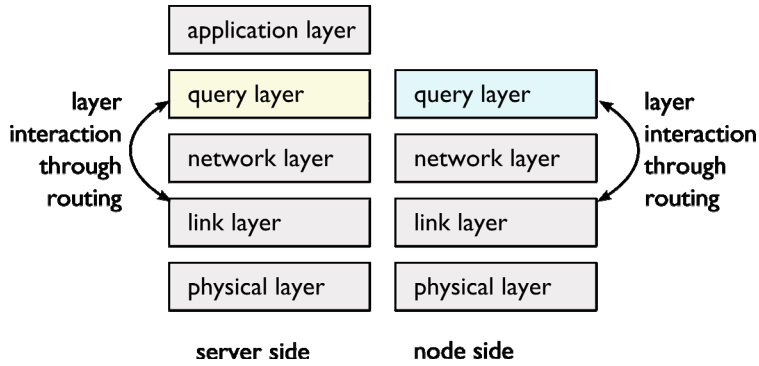

Fig. 10.17. A high level view of the query processing architecture used in Cougar and TinyDB.

The query layer houses a number of components some of which have already been described in the architecture put forward by [115] and are illustrated in Fig. 10.18.

The query layer is the backbone of the query processing system and provides a number of critical functions:

- The processing of the query itself. The query layer uses sophisticated techniques like catalogue management, query optimization to abstract the user from the physical details of the network in processing queries.
- Performing in-network processing to reduce communication given the need to preserve resources like energy and bandwidth. To do this the query layer generates different query plans with different trade offs for requirements such as accuracy, energy consumption and latency.
- Interacts with the routing layer to facilitate in-network processing. With traditional routing, the network layer on a node will automatically forward packets to the next hop towards the destination with the upper layers, completely unaware that data packets are moving through the node. In implementing in-network aggregation, for example, the query layer needs to be able to communicate with the network layer when it wants to intercept packets destined for the sink or leader node. In the

processing system described above, filters are used to first access the packets then modify and delete if necessary before passing on to the next hop onto the destination.

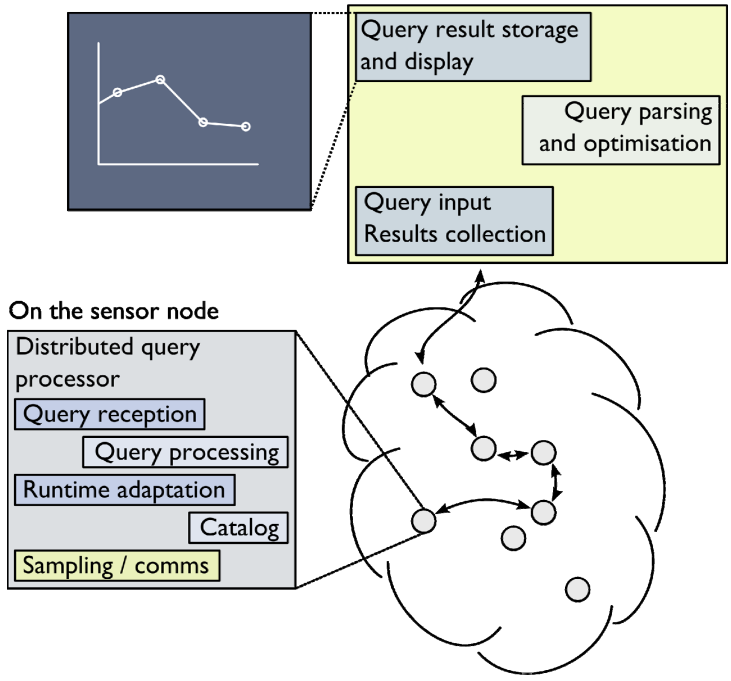

Fig. 10.18. A detailed view of the query processing architecture of TinyDB and Cougar [61].

Although they can probably be implemented on both hierarchical and flat network topologies, these database type approaches have so far been implemented on flat networks.

Middleware Architectures: Another approach to query processing in WSNs is that of query-based middleware systems. Two of the more popular are DSWare and SINA described in Sections 10.2 and 10.3 of the chapter. Although inspired by database systems in that they use SQL-based languages for construction of queries, these systems are considered middleware approaches as they also provide services that go beyond database query processing systems. Architecturally, these middleware approaches are similar to the database approaches. In both cases, the middleware layer sits between the application and network layers and in addition to providing facilities for query processing, supplies the services necessary for ensuring adequate WSN functionality. In the case of DSWare, the application layer allows the construction of SQL-based event type queries that are then registered, parsed, and used to generate execution plans. The underlying DSWare middleware layer is then responsible for registering and executing these events. The DSWare layer also interacts with the routing layer for improving network functioning, for example, improving power awareness.

SINA also comprises a middleware layer that sits above the network layer and provides an application-programming interface to the middleware layer via SQTL scripts. The middleware layer runs on all nodes in the network and like the database approach it selects the most suitable distribution of the query-based on the current network status and the type of query being executed. In addition to query processing functions, SINA also provides services for accessing sensor hardware and communication and supporting changing network topology (for example, sink mobility).

Both DSWare and SINA promote processing of aggregation-based queries and are suited to cluster-based topologies.

10.4.4.5. In-Network Processing Techniques for WSNs

Experiments have shown that the major part of power consumption costs is due to communication rather than computation [92-94, 107] describe in-network processing as the pushing of operations, particularly selections and aggregation of data, into the network to reduce communication. [97] describe it as the moving of computation from outside to inside the network in an effort to reduce energy consumption and improve network lifetime. The main goal in effect is to reduce communication in exchange for some form of computation within the network. In-network processing techniques are therefore accepted as being essential to improving energy efficiency in gathering information within the WSN and critical to any system that has improved network lifetime and energy efficiency as goals. The two most common in-network processing techniques used for reducing communication in WSN systems are packet merging and partial aggregation [99]. In addition, a number of other widely used techniques exist. These are discussed below.

Filtering: Filters are constructs used within the network (on nodes) to assist in processing [97]. A filter registers what kind of data it handles through matching and is triggered each time that type of data enters the node. Once it has been activated, the filter can manipulate the data by, for example, determining whether it is forwarded on or even generating a new message. Filtering is sometimes used in conjunction with data aggregation. [97] proposed using a filter to detect concurrent detections of four-legged animals from different sensors. It could then record what the desired interval was and ensure that only one response per interval, suppressing responses from other sensors. [116] take advantage of event properties of monitoring queries, and carry out data filtering during in-network processing.

Packet Merging: Packet merging, described by [99] combines a number of smaller records into a few larger ones in a bid to reduce the number of packets needing to be transmitted and in turn reducing the cost of communication as the cost of transmitting several smaller packets is greater than transmitting just one large packet. The Cougar query processing system is an example of a system that uses packet merging for reducing communication in processing some types of aggregate queries.

Partial Aggregation: Partial aggregation refers to the computation of intermediate results as the node receives readings. Communication is reduced as this combined partial result is transmitted on instead of all individual records. Aggregation can be achieved in various ways based on the type of correlation that exists in the WSN. This may be spatial correlation (due to physical proximity of nodes and therefore similarity in the readings), temporal correlation (if there is little variation in the sensed attribute based on the sampling frequency) or correlation in the data itself (due to overlap in sensor coverage) [117].

Tree-based Aggregation: In tree-based aggregation one node is designated the root node. A broadcast message is sent out with data on the ID of the node sending the message and its depth in the tree. In the case of the root the depth would be zero. When nodes receive this message they assign themselves a level by adding one to the value in the message received and assign the ID in the message as their parent. Broadcasting of the message continues until all nodes within range have received it and assigned themselves a level and parent. TinyDB is one query processing system that uses a tree-based aggregation service called TinyAGgregation (TAG) [118, 99].

There are a number of advantages to the TAG approach. First, a reduction in the number of communications needed to calculate aggregates when compared to aggregation that occurs at one centralized location. Second, as data moves up the tree back to the base station nodes usually are required to transmit a maximum of one message. This is in direct contrast with centralized aggregation methods where the number of transmissions increases dramatically as data moves towards the root node. This can of course lead to a decrease in the lifetime of the network as battery power is quickly drained. Third, it allows aggregation even in networks where connectivity is intermittent or disconnections occur since disconnected nodes can reconnect by listening to other nodes' partial state records as they flow up the tree. This is possible since the partial state record includes information on the query that was issued.

TAG, therefore, presents a simple interface, flexible naming and generic operators for constructing aggregate queries and is not application specific as with other approaches using aggregation like Directed Diffusion [119] and Greedy Aggregation [120]. Further, it separates the logic of aggregation from routing details so that the focus is squarely on the application and leaves routing decisions to the system. This is in contrast to Directed Diffusion, which puts aggregation mechanisms within the routing layer. The TAG approach allows a stream of aggregate values that change as sensor readings and the underlying network layout change and does this in an energy-efficient and bandwidth-efficient way.

TAG, however, does have limitations, as it does not allow joins and cannot respond to events that occur within the network. The authors have indicated that future work will focus on developing an efficient way of aggregating the results of those event-based queries across nodes before transmitting that information to interested nodes.

Cluster-based Aggregation: In some cases nodes are in close physical proximity to each other and queries may need to retrieve information based on this spatial correlation. For example, a query may want to determine the average humidity over a particular area. Typically, clusters are formed consisting of nodes in close proximity based on some metric, for example, signal strength. Clusterheads are elected and act as the data aggregator and router for cluster members. SINA and Cougar, are examples of query processing systems that incorporate cluster-based aggregation in processing aggregate type queries.

Discussion: Given the importance of considering in-network processing when developing efficient WSN applications, this section examined some of the more popular techniques used. Data aggregation is most widely used, with the type of aggregation dependent on the network topology and to some extent the type of queries being issued. Systems that incorporate in-network processing and are therefore information-generating systems (as opposed to simple data gathering) have tended to be built on cluster-based topologies. In heterogeneous configurations, cluster heads can be configured to be more computationally powerful and are conducive to supporting data aggregation. There have, however, been, information-generation systems based on flat topologies that allow tree-based aggregation. Where scalability is an issue, and where in-network processing is computationally intensive, reported systems have had the tendency to lean towards a cluster-based configuration, this being more energy efficient and practical in creating a system with an extended lifetime.

Clustered networks have the advantage of conveniently allowing aggregation at the clusterhead and are most effective in networks that are static and where the cluster structure stays unchanged for a considerable period. With dynamic clusters, however, problems can occur with energy expended in continuously updating nodes in order to keep the clusters consistent with the underlying network topology. Tree-based aggregation is simple and useful but can lead to problems. Given a node failure in the network, for example, a packet lost at a given level of the tree can lead to all data being aggregated up that node's sub-tree being lost as well. This can lead to incomplete data making its way up the tree particularly if nodes only have one route to the sink and connectivity gaps occur.

Having set the requirements, and outlined the topologies, architectures, and protocols in this section, the next section describes a new, hybrid, approach that combines query-based and macroprogramming concepts and evaluates it in light of the requirements previously discussed.

10.4.5. A Distributed Complex Query Processor

The database-type architecture forms the basis for the complex query processing architecture put forward by the authors here. The two-part architecture consists of server-side software, which is accessed by the user and used for constructing and posing declarative-type queries, and node-side software, which is in effect the distributed

complex query processor (DCQP). A key component of the architecture will be the Region and Query Management Layer that will be responsible for not only query processing but will have additional tasks related to the creation, maintenance and update of regions within the network.

A number of factors influenced this selection. First, a key feature of the distributed complex query processor (DCQP) proposed is ease-of-use and familiarity for users. An SQL-based, database-like approach is therefore preferred. Although attractive, a middleware approach was not selected as the aim with the DCQP is to create a system that is separated as far as possible from network level configuration issues. The idea is that the system will sit on top of a functioning network. Second, the need for in-network processing for resolution of complex queries dictates that a query layer that allows in-network processing of queries is essential. The approaches above have shown that distributing the query layer to all nodes and providing functions for query processing and optimization are effective. The DCQP query layer therefore, will allow the formulation of query execution plans, dynamic optimization of these plans based on the attributes of interest and node state as maintained by the regions. Third, the DCQP architecture will incorporate a query layer that is decoupled as far as possible from the underlying layers. The idea here is to make the system as portable as possible by not strictly linking it to any particular routing protocol, for example, but instead identifying features that would make particular protocols more suitable than others.

A number of candidate architectures were investigated as a starting point for a distributed query processing system that incorporates best practices for in-network processing and resolution of complex queries while at the same time allowing the incorporation of novel techniques and strategies for the types of queries proposed. The next subsections describe in more detail the design choices and assumptions made and give reasons for these selections.

10.4.5.1. The Network Model

Key to the research work is the creation of logical regions within the network. These regions constitute a critical component in making a decision on where a query should be disseminated, how the query should be processed within and between regions and where the complex query will be ultimately resolved. The regions will have one “leader node” each which acts as a manager, data aggregator and communicator to a gateway as well as other region leaders when needed. This role is remarkably similar to that of clusterheads in network-level clusters [102]. Additionally, like some clusters, regions should be and are likely to be, in the approach proposed here, dynamic. Consequently, it makes sense that these logical region leaders map to physical clusterheads. These requirements, therefore, directly influence the choice of network model for the proposed work.

First, the network model proposed has a hierarchical topology. Sensors are randomly deployed and the transmission range is identical for all devices. For simplicity, an ideal

MAC layer is assumed and node death considerations are not taken into account at this stage. Also assumed is that a route has been established between nodes in the network and the gateway node. (Gateway node here is defined as the node from which a query is disseminated and through which results are acquired. Gateways are not fixed; any node could potentially become a gateway at some point in time.) To facilitate region-based routing, a cluster-based routing protocol that allows dynamic cluster formation and supports inter-node communication and communication with the external world, is selected. Two possible choices identified so far are the Hybrid Energy-Efficient Distributed Clustering protocol (HEED) presented by [121], which focuses on scalable data aggregation and increased network lifetime; and a dynamic clustering algorithm (DCRR) presented by [122] in which cluster heads are dynamically selected in the region where an event occurs according to their residual energy.

10.4.5.2. A Distributed Complex Query Processing Architecture

The architecture proposed for the DCQP consists of server-side software which is accessed by the user and used for constructing, posing and parsing of declarative-type queries and node-side software which is in effect the distributed complex query processor (DCQP) implementing query processing functions in-network.

Server-side Software: This component will host an application layer that will allow the construction of queries along with an underlying Region Query Management Layer that will be responsible for parsing of these queries and dissemination to the network. Additionally, the server-side software will allow the receipt of query results for both presentation purposes and for persistent storage. This architecture is equivalent to that of TinyDB and Cougar as shown in Fig. 10.17.

A key output of the work described here will be the creation of a system that will allow the user to either specify new logical regions as a component of a query being constructed or make use of regions already in existence within the network. This will involve the extension of a SQL-based query language to incorporate the region construct and the development of an associated parser for the language. The region abstraction will be a part of the query language used, and is essential to query dissemination as well as the creation of query execution plans.

Node-side Software: This component will also host a region query management layer which will from an architectural standpoint sit above the network layer. The node-side component is really a distributed query processor hosted by all nodes in the network.

The region query management layer will deal with query reception, dissemination, processing and delivery of results. In addition, this layer will need to manage the logical regions, which are implemented as part of the complex query. This functionality will be implemented as a region management component of the query management layer itself.

Region management will include the creation, maintenance and updating of region information on each node within the network. This may involve, for example, a region leader upon receipt of a query, informing its members of a new attribute of interest; or perhaps the termination of a previously posed query along with its associated regions; or the handing-off of a region leader's duties to a backup leader when the node's energy level falls below a certain threshold. Once a query is received, execution plans are created dynamically based on the required region formation or existing regions that need to be accessed. Region management is therefore an integral part of query plan formulation and query optimization. The query plan is then executed. Once results have been acquired these are communicated to the server-side component.

As in the Cougar system, the query management layer may have to interact with the routing mechanism in the network layer in order to enable in-network processing, which is a key feature of the region-based query processing approach proposed. This interaction, however, is anticipated to be minimal since a cluster-based routing protocol that maps the logical regions to physical clusters is proposed as a way of minimizing intrusion into the network layer by the query management layer.

Fig. 10.19 shows diagrammatically, the proposed complex query processing system architecture.

In the following section, the DCQP approach is evaluated via simulation.

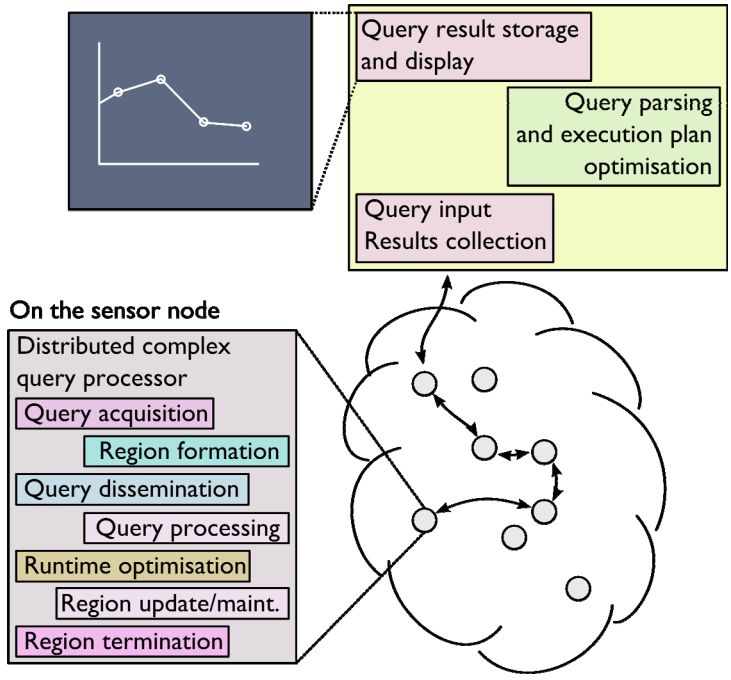

Fig. 10.19. A detail view of the functions provided by the complex query processing system.

10.4.5.3. Acoustic Monitoring Using Region-based Querying

Habitat monitoring is already a rich area of research in the area of WSNs particularly because of the benefits to science and education. Within this group are applications based on acoustic sensing, which are concerned with event detection and classification as well as monitoring and localization. Bioacoustics research is a specific example and acoustic sensing in that context can be used to help scientists acquire acoustic data which can then be used to distinguish between animals, species and census counts.

The processing required for such applications usually involves complex signal processing operations and therefore present a number of challenges and requirements that are not evident in traditional monitoring systems. The challenges include the heavy computation needed due to very high data rates and the need for development of specific algorithms to facilitate on-line processing of very large amounts of data.

One such acoustic monitoring application, VoxNet focused on creating a system that allowed the detection of marmot alarm calls. The work was informed by the requirements of scientists to detect these marmots in the field and then localize their positions. These calls, therefore, were used to help determine the location of the animal at the time the call was detected, relative to known burrow locations. Although for some systems simple recording and offline analysis of the data fulfils application requirements, in the case of this bioacoustic monitoring system it was important that the system produce timely results. The acoustic event detection and localization application consisting of eight nodes was deployed over an area of about 9800 sq meters (2.4 acres). A gateway node was then positioned about 200 m away from the nearest node. A map of the deployment is displayed in Fig. 10.20.

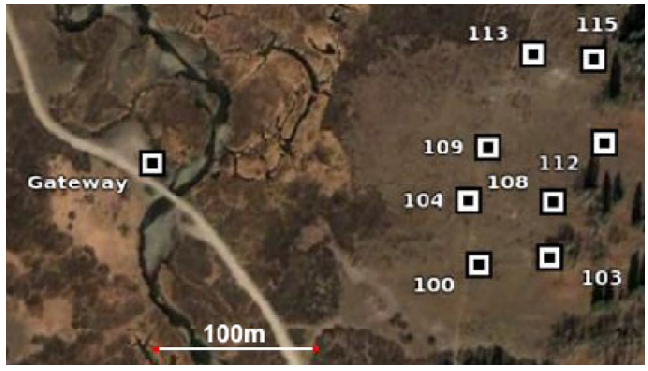

Fig. 10.20. Map of VoxNet deployment area.

Localization of Acoustic Signals – the VoxNet approach: Once a node detects an acoustic signal, it is timestamped and an attempt made to determine the location of the sound. The localization algorithm used, the Approximated Maximum Likelihood algorithm (AML) [123] functions as follows: a stream of audio is processed through a

"fast path" detector to identify possible alarm calls; the algorithm then estimates bearing to the caller from multiple points followed by fusion of those estimates to produce an estimate of the sound location.

The algorithm was expressed as a WaveScript program, which is a logical dataflow graph of stream operators, connected by streams. Once the program runs on these input streams, results are streamed back to data storage components, in this case the gateway or sink node. This execution of the WaveScript program constitutes the extent of in-network processing with the result being an estimate of the direction of arrival (DOA) of the acoustic signal. Nodes that detect acoustic events then send timestamped, DOA values to the sink. In some cases, depending on time availability, AML may not be executed on the node to produce DOA estimates and instead streams of raw detections are sent instead. The size of a DOA packet is 800 bytes as compared to 32 kB for a raw detection.

In the application, a minimum of 3 acoustic signals or DOA estimates are needed to localize a sound. At the sink, these DOA estimates along with the timestamps are used to determine, first, whether the signals are indeed from the same acoustic event and if they are, they can be combined to determine the location at which the sound occurred.

For purposes of this work, metrics from the VoxNet deployment representing the number of data packets transmitted in localizing one event, the number of hops over which these were sent as well as the total distance travelled were used. A comparison was then made between the real-life deployment results and the results of the simulations incorporating in-network, region-based processing.

A Region-based Approach to Acoustic Signal Localization: The authors maintain that an approach that incorporated in-network processing using the concept of dynamic regions would be just as effective, if not more, than sending all raw data or partially processed data, that is, DOA estimates, back to the gateway or sink for analysis. In this set of experiments, given the limitations of the simulator used, estimation of query processing times and generation of random events were not possible. Instead, metrics not dependent on time, like packet size, hop count and number of packets transmitted were used to compare the efficiency and feasibility of the approach as compared to that taken in the VoxNet application in the field.

Simulation Scenario and Setup: For all simulations, a number of controls were maintained:

- The deployment of nodes in the simulator mirrored the actual real-life deployment topology. Nine nodes including the sink were positioned in the simulation window at the same relative locations as they were positioned in the actual deployment.
- All nodes were initialized with an equal and consistent amount of energy.
- Radio broadcast range was set at a 200 unit radius (a unit corresponding to a meter). This was the broadcast range of nodes in the VoxNet deployment.
- Each node's location was unique within the two-dimensional plane.

- A list of acoustic events, the ID of nodes that should detect these events and the intervals at which these events are to occur was defined prior to the start of each simulation. Lists contained between 3 and 5 events.

10.4.5.4. Region Based Query Resolution

A continuously running event detection task (referencing the event list) is implemented on all nodes. At each time interval (a system clock tick), each node checks the event list. If an event for that node exists and the current time matches the scheduled event then an acoustic event is triggered.

The detecting node logs the time of the event and checks if it has any related inquiries in its "inquiry cache". An inquiry indicates that another node has previously detected a possibly correlated event and has requested and is possible awaiting a response. If a node with an inquiry message in its inquiry cache detects a related event (that is, it is within the valid time slot) it sends a response to the node that sent the inquiry. If the node detecting the event does not have any inquiries, it sets itself as a region leader and sends an inquiry message to its one hop neighbours. This inquiry message contains the ID of the node that has detected the event and the event's timestamp. A node receiving this message registers the inquiry if it has not had an event that matches that inquiry. A matching or related event is one that falls within a time slot that was determined experimentally using the simulator. If the node detecting the event does have an inquiry, it checks to see if its event has occurred within the valid time slot (a time range is used to indicate whether two or more signals are correlated and therefore considered as relating to the same acoustic event; this is referred to as the valid time slot) and a "DATA" message is transmitted to the node who sent the inquiry message (the region leader). Any event occurring out of the valid time slot will not be sent to the region leader in response to its inquiry message.

The region leader, upon receipt of a DATA message, records that message and if it has already received the minimum (3) required or more sends the result to the sink via a RESULT message. In the simulation, if the region leader has already received 3 messages it still waits for a period of time before sending a result on. The reason is that the greater the number of messages, the better the accuracy of the measurement (this is also the model used in the VoxNet system). The time a region leader waits was again determined through use of the simulator although in the real life deployment this value was determined experimentally and is referred to as a fuzz factor.

The simulation was allowed to run until either a result was sent to the sink or if the time for detection of a particular event had elapsed. For example, an insufficient number of detections (less than 3) were detected within the valid time slot. The results of this simulation in comparison to the in-the-field results are analyzed in Sections 10.4.5.5. and 10.4.5.6.

10.4.5.5. Efficiency of Region-based Querying

The results obtained indicate that regions can be used to support in-network query processing. In analyzing how much more efficient, if at all, the region-based approach is as compared to the in-network aggregation and centralized approaches, energy efficiency is evaluated using one parameter, that is, the number of messages generated in returning a query response to the sink.

The results of running a number of 50-node simulations indicate that on average region-based query processing resulted in an almost 86 % decrease in the number of transmissions over the centralized approach and a 68 % decrease on average over the approach using in-network aggregation.

For a 100-node simulation the results were equally impressive with over 91 % reduction over the centralized approach and an almost 65 % decrease when compared to the simulation using in-network aggregation. For a 150-node simulation the results for the region-based approach were more marked when compared to the centralized approach with over 95 % reduction in the number of messages while there was an over 45% decrease when compared to the simulation using in-network aggregation. The results are summarised in Fig. 10.21.

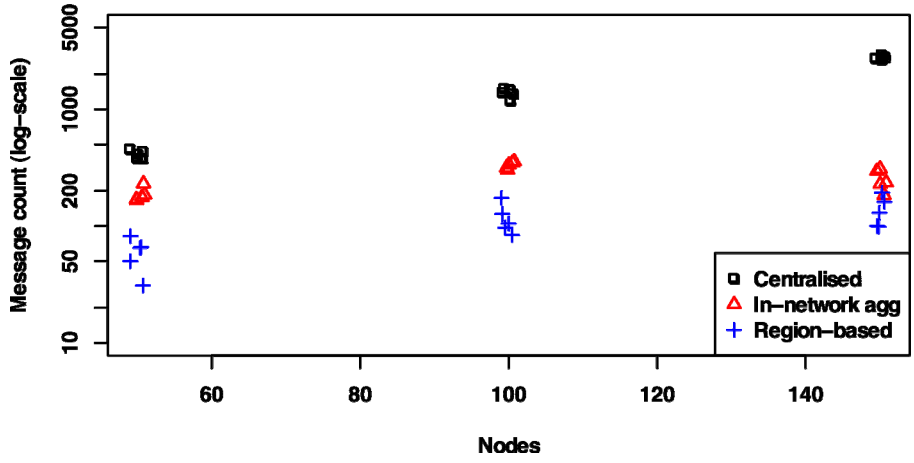

Fig. 10.21. Comparison of number of messages required for query resolution in 50, 100 and 150-node networks.

The results showed that the efficiency of the region-based querying approach increased as the size of the network increased when compared to the centralized approach, going from 86 % to 91 % to 95 % for a 50, 100 and 150 node network respectively. In comparison to the in-network aggregation approach, however, this was not the case. Although communication was less, the relative percentages showed an overall decrease, going from 68 % to 65 % to 45 % for a 50, 100 and 150 node network respectively. It

would have been interesting to run these algorithms on even larger networks in determining whether this trend would continue, however, the limitation of the simulator used made this impossible at this time. Based on these results, however, although feasible, the scalability of the region-based approach is an issue to be considered and examined more closely in future work.

10.4.5.6. Effectiveness of Region-based Querying

Again, the results obtained indicate that regions can be used to support in-network query processing and are a viable approach in the context of a real-life deployment scenario. In analyzing the effectiveness and feasibility of the region-based processing approach compared to that used in the VoxNet application a number of measures were taken. First, the number of data packets was measured and along with the packet size was used to calculate the total data transmission required for query resolution. This was done in multiple runs for cluster/region sizes of 3 and 4 nodes in the case of the VoxNet and region-based approaches respectively, and an average taken. In addition, the average total data transported was calculated for the VoxNet system in scenarios where all raw data was sent to the sink and also in cases where in-network processing for DOA estimation was carried out.

The results, displayed in Fig. 10.22, clearly indicate that in terms of data transmission savings the region-based approach exhibits an advantage over the query processing approach used in the VoxNet system. This was evident even when some in-network processing was carried out. For a 3 and 4-node region there was approximately a 28 % and 38 %, reduction respectively in the amount of data transmitted over the VoxNet approach with in-network processing. The decrease is even more marked with figures of 61 % and 67 % for the VoxNet processing approach with no in-network analysis.

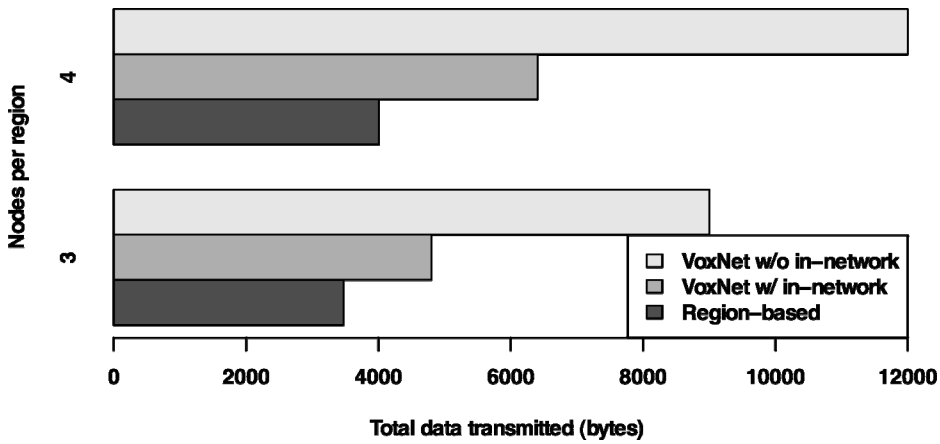

Fig. 10.22. Total amount of data transmitted in query resolution.

The simulation again confirmed the feasibility of the approach although a number of issues need to be investigated in future work. One, as identified before, is the scalability of the approach. In this simulation the network was quite small (9 nodes) and the regions created as a consequence also limited in size. An interesting exercise for the future would be to investigate the approach using larger regions and also to test the region query-processing algorithm in the field.

10.5. Conclusions

An extensive literature review has shown the absence of complex query mechanisms in the query processing systems in use today. The review did highlight, however, applications in which these queries could be used along with in-network processing to both extend and improve WSNs' deployment value.

Both the usefulness of complex queries and the feasibility of using the proposed logical abstractions (called regions) to facilitate query processing in WSN systems have been positively assessed. The preliminary experiments have produced promising results but also highlighted a number of areas that present the community with open research questions: What are the means for efficient query-based region setup ? How can dynamic regions be implemented which can change while query processing is occurring? (This becomes extremely important in cases where regions are established over an attribute that is changing with time.) What is the network size/scale at which the cost of region set-up is justified? (Combining it with cluster formation to reduce energy consumption is perhaps one approach.) For simplicity, the work here considered a scenario where a node could only be a member of a maximum of one region. Ideally, nodes should be able to be members of multiple regions. A node's single region membership can be an issue in facilitating the processing of multiple queries simultaneously or processing complex queries containing subqueries with additional attributes of interest. Moreover, here, once a response was sent to the sink the querying process terminated. In future work, the idea of the sink issuing further queries needs to be investigated as well as inter-region communication. Finally, what are the hardware support requirements to enable a system such as the one here be reliably deployed ?

This is clearly a fruitful domain and although much has been achieved, it is the authors' belief that there is much still to be done to make WSN information extraction acceptable and accessible to a wider audience.

References

[1]. A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson, Wireless sensor networks for habitat monitoring, in *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, United States, 2002, pp. 88–97.

[2]. R. Guy, B. Greenstein, J. Hicks, R. Kapur, N. Ramanathan, T. Schoellhammer, T. Stathopoulos, K. Weeks, K. Chang, L. Girod, D. Estrin, Experiences with the extensible sensing system, *Tech. Rep*. 61, CENS, 2006.

[3]. K Langendoen, A. Baggio, O. Visser, Murphy loves potatoes experiences from a pilot sensor network deployment in precision agriculture, in *Proceedings of 20th International Parallel and Distributed Processing Symposium, IPDPS' 2006*.

[4]. Thaddeus R. F. Fulford-Jones, Gu-Yeon Wei, and Matt Welsh, A portable, low-power, wireless two-lead EKG system, in *Proceedings of Annual International Conference of the IEEE Engineering in Medicine and Biology*, San Francisco, CA, United States, Vol. 26, III, 2004, pp. 2141–2144.

[5]. Mauricio Castillo-Effen, Daniel H. Quintela, Ramiro Jordan, Wayne Westhoff, and Wilfrido Moreno, Wireless sensor networks for flash-flood alerting, *In Proceedings of the IEEE International Caracas Conference on Devices, Circuits and Systems, ICCDCS'04,* Dominican Republic, 2004, pp. 142–146.

[6]. K. Pister, The 29 palms experiment - tracking vehicles with a UAV-delivered sensor network, 2007, http://robotics.eecs.berkeley.edu/~pister/29Palms0103

[7]. *HPWREN*, High performance wireless research and education network, 2007, http://en.wikipedia.org/wiki/High_Performance_Wireless_Research_and_Education_Network

[8]. *ROADNet*, Real-time observatories, applications and data management network, 2007, http://roadnet.ucsd.edu

[9]. W. R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences,* Vol. 2, Maui, HI, USA, 2000, p. 10.

[10]. S. Madden, R. Szewczyk, M. J. Franklin, D. Culler, Supporting aggregate queries over ad-hoc wireless sensor networks, in *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, USA, 2002, pp. 49–58.

[11]. G. J. Pottie, W. J. Kaiser, Wireless integrated network sensors, *Communications of the ACM,* 43, 5, 2000, pp. 51–8.

[12]. P. Bonnet, J. Gehrke, P. Seshadri, Querying the physical world, *IEEE Personal Communications,* 7, 5, 2000, pp. 10–15.

[13]. Y. Yao, J. Gehrke, Query processing for sensor networks, *In Proceedings of Conference on Innovative Data Systems Research (CIDR),* 2003.

[14]. J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, D. Ganesan, Building efficient wireless sensor networks with low-level naming, *Operating Systems Review (ACM)*, Vol. 35, Banff, Alta., Canada, 2002, pp. 146–159.

[15]. R. Newton, Compiling Functional Reactive Macroprograms for Sensor Networks, Masters Thesis, *Massachusetts Institute of Technology*, 2005.

[16]. R. Newton, G. Morrisett, M. Welsh, The regiment macroprogramming system, in *Proceedings of the 6th International Conference on Information Processing in Sensor Networks,* 2007, Cambridge, Massachusetts, USA, pp. 489-498.

[17]. H. Qi, Xiaoling Wang, and K. Chakrabarty, Multisensor data fusion in distributed sensor networks using mobile agents, in *Proceedings of the 5th International Conference on Information Fusion*, Annapolis, MD., 2001.

[18]. C. L. Fok, G. C. Roman, and C. Lu, Mobile agent middleware for sensor networks: an application case study, in *Proceedings of Fourth International Symposium on Information Processing in Sensor Networks* (IEEE Cat. No. 05EX1086), Los Angeles, CA, USA, 2005, pp. 382–387.

[19]. M. Chen, T. Kwon, Y. Yuan, and V. C. M. Leung, Mobile agent based wireless sensor networks, *Journal of Computers*, 1, 1, 2006, pp. 14–21.

[20]. Q. Wu, N. S. V. Rao, J. Barhen, S. S. Iyenger, V. K. Vaishnavi, H. Qi, and K. Chakrabarty, On computing mobile agent routes for data fusion in distributed sensor networks, Knowledge and Data Engineering, *IEEE Transactions on*, 16, 6, 2004, pp. 740–753.

[21]. D. Massaguert, C. Fok, Nalini Venkatasubramanian, G. Roman, C. Lu, Exploring sensor networks using mobile agents, in *Proceedings of the International Conference on Autonomous Agents,* Hakodate, Japan, 2006, pp. 323–325.

[22]. R. Tynan, G. M. P. O'Hare, D. Marsh, and D. O'Kane, Multi-agent system architectures for wireless sensor networks, in *Proceedings of 5th International Conference on Computational Science, ICCS 2005,* (Lecture Notes in Computer Science Vol. 3516), Atlanta, GA, USA. Springer-Verlag, 2005, pp. 687–94.

[23]. D. Marsh, R. Tynan, D. O'Kane, and G. M. P. O'Hare, Autonomic wireless sensor networks, *Engineering Applications of Artificial Intelligence,* 17, 7, 2004, pp. 741–748.

[24]. Robert Wesson, Frederick Hayes-Roth, John W. Burge, Cathleen Stasz, and Carl A. Sunshine, Network structures for distributed situation assessment, *IEEE Transactions on Systems, Man and Cybernetics, SMC-11*, 1, 1981, pp. 5–23.

[25]. D. N. Jayasimha, S. Sitharama Iyengar, and R. L. Kashyap, Information integration and synchronization in distributed sensor networks, *IEEE Transactions on Systems, Man and Cybernetics,* 21, 5, 1991, pp. 1032–1043.

[26]. L. Prasad, S. S. Iyengar, R. L. Kashyap, and R. N. Madan, Functional characterization of sensor integration in distributed sensor networks, in *Proceedings of the Fifth International Parallel Processing Symposium* (Cat. No. 91TH0363-2), Anaheim, CA, USA, 1991, pp. 186–93.

[27]. H. Qi, S. Iyengar, and K. Chakrabarty, Multiresolution data integration using mobile agents in distributed sensor networks, Systems, Man and Cybernetics, Part C: Applications and Reviews, *IEEE Transactions on*, 31, 3, 2001, pp. 383–391.

[28]. Yao Yuxia, Tang Xueyan, and Lim Ee-Peng, In-network processing of nearest neighbor queries for wireless sensor networks, Database Systems for Advanced Applications, in *Proceedings of 11th International Conference, DASFAA'2006.* (Lecture Notes in Computer Science Vol. 3882), Singapore, 2006, pp. 35–49.

[29]. K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie, Protocols for self-organization of a wireless sensor network, *IEEE Personal Communications*, 7, 5, 2000, pp. 16–27.

[30]. H. Qi, S. S. Iyengar, and K. Chakrabarty, Distributed multi-resolution data integration using mobile agents, in *Proceedings of IEEE Aerospace Conference-2001* (Cat. No. 01TH8542), Big Sky, MT, USA, 2001, pp. 3–1133.

[31]. Hairong Qi, Yingyue Xu, and Xiaoling Wang, Mobile-agent-based collaborative signal and information processing in sensor networks, in *Proceedings of the IEEE,* 91, 8, 2003, pp. 1172–1183.

[32]. Hairong Qi and Wang Feiyi, Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks, in *Proceedings 13th International Conference on Wireless Communications,* Calgary, Alta., Canada, 2001, pp. 147–53, TRLabs/Univ. Calgary.

[33]. H. Qi, Xiaoling Wang, S. Sitharama Iyengar, K. Chakrabarty, High performance sensor integration in distributed sensor networks using mobile agents, *International Journal of High Performance Computing Applications,* 16, 3, 2002, pp. 325–335.

[34]. G. M. P. O'Hare, D. Marsh, A. Ruzzelli, and R. Tynan, Agents for wireless sensor network power management, in *Proceedings of International Workshop on Wireless and Sensor Networks (WSNET-05),* Oslo, Norway, 2005.

[35]. R. Tynan, D. Marsh, D. O'Kane, G. M. P. O'Hare, Intelligent agents for wireless sensor networks, in *Proceedings of the International Conference on Autonomous Agents,* Utrecht, Netherlands, 2005, pp. 1283–1284.

[36]. C. Min, K. Taekyoung, C. Yanghee, Data dissemination based on mobile agent in wireless sensor networks, in *Proceedings of Conference on Local Computer Networks, LCN,* Sydney, Australia, 2005, pp. 527–528.

[37]. L. Szumel, J. LeBrun, J. D. Owens, Towards a mobile agent framework for sensor networks, in *Proceedings of 2nd IEEE Workshop on Embedded Networked Sensors, EmNetS-II,* Sydney, Australia, 2005, pp. 79–87.

[38]. Philip Levis and David Culler, Mate: A tiny virtual machine for sensor networks, in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, San Jose, CA, United States, 2002, pp. 85–95.

[39]. B. Karlsson, Oscar Bäckström, Wlodek Kulesza, and Leif Axelsson, Intelligent sensor networks: An agent-oriented approach, in *Proceedings of REALWSN,* Stockholm, Sweden, 2005.

[40]. Jade, http://jade.tilab.com, 2006.

[41]. V. K. Vaishnavi, Q. Wu, N. S. V. Rao, H. Qi, K. Chakrabarty, S. S. Iyenger, and J. Barhen, On computing mobile agent routes for data fusion in distributed sensor networks, *IEEE Transactions on Knowledge and Data Engineering,* 16, 6, 2004, pp. 740–53.

[42]. Richard Tynan, Antonio Ruzzelli, and G. M. P. O'Hare, A methodology for the deployment of multi-agent systems on wireless sensor networks, in *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering, (SEKE '07)*, Taiwan, China, IJSEKE press, 2005.

[43]. Kenneth N. Ross, Ronald D. Chaney, George V. Cybenko, Daniel J. Burroughs, and Alan S. Willsky, Mobile agents in adaptive hierarchical bayesian networks for global awareness, in *Proceedings of the IEEE International Conference on Systems,* Man and Cybernetics, San Diego, CA, USA, 1998, pp. 2207–2212.

[44]. M. Mertsock, D. Stawski, Wireless sensor nodes as intelligent agents: Using expert systems with directed diffusion, 2005, www.mertsock.com/downloads/wsn-ia-dd.pdf

[45]. K. Romer, O. Kasten, F. Mattern, Middleware challenges for wireless sensor networks, *SIGMOBILE Mob. Comput. Commun. Rev.,* 6, 4, 2002, pp. 59–61.

[46]. Ryo Sugihara and Rajesh K. Gupta, Programming models for sensor networks: A survey, *ACM Transactions on Sensor Networks,* 4, 2, 2008.

[47]. Mate, Mate: Building application-specific sensor network language runtimes, 2003, http://www.cs.berkeley.edu/~pal/research/mate.html

[48]. Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu, Rapid development and flexible deployment of adaptive wireless sensor network applications, in *Proceedings of International Conference on Distributed Computing Systems,* Columbus, OH, United States, 2005, pp. 653–662.

[49]. Liu Ting and M. Martonosi, Impala: a middleware system for managing autonomic parallel sensor systems, *SIGPLAN Notices*, 38, 10, 2003, pp. 107–118.

[50]. A. Boulis, C. Han, R. Shea, M. B. Srivastava, Design and implementation of a framework for efficient and programmable sensor networks*,* in *Proceedings of the 1st International Conference on Mobile Systems,* Applications and Services, San Francisco, California, 2003, pp. 187-200.

[51]. A. Boulis, C. Han, R. Shea, M. B. Srivastava, Sensorware: Programming sensor networks beyond code update and querying, *Pervasive and Mobile Computing*, 3, 4, 2007, pp. 386–412.

[52]. C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. Murphy, G. Pietro Picco, Mobile data collection in sensor networks: The tinylime middleware, *Pervasive and Mobile Computing,* 1, 4, 2005, pp. 446–469.

[53]. A. L. Murphy, G. P. Picco, G. C. Roman, Lime: a coordination model and middleware supporting mobility of hosts and agents, *ACM Transactions on Software Engineering and Methodology,* 15, 3, 2006, pp. 279–328.

[54]. A. Silberstein, Push and pull in sensor network query processing, in *Proceedings of Southeast Workshop on Data and Information Management (SWDIM '06),* Raleigh, North Carolina, 2006, http://www.cs.duke.edu/~adam/

[55]. P. Bonnet, J. Gehrke, and P. Seshadri, Towards sensor database systems, Mobile Data Management, in *Proceedings of the 2nd International Conference, MDM 2001,* (Lecture Notes in *Computer Science* Vol., 1987), Hong Kong, China, Springer-Verlag, 2001, pp. 3–14.

[56]. R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, in *Proceedings of the 1st International Conference on Peer-to-Peer Computing,* Linkoping, Sweden, 2002, pp. 101–102.

[57]. P. Bonnet, P. Seshadri, Device database systems, in *Proceedings of the International Conference on Data Engineering,* San Diego, California, 2000, p. 194.

[58]. R. Govindan, J. M. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker, The sensor network as a database, Technical Report 0-771, Computer Science Dept., *University of Southern California.,* 2000.

[59]. Y. Yao, J. Gehrke, The cougar approach to in-network query processing in sensor networks, *SIGMOD Rec.,* 31, 3, 2002, pp. 9–18.

[60]. P. Seshadri, Predator: A resource for database research, *SIGMOD Rec,* 27, 1, 1998, pp. 16–20.

[61]. J. Gehrke, S. Madden, Query processing in sensor networks, *IEEE Pervasive Computing,* 3, 1, 2004, pp. 46–55.

[62]. A. Demers, J. Gehrke, R. Rajmohan, N. Trigoni, Y. Yong, The cougar project: a work-in-progress report, *SIGMOD Record*, 32, 4, 2003, pp. 53–59.

[63]. S. Nath, Y. Ke, P. B. Gibbons, B. Karp, and S. Seshan, Irisnet: An architecture for enabling sensor-enriched internet service, Technical IRP-TR-03-04, *Intel Research*, June 2003.

[64]. Shen Chien-Chung, C. Srisathapornphat, and C. Jaikaeo, Sensor information networking architecture and applications, *IEEE Personal Communications*, 8, 4, 2001, pp. 52–59.

[65]. L. B. Ruiz, J. M. Nogueira, and A. A. F. Loureiro, Manna: a management architecture for wireless sensor networks, *IEEE Communications Magazine*, 41, 2, 2003, pp. 116–125.

[66]. L. Shuoqi, S. H. Son, J. A. Stankovic, Event detection services using data service middleware in distributed sensor networks, in *Proceedings of Information Processing in Sensor Networks. 2nd International Workshop, IPSN 2003,* Lecture Notes in Computer Science, Palo Alto, USA, Vol. 2634, 2003, pp. 502–517.

[67]. S. Madden, M. J. Franklin, Fjording the stream: an architecture for queries over streaming sensor data, in *Proceedings of 18th International Conference on Data Engineering,* San Jose, CA, USA, 2002, pp. 555–566.

[68]. S. R. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, Tinydb: an acquisitional query processing system for sensor networks, *ACM Transactions on Database Systems*, 30, 1, 2005, pp. 122–73.

[69]. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, System architecture directions for networked sensors, in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS,* Cambridge, MA, 2000, pp. 93–104.

[70]. SCADDS, Tinydiffusion, Technical report ISI-TR-638, *USC/Information Sciences Institute,* 2007.

[71]. S. Madden, M. J. Franklin, J. M. Hellerstein, W. Hong, The design of an acquisitional query processor for sensor networks, in *Proceedings of the ACM SIGMOD International Conference on Management of Data,* San Diego, CA, United States, 2003, pp. 491–502.

[72]. S. Ganeriwal, R. Kumar, M. B. Srivastava, Timing-sync protocol for sensor networks, in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems SenSys'03,* Los Angeles, CA, United States, 2003, pp. 138–149.

[73]. J. M. Hellerstein, H. Wei, S. Madden, K. Stanek, Beyond average: toward sophisticated sensing with queries, in *Proceedings of Information Processing in Sensor Networks. Second International Workshop, IPSN'2003.* (Lecture Notes in Computer Science, Vol. 2634), Palo Alto, CA, USA, 2003, pp. 63–79.

[74]. P. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden, Task: Sensor network in a box, in *Proceedings of the 2nd European Workshop on Wireless Sensor Networks, EWSN 2005,* Istanbul, Turkey, 2005, pp. 133–144.

[75]. I. Chatzigiannakis, G. Mylonas, S. Nikoletseas, Jwebdust: A java-based generic application environment for wireless sensor networks, *Lecture Notes in Computer Science,* Vol. 3560, Marina del Rey, CA, United States, 2005, pp. 376–386.

[76]. N. Sadagopan, B. Krishnamachari, A. Helmy, Active query forwarding in sensor networks, *Ad Hoc Networks,* 3, 1, 2005, pp. 91–113.

[77]. K. Whitehouse, C. Sharp, Eric Brewer, D. Culler, Hood: A neighborhood abstraction for sensor networks, in *Proceedings of the 2nd International Conference on Mobile Systems, Applications and Services MobiSys'2004*, Boston, MA, United States, 2004, pp. 99–110.

[78]. M. Welsh, G. Mainland, Programming sensor networks using abstract regions, in *Proceedings of 1st Symposium on Networked Systems Design and Implementation (NSDI '04),* San Francisco, CA, USA, 2004, pp. 29–42.

[79]. K. Whitehouse, F. Zhao, Jie Liu, Semantic streams: A framework for composable semantic interpretation of sensor data, *Lecture Notes in Computer Science*, Vol. 3868, Zurich, Switzerland, 2006, pp. 5–20.

[80]. R. Poli, W. B. Langdon, Backward-chaining evolutionary algorithms, *Artificial Intelligence,* 170, 11, 2006, pp. 953–982.

[81]. R. Newton, M. Welsh, Region streams: functional macroprogramming for sensor networks, in *Proceedings of the 1st International Workshop on Data Management for Sensor Networks,* Toronto, Canada 2004, pp. 78-87.

[82]. R. Newton, D. Arvind, M. Welsh, Building up to macroprogramming: An intermediate language for sensor networks, in *Proceedings of 4th International Symposium on Information Processing in Sensor Networks, IPSN 2005,* Los Angeles, CA, United States, 2005, pp. 37–44.

[83]. R. Gummadi, O. Gnawali, R. Govindan, Macro-programming wireless sensor networks using Kairos. Distributed Computing in Sensor Systems, in *Proceedings of 1st IEEE International Conference, DCOSS 2005.* Lecture Notes in Computer Science, Marina del Rey, Vol. 3560, CA, USA, 2005, pp. 126–140.

[84]. E. Katsiri and A. Mycroft, Knowledge representation and scalable abstract reasoning for sentient computing using first-order logic, in *Proceedings of 1st Workshop on Challenges and Novel Applications for Automated Reasoning,* Miami, FL, 2002, pp. 73–82.

[85]. T. Daniel, R. M. Newman, E. Gaura, and Mount S, Complex query processing in wireless sensor networks, in *Proc. 2nd ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks, PM2HW2N'07,* 2007, pp. 53–60.

[86]. Argo. Argo: part of the integrated global observation strategy, 2007, www-argo.ucsd.edu

[87]. K. Martinez, R. Ong, J. Hart, Glacsweb: A sensor network for hostile environments, in *Proceedings of 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks,* Santa Clara, CA, United States, 2004, pp. 81–87.

[88]. Grape Networks, Grape networks' climate genie, 2007, http://findarticles.com/p/articles/mi_m0EIN/ is_2007_Nov_12/ai_n21094407

[89]. M. Hirafuji, T. Fukatsu, T. Haoming, T. Watanabe, S. Ninomiya, A wireless sensor network with field-monitoring servers and metbroker in paddy fields, in *Proceedings of World Rice Research Conference,* Tokyo and Tsukuba, Japan, 2004.

[90]. Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin, A wireless sensor network for structural monitoring, in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems,* Baltimore, MD, United States, 2004, pp. 13–24.

[91]. Jerome Peter Lynch, Kincho H. Law, Anne S. Kiremidjian, Thomas Kenny, and Ed Carryer, A wireless modular monitoring system for civil structures, in *Proceedings of the International Society for Optical Engineering, SPIE'02,* Los Angeles, CA, United States, 2002, pp. 1–6.

[92]. Codeblue: Wireless sensor networks for medical care, 2007, http://www.eecs.harvard.edu/~mdw/proj/codeblue

[93]. AID-N. Aid-n Project 2007, http://www.aidn.org.au

[94]. Kim Sukun, Pakzad Shamim, Culler David, Demmel James, Fenves Gregory, Glaser Steven, and Turon Martin, Health monitoring of civil infrastructures using wireless sensor networks, in *Proceedings of the 6th International Symposium of Information Processing in Sensor Networks, IPSN,* April 2007, pp. 254-263.

[95]. B. Son, Yong-Sork Her, and Jung-Gyu Kim, A design and implementation of forest-fires surveillance system based on wireless sensor networks for South Korea mountains, *International Journal of Computer Science and Network Security*, 6, 9B, 2006, pp. 124–130.

[96]. Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook, Firewxnet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments, in *Proc. of the of 4th International Conference on Mobile Systems, Applications and Services (MobiSys 2006),* Uppsala, Sweden, 2006, pp. 28–41.

[97]. Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy, Active query forwarding in sensor networks, *Ad Hoc Networks*, 3, 1, 2005, pp. 91–113.

[98]. Atish Datta Chowdhury and Shivashankar Balu, Consensus: A system study of monitoring applications for wireless sensor networks, in *Proceedings of the Conference on Local Computer Networks,* LCN, Tampa, FL, United States, IEEE Computer Society, Los Alamitos, CA United States, 2004, pp. 587–588.

[99]. I. Carreras, I. Chlamtac, H. Woesner, and H. Zhang, Nomadic sensor networks, in *Proceedings of the 2nd European Workshop on Wireless Sensor Networks, (EWSN' 2005),* Istanbul, Turkey, 2005, pp. 166–175.

[100]. H. Karl and A. Willig, A short survey of wireless sensor networks, Technical TKN-03-018, Telecommunication Network Group, *Technische Universitat,* 2003.

[101]. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, Wireless sensor networks: a survey, *Computer Networks*, 38, 4, 2002, pp. 393–422.

[102]. Ali Iranli, Morteza Maleki, and Massoud Pedram. Energy efficient strategies for deployment of a two-level wireless sensor network, in *Proceedings of the International Symposium on Low Power Electronics and Design*, San Diego, CA, United States, 2005, pp. 233–238.

[103]. Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, Wei Hong, A Macroscope in the Redwoods, in *Proceedings of the 3rd international conference on Embedded networked sensor systems, (SenSys '05),* New York, NY, USA, 2005, pp. 51-63.

[104]. K. Akkaya and M. Younis, A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3, 3, 2005, pp. 325–349.

[105]. M. Younis, M. Youssef, and K. Arisha, Energy-aware routing in cluster-based sensor networks, in *Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, 2002, p. 129.

[106]. Nauman Israr and Irfan Awan, Multihop clustering algorithm for load balancing in wireless sensor networks, *International Journal of Simulation: Systems, Science and Technology,* 8, 3, 2007, pp. 13–25.

[107]. C. Chen, J. Ma, Ke Yu, Designing energy-efficient wireless sensor networks with mobile sinks, 2006, http://www.sensorplanet.org/wsw2006/1_Chen_WSW06_final.pdf

[108]. P. Gupta and P. R. Kumar, The capacity of wireless networks, *IEEE Transactions on Information Theory*, 46, 2, 2000, pp. 388–404.

[109]. Rahul C. Shah, Sumit Roy, Sushant Jain, and Waylon Brunette, Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks, *Ad Hoc Networks,* 1, 2-3, 2003, pp. 215–233.

[110]. Gokhan Mergen, Qing Zhao, and Lang Tong, Sensor networks with mobile access: Energy and capacity considerations, *IEEE Transactions on Communications*, 54, 11, 2006, pp. 2033–2044.

[111]. Yong Yao and Johannes Gehrke, The cougar approach to in-network query processing in sensor networks, *SIGMOD Rec.,* 31, 3, 2002, pp. 9–18.

[112]. S. Ramanathan and M. Steenstrup, A survey of routing techniques for mobile communications networks, *Journal of Special Topics in Mobile Networks and Applications (MONET),* 1, 2, 1996, pp. 89–104.

[113]. J. N. Al-Karaki and A. E. Kamal, Routing techniques in wireless sensor networks: a survey, *IEEE Wireless Communications*, 11, 6, 2004, pp. 6–28.

[114]. B. Krishnamachari, D. Estrin, and S. Wicker, Modeling data-centric routing in wireless sensor networks, in *Proceedings of IEEE InfoCom*, 2002.

[115]. R. Eskicioglu, S. Ahmed, S. Hussain, A query processing architecture for sensor networks, in *Proceedings of WICON Workshop on Information Fusion and Dissemination in Wireless Sensor Networks (SensorFusion),* Budapest, Hungary, 2005.

[116]. Yang Xiaoyan, Lim Hock Beng, Tamer M. Özsu, and Tan Kian Lee, In-network execution of monitoring queries in sensor networks, in *Proceedings of SIGMOD Conference*, 2007, pp. 521-532.

[117]. K. Ramamohanarao, L. Kulik, S. Selvadurai, B. Scholz, U. Roehm, E. Tanin, A. Viglas, A. Zomaya, and C. Leckie, A survey on data processing issues in wireless sensor networks for enterprise information infrastructure, May 2006, http://www.eii.edu.au/files/EIIAnnualRpt2006.pdf

[118]. S. Madden, M. J. Franklin, J. M. Hellerstein, and Hong Wei, Tag: a tiny aggregation service for ad hoc sensor networks, in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02),* Boston, MA, USA, 2002, pp. 131–46.

[119]. C. Intanagonwiwat, R. Govindan, and D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks, MobiCom 2000, in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking,* Boston, MA, USA, 2000, pp. 56–67.

[120]. Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann, Impact of network density on data aggregation in wireless sensor networks, in *Proceedings of International Conference on Distributed Computing Systems,* Vienna, Austria, 2002, pp. 457–458.

[121]. O. Younis and S. Fahmy, Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks, *IEEE Transactions on Mobile Computing*, 3, 4, 2004, pp. 366–379.

[122]. Guo Bin, Li Zhe, and Meng Yan, A dynamic-clustering reactive routing algorithm for wireless sensor networks, in *Proceedings of 1st International Conference on*

Communications and Networking in China, ChinaCom '06, Beijing, China, 2007, pp. 4149783.

[123]. A. M. Ali, Yao Kung, T. C. Collier, C. E. Taylor, D. T. Blumstein, and L. Girod, An empirical study of collaborative acoustic source localization, in *Proceedings of the 6th International Symposium on Information Processing in Sensor Networks*, Cambridge, MA, USA, 2007, pp. 41–50.