Senceive: A Middleware for a Wireless Sensor Network

Christian Hermann and Waltenegus Dargie
Chair for Computer Networks
Faculty of Computer Science
Technical University of Dresden
01062 Dresden, Germany
Email: s6928045@inf.tu-dresden.de, waltenegus.dargie@tu-dresden.de

***Abstract*— A significant amount of research effort is being carried out by the research community to increase the scope and usefulness of wireless sensor networks; to optimise life time by developing energy efficient power management, self-organising, medium access and routing protocols; and to reduce the cost of sensing nodes so that dense and robust deployment is possible. Though much has already been achieved, currently the cost of commercially available wireless sensor nodes is considerable and the wide applicability of proposed or existing protocols is still under investigation. One essential problem associated with cost or wide applicability of protocols is that sensor networks are application-specific. Protocols and in-network algorithms are optimised for particular sensing tasks. On the other hand, in research environments researchers would like to experiment not with a single application but with many applications. Considering the not-so-cheap sensing nodes available on the market and the management overhead of deploying wireless sensor networks, it is not economical or efficient to dedicate wireless sensor networks just to a single application, not at present at any rate. We therefore propose a middleware that enables researchers to experiment with multiple applications while providing them with essential in-network functionalities to satisfy individual application's requirements. The middleware cleanly separates sensing from network management so that application developers can obtain data from the wireless sensor networks without having to deal with management concerns.**

I. Introduction

Several applications have been proposed for wireless sensor networks. In some cases, field investigation and test have already been made and the experiences have been reported. These include habitat monitoring [13], active volcano sensing [18], structural health monitoring [2], and under ground mining [15]. These applications clearly demonstrate the greater scope and usefulness of wireless sensor networks which are made up of several spatially distributed, low-cost, lightweight, and smart sensor nodes with embedded software to fulfill complex tasks in a cooperating manner. The nodes gather data from their surroundings either continuously or periodically. The collected data will be processed locally or transmitted to a sink possibly in a multi-hop fashion. In the transmission path, some nodes act as relays, conveying locally gathered (and processed) data together with data from other nodes. At the base station, application software analyzes the collected data, produces a system log, invokes an alarm, or stores the data in a database for future use.

Compared to traditional computer networks, the design and implementation of a wireless sensor network requires confluence of many fields, including sampling technology, signal processing, networking, power management, embedded systems, information aggregation, and distributed computing, and many more.

At present, however, the challenges facing the deployment of wireless sensor networks contend with the opportunities. One typical constraint is that the sensor nodes are powered with limited capacity batteries, thereby limiting the lifetime of the network they constitute. In some applications, it is not feasible to recharge or replace sensors for many practical reasons. Subsequently, minimising the power consumption of a wireless sensor network is one of the main focuses of research. A plethora of protocols have already been proposed in all aspects of wireless sensor network. For example, for managing the power consumption of a sensor node, dynamic voltage and frequency scaling have been proposed [14]; for minimizing collusion, overhearing and idle listening, medium access protocols which accommodate dynamic or periodical sleeping have been proposed [17], [16]. There are other protocols and algorithms which aim to reduce the overall data traffic in the network by allowing in-network processing. Most of these protocols are, however, optimal to specific sensing tasks and may not be easily generalized. This is why it is argued by many that a sensor network should be deployed for a particular sensing task; this task should be known at a time of deployment and does not change or changes only slowly over time [10].

Similarly, research contributions have been made in developing middleware and data access and aggregation systems. There are two main motivations for these latter contributions. Firstly, at present the cost of commercially available sensor nodes or developing sensor nodes from scratch for particular applications is considerable; therefore setting aside an entire sensor network just for one particular application may not be feasible. Secondly, experimenting with multiple applications in parallel can be done only with the support of a middleware or a data access system which abstracts network management applications which are interested in the data that can be extracted from sensor networks. The focus of this paper is a sensor network middleware - the Senceive

Middleware - which we develop to experiment with multiple applications at our Chair, chair of Computer Networks, the Technical University of Dresden. Senceive separates sensing from network management (which is unavoidable at present) and enables application developers to define complex sensing tasks and in-network processing. We will discuss in this paper the conceptual architecture of our middleware as well as its implementation. The sensor network which the middleware abstracts is set up by several MICAz and Mica2 sensor nodes.

The rest of this paper is organised as follows: in section I we summarise related work; in section III, we discussed the requirements of the Senceive middleware; in section IV, we present the conceptual architecture of our middleware and discuss its implementation. In section V, we will give a report about the experiences learned in experimenting with commercially available sensors, namely, MICA2 and MICAz sensors. And finally, in section VI, we give concluding remarks and outlook to future work.

II. RELATED WORK

A. MoteWorks

Crossbow's MoteWorks [6] consists of three layers. The Mote Tier at sensor level supports self-organising networking to connect all nodes within range to the server by using the XMesh software [7] installed at the motes. The Server Tier runs XServe software [8] handling data translation and storage and providing interfaces for the client applications. At the Client Tier every application using XServe interfaces is able to gather and analyse data provided by the network. Crossbow provides MoteView [5] as a full-featured data analysing application and MoteConfig [4] as programming utility on the client side. Advantages of MoteWorks are the full support for the whole Crossbow sensor hardware product line, an optimized development environment and a sophisticated user interface, providing charts, health monitoring and data conversion to engineering units. MoteView for example provides temperature values in degree Celsius, Fahrenheit or Kelvin, acceleration in m/s2 or g units and gives status information about forward queues, dropped packets, retries, battery or path.

On the other hand MoteWorks also has some disadvantages. It sticks to TinyOS 1.x, which makes it hard to extend. It also restricts packet size to 55 byte whereas TinyOS would allow up to 235 byte. The software provides only a small command set to gather data in trivial way by simply defining a sensing interval. And even this is restricted to a minimum interval of 300ms [6] whereas several applications, especially for sound and acceleration need a higher sampling rate which actually is supported by the hardware. Finally it has to be stated that the provided framework is profound (manuals have about 500 pages), not application specific by nature and therefore hard to adopt to specific needs. All these disadvantages are probably the reason no research projects so far base on MoteWorks.

B. sdlib: Sensor Data Library

In contrast to the full featured approach provided by Crossbow sdlib [3] offers a component library for nesC developers. This library includes data collection and dissemination as integral parts of most sensor network systems. The components are implemented in nesC as well and thus can be integrated easily in existing nesC projects. Sdlib is not a middleware in the natural sense of hiding network implementation and offering an abstract user interface. But usually developers of wireless sensor network software which are new to the field of programming in nesC have to master specific concepts like asynchronous split-phase programming, sidestep race conditions or resource arbitration. The intention of the authors of sdlib is to support those developers with reliable, powerful and well tested components for these recurring common cases. Unfortunately sdlib is based on TinyOS 1.x and is no longer a research focus, because original researchers are now working on a successor called Declarative Sensor Networks (DSN). This shall address an increased number of common development tasks within wireless sensor networks. So far there is no release of this software available for further investigation.

C. TinyDB

TinyDB [12] is the pioneer in sensor network database abstraction developed at University of California Berkeley. Its sensor software implementation running on each node includes a schema manager to handle different types of readings and node properties, a query processor, a small memory manager and a topology manager for efficient routing. The Java-based client interface provides functionality to extract information about the network, build SQL-like queries, inject them into the network and listen for results. It also provides graphical user interfaces to construct queries, display sensor results and visualize network topology. The rich query language, with extensions for query duration and sample rates lets users describe the data they want to gather without requiring any knowledge about how this data might be gathered. Multiple queries are allowed and managed by a query execution planning engine. TinyDB also manages the underlying radio network and ensures relatively reliable data delivery. Finally low power optimization is also one of the advantageous aspects of TinyDB. On the other hand TinyDB is based on TinyOS 1.x, supporting no specific in-node processing capabilities.

D. Cougar

Cougar [19] is also a database approach quite similar to TinyDB developed at Cornell University. It also defines a high level SQL-like declarative query language. In such a query the FROM statement describes a node or a group of nodes called Abstract Data Types (ADTs) whereas the SELECT and WHERE statements refer to node specific data, invoking an abstract method which includes attributes for the node like input arguments, output values or timestamp. The query string is translated into a relational algebra expression. Then an optimizer combines all expressions from actual active queries and builds up a query execution plan. At the end a command injector distributes the commands into the sensor network according to the execution plan. Another issue considered in this project is about using adaptive query processing that adds

statistics-gathering to regular query processing and piggybacks small feedback data on results to long-running queries. The effort of this is to reduce administration overhead for gathering status information which is permanently necessary to address the general problem of lacking absolute knowledge about the global state of the network.

E. Mat

Mat [11] is developed at University of California, Berkeley. It is one of the first agent-based approaches and implements a virtual machine which interprets byte-code instructions. A capsule, which can be propagated quickly through the network, holds up to 24 instructions, each a single byte long. Larger programs can be built using multiple capsules. These programs are provided with three execution contexts which are associated to three events: clock timers, message receptions and message send requests. Each context has its own operand and subroutine address stack. This clear separation helps avoiding concurrency problems often occurring with standard TinyOS implementations. Mat is designed to run on MICA platform as well as on rene2, which offers even more restricted hardware resources with 1 KB RAM, 16 KB program memory and a data transmission rate of about 10 Kbit/s. Simple programs like a sense and send application with a binary size of about 5 KB would need a long time to propagate through the network. With Mat the same application can be expressed with 6 instructions and fits into one capsule of 24 Byte. It has to be stated that Mat is well designed for sensor networks with frequent changes of application. It also supports a reliable hardware abstraction which allows easy customization for relative simple sensing tasks. It provides the possibility for extending the instruction set with customized components. But the authors conclude as well that "'the interpretation overhead makes implementing complex applications entirely in Mat wasteful"' [11]. Mat also builds on TinyOS 1.x and is actually not enhanced for over three years now.

F. Agilla

As a pure agent-based middleware for wireless sensor networks Agilla [9] offers flexibility in dynamically adapt to changes in environmental conditions or multiple application requirements. The Agilla approach allows examining multiple phenomena without the need to initially specify at which location these might occur. Fields of application could be fire detection or parcel tracking. Every sensor node is able to run multiple agents and maintains a tuple space and a neighbour list. The tuple space contains locally available data which gives information about the node state and is the only possible way of communication among the agents on the node. There are also instructions which allow agents to remotely access tuple spaces of other nodes. The neighbour list contains the addresses of all nodes reachable within one hop. Agents may migrate quickly to other nodes, carrying their code and state, but leaving the tuple space at the node. The agent code itself is not written in nesC but in an own abstract stack operation based Assembler-like language. The definitive advantage of Agilla is its ability to dynamically adapt to changes in the network environment as well as to changes in application requirements. On the other hand Agilla is not applicable to complex sensing tasks. It is based on TinyOS 1.1, provides just primitive sensing operations, does not support streaming or long running sensing implicitly and has no timer abstraction. Extending Agilla is hardly possible because it already needs about 3.59 KB of the 4 KB of available RAM. Further complex data processing such as buffered high-frequency stream sampling is therefore infeasible.

III. SYSTEM REQUIREMENTS

The related work discusses so far focuses on either the query aspect or the management aspect of a sensor network. We build upon exiting work but explicitly support multiple applications through the employment of a lightweight query language and query processing engine; we also provide a highly flexible network configuration (management) interface to enable a network administrator define complex sensing tasks and orchestrate query requests so that the network does not over flooded by requests from independent applications which have no view of the other applications accessing one and the name sensor network.

More explicitly, Senceive satisfies the following requirements:

A. Sensor Data Quality

The implementation should provide sensor data quality as best as possible, supporting high frequent data sampling. A special focus in the hardware programming is set on recording and analysing the audio signal. This includes for example calculation of noise level, zero crossing rate or signal energy. Also tone-detection and range measurement should be regarded as point of interest. A further minor goal is to provide users with engineering units instead of raw data.

B. Specific Data Querying

The middleware should enable users to query each single sensor separately and gather data from connected sensors in the current environment by using a well defined interface. This interface should be locally and remotely accessible and include support for a SQL-like querying language as proposed by Bonnet et al. [1]. Such a language includes snapshot queries for immediate data requests, long run queries for periodical data sampling and historical queries for access to previously stored data.

C. Explicit and Automatic Network Configuration

A separate interface should provide an administrator to configure the whole network as well as individual nodes. Dynamic ad hoc recognition of new sensors and configuration of these should also be supported by the middleware. This includes dynamic information about available sensors on the node.

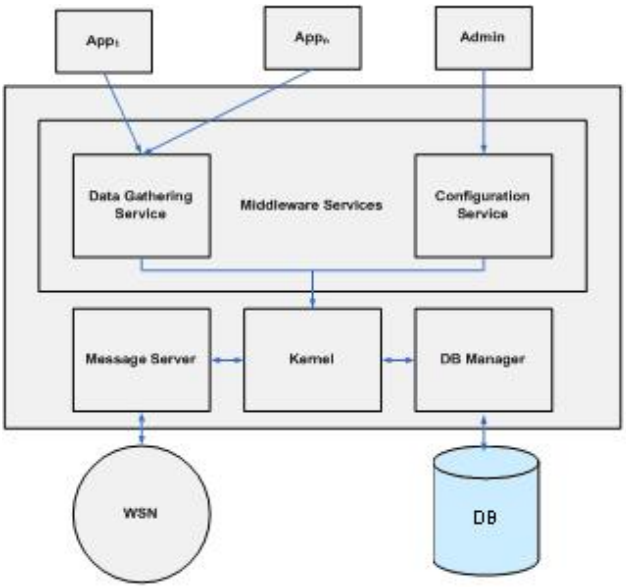

Fig. 1. The conceptual architecture of the Senceive Middleware

D. Status Information

The middleware should allow gaining meta information about the network infrastructure like routes or congestion information. Also specific node status information is of interest. This includes battery status, connection quality and sensor usage.

IV. ARCHITECTURE

This section gives an overview of the conceptual architecture of the middleware and describes the basic building blocks. The design decisions draw from several concepts and approaches. The middleware is designed as database system like TinyDB; the basic three tiered architecture is inspired by Crossbows MoteWorks; the implemented routing protocols are improved versions of the sdlib protocols; and the query processing is similar to the Cougar approach.

The whole system can be regarded as a three-tiered architecture as illustrated in figure 1. The mote tier (denoted as WSN) provides a communication interface for node control and is implicit part of the middleware. The server tier is illustrated in the middle in detail and acts as the central control instance of the network. The client tier finally holds one administrator for the network configuration and multiple applications for parallel data gathering.

The Message Server handles message sending, reception and pre-selection for further processing. The kernel contains business logic to interpret queries; handled incoming messages; perform periodic tasks; and persists gathered data using the DB Manager which encapsulates the database specific control methods. This clear separation allows easy replacement of the concrete database attached to the middleware.

While the Message Server uses the interface to the mote tier, the middleware services provide the interfaces to the client tier. There are two distinct services to separate data gathering and configuration. It is important to notice that users should not be allowed to configure nodes, as they can not be expected to have global knowledge about all users needs.

The Data Gathering Service provides the following functionalities:

- Send snapshot queries
- Send historical queries
- Start long run queries with or without data listener
- Provide information about available sensors in the network
- Register listener for changes in network status

The Configuration Service provides the following essential functionalities:

- Provides detailed network status information
- Provides information about configuration aspects
- Modifies individual mote configuration
- Modifies global configuration
- Manual mote adding and removal

The implementation of the mote software builds upon TinyOS 2.x including support for Crossbow MICAz with MTS300 or MTS310 sensor board. With version 2.0.1, TinyOS provides driver support for those boards including single sensor access, stream sampling, resource arbitration and power management. The motes autonomously register with an existing network, synchronize to global network time, receive commands to alter configuration or start sensing tasks and reliably deliver data to the sink.

Node deployment is carried out in two different ways: The first is supervised by the network administrator who configures the mote program according to the mote hardware, including a sensor board specific description and a globally unique id; additionally, the administrator should provide the middleware with information about the location of the newly deployed mote to support users with this important information. As a second way it should be possible to allow former network nodes or absolute new nodes (programmed with the same software) to enter the network and register dynamically. In this case the middleware configures the new node automatically, performing time synchronisation, and sends node information to the applications registered with a network status listener.

A. Mote-Tier

As there is no in network point-to-point communication necessary, routing gets simplified essentially. Establishing a routing tree is a good choice to match the applications need. All nodes reliably deliver data back over multiple hops to the sink which directly forwards the data via serial or Ethernet connection to the middleware server. We found the collection protocol introduced by sdlib as a good choice for this purpose. Command distribution on the other hand is intended to be realized by using the dissemination protocol also introduced by sdlib and provided by TinyOS 2.x. Senceive supports a Time-Diffusion time Synchronisation (TDS). With this technique the sink acts as precise time server and broadcasts the reference time to all master nodes. The master nodes then synchronise

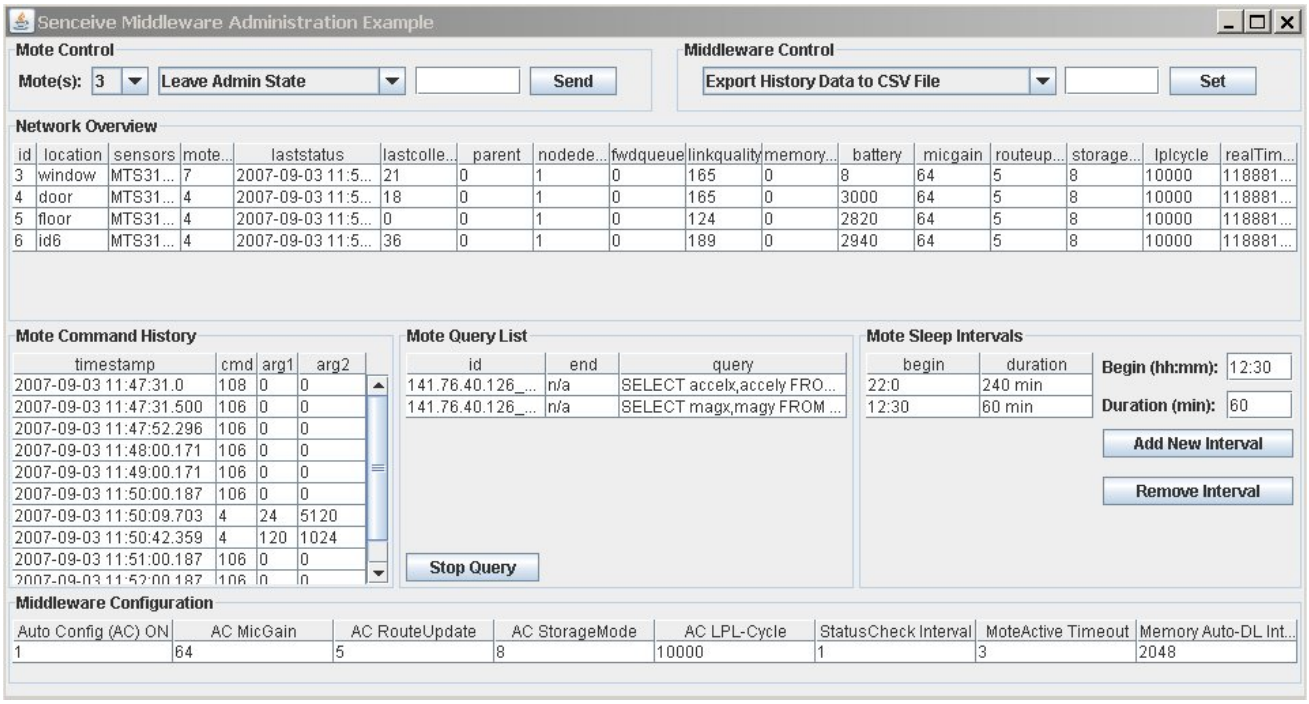

Fig. 2. An overview of the Senceive Middleware (admin interface)

their neighbours using the received reference time. Thus the established equilibrium time in the wireless sensor network is the reference time broadcasted by the sink. There is a drawback with this approach because master nodes are not explicitly defined to be always in range to the sink. As a result some might get not synchronized.

This is solved by periodically redefining which nodes are master nodes. Anyway TDS as well as other hierarchical synchronisation techniques always lead to decreasing accuracy with increasing node depth, according to unpredictable and indeterministic time used for local time transmission. But usually this gap is in the range of some milliseconds and thus acceptable for most applications. For instance, TinyOS running on MICAz provides a millisecond timer with an unsigned counter width of 32-bit. This offers nearly 50 days continuous running time without overflow. The approach works as follows. By starting the middleware and the base station, which is directly connected via serial or Ethernet connection to the middleware both synchronize to be able to provide received data with a UTC timestamp. When a mote is activated, it should be in a non-synchronized state and immediately tries to synchronize with a local mote which is synchronized. Message transfer delays should be considered to minimize deviation with increasing node depth.

B. Server Tier

Communication with the applications may involve remote access. At present Senceive supports Java Remote method Invocation. Application developers are provided with a reference implementation how to use RMI and how to access the server tiers services for data gathering and configuration. Query Processing is probably the most complicated part of the middleware. The kernel combines all queries for a mote and generates one resulting command satisfying all needs. If this is not possible due to hard- or software restrictions the user query leading to this problem will be rejected including an error message explaining the problem. This is realized by throwing an exception. The middleware also combines commands for several nodes to reduce message overhead by using multicast or broadcast command.

Data Storage is realized using a separate MySQL server running on the same host as the middleware. JDBC driver to access the data is provided for MySQL. All SQL request are encapsulated within one class to allow possible replacement. This database is also used to store configuration of the network and the middleware.

V. IMPLEMENTATION AND PERFORMANCE EVALUATION

Due to the enormous gap in hardware resources comparing the sensor nodes with a standard computer running the server side Java middleware program, it is useful to focus our evaluation on the behaviour of the nesC implementation. During development as well as in the test phase the Java program never faced any message or data processing problems, although query and data processing is no trivial processes. All measurements and performance tests related to the wireless network are done using a common hardware setup. This consists of several MICA-z motes with the network all of which are equipped with the MTS310 sensor board. The tests described in the following examine performance and

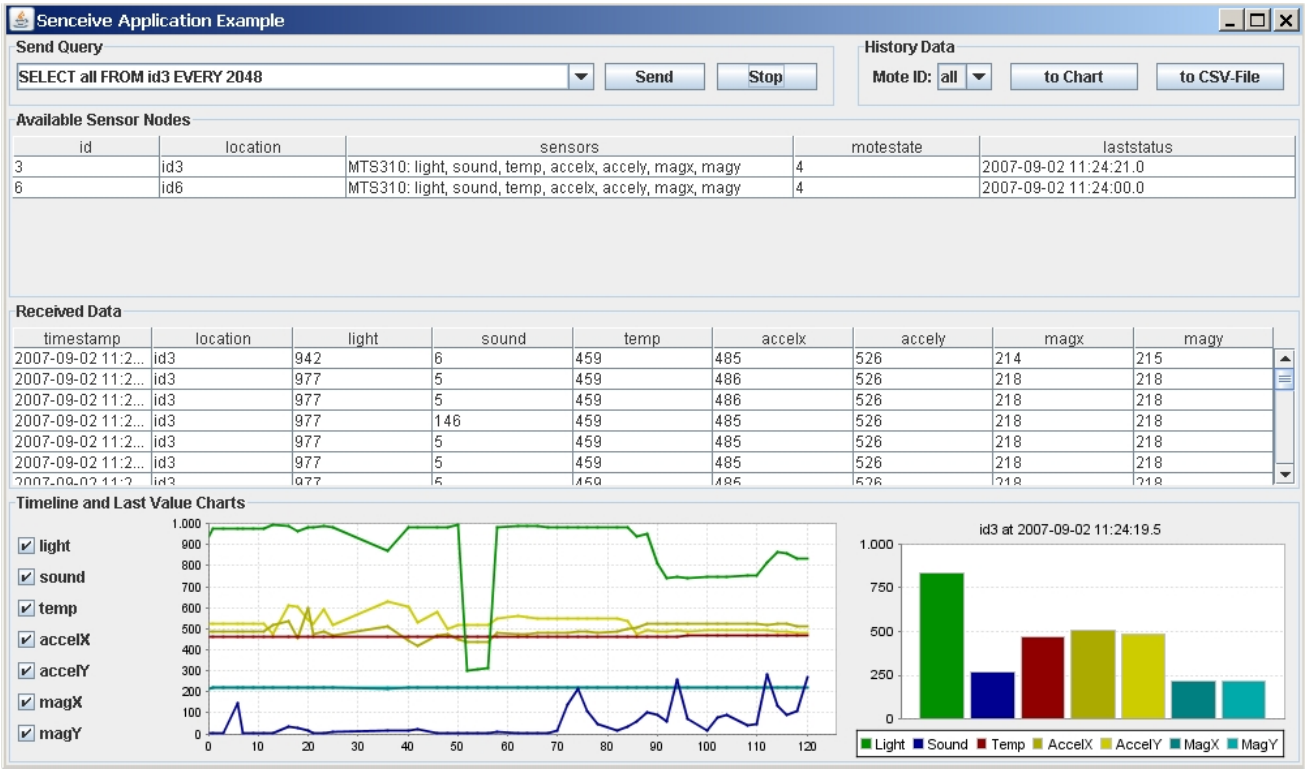

Fig. 3. An overview of the Senceive Middleware (application interface)

reliability of the routing protocols, sensor data quality and energy consumption.

The implementation uses the latest sensor board drivers provided by Crossbow. These drivers include resource arbitration, single sensor access and stream sampling support. As a summary, the middleware implementation offers several features listed below:

- Component based software architecture;
- Full support for Crossbow MPR2400 (MICAz) motes; full support for Crossbow MTS300 and MTS310 sensor boards; full support for Crossbow MIB510 and MIB600 interface boards;
- Multi-hop command dissemination and data collection;
- Network and node status information;
- Network wide time synchronisation; item Multiple sensor sampling up to 5Hz;
- Single or dual sensor stream sampling up to 4 kHz;
- In-node noise level calculation;
- SQL-like querying language;
- Snapshot, long-run and historical queries;
- Multiple query handling; and,
- Remote service access via Java RMI.

A. Data Collection

The implemented time synchronization described in the previous section allowed calculation of the collection time needed to send a packet from a node to the base station. To test the data collection performance of the system, several MICAz nodes were distributed in an apartment in different rooms in such as way that the node depth increases with every node. Whereas some nodes directly communicated with the base station, other nodes used intermediate nodes, based on the local decision regarding the signal strength. This ensured the establishment of a link with reliable quality. Test results were positive. No response was lost and collection time was in an acceptable and mostly expected range. The average collection time is about 10 ms for nodes within a one-hop range; 14ms for nodes within two-hop range; 20ms for nodes within three-hop range and 46ms for nodes within four-hop range. The data collection time fluctuated from 10ms to 25ms for nodes within one-hop range; 0ms to 24ms for nodes within two-hop range; 3ms to 45ms for nodes within three-hop range; and 33ms to 64ms for nodes within four-hop range. A further test starting multiple long run queries on the motes resulted in the same collection times.

The results led to the expected conclusion that collection time generally increases with increasing node depth. Only nodes within two-hop range showed unexpected behaviour, usually needing a shorter time to deliver message to the base station. A probable cause for this is the time synchronisation. As the implemented time synchronisation protocol does only provided synchrony with a resolution of around 20ms, real collection times could be higher. The above results can nevertheless be regarded as proof of reliability. All node messages reach the base station with a relatively short delay.

B. Command Dissemination

The test environment for the collection performance is also used to test command dissemination behaviour. Unfortunately the dissemination protocol does not offer any meta information about routing status as the collection protocol does. Thus it was hard to build up a meaningful test environment which can formally approve performance and reliability of the protocol.

Basically the dissemination protocol can be seen as bottleneck of our system. The most restricting characteristic of the protocol is its (purposely intended) behaviour of distributing only the latest command. This behaviour can lead to a loss of a command if a second command is sent directly after the first one or after a short time span. Especially in large scale networks the middleware would theoretically need to send several commands in parallel, for example if a new application starts a query which affects several nodes in different ways.

Tests show an expected behaviour. Motes with a node depth of 1 always receive the command with a negligible delay and answer promptly. With a higher node depth the answer time increases. During testing, nodes within one-hop always answers directly, while motes within two and three hops away usually answer with a 0 to 2 second delay and nodes within four hops away usually need 1 to 2 seconds to answer. In some cases the answer time is significantly higher reaching up to 4 seconds. The positive result of the test was that no command was lost.

The results showed that the dissemination protocol reliably delivered commands to the nodes as long as time span between sending two commands is large enough. On the other hand the protocol did not guarantee a low delivery time if multi-hop command delivery were necessary. Comparing worst cases led to a dramatic difference as command delivery time reached up to the factor of ca. 100 times slower than data collection time. It is left as a future work to develop an alternative command distribution protocol.

VI. DISCUSSION

This paper introduced the Senceive middleware for wireless sensor networks supporting multiple applications with data gathering services. The intended use within the academic environment makes Senceive general purpose sensor network software than other approaches. Possible applications range from long term usage like habitat monitoring to high frequent sampling tasks such as motion modelling. The middleware is characterized by applying multi hop routing techniques, energy aware resource management and globally synchronised time. More sophisticated design issues like security aspects have been left out as they would exceed the scope of this paper.

At present the middleware implementation fully supports hardware platforms of Crossbow MICAz with MTS300 and MTS310 sensor boards. These boards provide light, sound and temperature sensors, plus a 2-axis accelerometer and a 2-axis magnetometer on the MTS310. As basis for this implementation, the most popular operating system for wireless sensor networks was used: TinyOS. The component based programming language nesC is developed especially for the purposes of TinyOS and simplifies the software development by reusing components. But with the release of a fundamentally changed version 2.x in 2006 the TinyOS development community faces a tough challenge to port code from version 1.x. TinyOS 2.x still faces a reduced driver support and lacks broad contribution. It will need some time before this open source project is able do support commercial applications.

Most existing middleware approaches introduced in this paper are still based on TinyOS 1.x and therefore also struggle with code porting. Useful concepts and reference implementations like data collection and command dissemination protocols, a query language for specifying requested data, and processing principles for managing parallel queries are adopted from other wireless sensor network software like sdlib, TinyDB and Cougar. The middleware's component based software architecture and consequent model view controller ensures extensibility and enables further improvements. Multi-hop command dissemination and data collection, status information, automatic node recognition and configuration, high frequent sensor sampling and a public interface remotely accessible via RMI providing snapshot, long-run and historical queries through an SQL-like querying language and automatically managing parallel queries make the Senceive middleware a profound software that support complex sensing tasks.

At node level, unrealized features mainly concern data quality. Especially high frequency stream sampling and voice recording could not be realized. The evaluation identified some drawbacks posing questions which could be subject to further development. Two are essential to the performance and energy efficiency. The dissemination protocol used to propagate commands is broadcast oriented, whereas an optimized unicast approach would better fit to the needs. Further work should also try to enable the low power listening concept, which still lacks compatibility problems with the routing protocols. But as it is a simple and well scaling approach in theory to improve energy efficiency it is worth tracking future TinyOS releases, especially because developers recently announced its tight integration.

Taking an outlook to the future leads to the reasonable assumption that public areas will be covered by wireless sensor networks. Making them available to many applications rather than to a few by providing standardized interfaces through middleware like the one presented in this work

VII. CONCLUSION

We presented the Senceive middleware for supporting multiple applications which employ one and the same wireless sensor network to obtain data pertaining to a physical environment. The premise for the need to develop a middleware is that at present dedicating a sensor network for a single, specific application in research environments given the cost of establishing and maintaining presently affordable wireless sensor networks is not optimal. Senceive separates network management concerns from data aggregation and collection. This way, it is possible for applications to declaratively issue a sensing task without the need to have to directly deal with

management problems. Hence, the middleware consists of two essential components: a query interface and a network configuration interface. The query interface processes queries from multiple applications, and provides the applications with expressive and adequate SQL-like query language. At present, Senceive supports snap-shot and long-run queries. The configuration interface enables network administrators - it is hardly possible to avoid administrative tasks in real wireless sensor networks - to configure nodes and to prioritising sensing tasks.

Admittedly, such flexibility is attained at the expense of running the network at sub-optimal operation cost. For if the sensing task is known at the time the network is deployed, it is possible to configure the network to operate with highly efficient in-network algorithms and communication protocols. On the other hand, by defining elementary aggregation functions such as statistical mean, mode, median, standard deviation, etc., it is also possible to dynamically configure a network to carry our complex in-network processing tasks within the network. This is left as a future task.

REFERENCES

[1] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 7(5), 2000.

[2] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri. Monitoring civil structures with a wireless sensor network. *IEEE Internet Computing*, 10(2):26–34, 2006.

[3] D. Chu, K. Lin, A. Linares, G. Nguyen, and J. M. Hellerstein. Sdlib: a sensor network data and communications library for rapid and robust application development. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 432–440, New York, NY, USA, 2006. ACM.

[4] Crossbow Technology, Inc. *MoteConfig User's Manual*, 2007.

[5] Crossbow Technology, Inc. *MoteView Users Manual*, 2007.

[6] Crossbow Technology, Inc. *MoteWorks Getting Started Guide*, 2007.

[7] Crossbow Technology, Inc. *XMesh User's Manual*, 2007.

[8] Crossbow Technology, Inc. *XServe Users Manual*, 2007.

[9] C.-L. Fok, G.-C. Roman, and C. Lu. Mobile agent middleware for sensor networks: an application case study. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 51. IEEE Press, 2005.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, 2000.

[11] P. Levis and D. Culler. Mat: A tiny virtual machine for sensor networks. In *The 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002.

[12] S. Madden, J. Hellerstein, and W. Hong. Tinydb: In-network query processing in tinyos. Technical report.

[13] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, pages 88–97, 2002.

[14] R. Min, M. Bhardwaj, S. Cho, N. Ickes, E. Shih, A. Sinha, A. Wang, and A. P. Chandrakasan. Energy-centric enabling technologies for wireless sensor networks. *IEEE Communications Magazine*, pages 28–39, 2002.

[15] M. Ndoh and G. Delisle. Geolocation in underground mines using wireless sensor networks. In *Antennas and Propagation Society International Symposium*, pages 229–232, 2005.

[16] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Optimizing sensor networks in the energy-latency-density design space. *IEEE Transactions on Mobile Computing*, 1(1):70–80, 2002.

[17] Y. Wei, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, pages 1567–1576, 2002.

[18] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

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