Special issue on International Workshop on Networked Sensing System

Trans. of the Society of Instrument and Control Engineers
Vol.E-S-1, No.1, 58/67 (2006)

A Framework for a Distributed and Adaptive Query Processing Engine for Wireless Sensor Networks†

Supriyo Chatterjea *, Lodewijk van Hoesel * and Paul Havinga *

Wireless sensor networks (WSNs) are formed of tiny, highly energy-constrained sensor nodes that are equipped with wireless transceivers and can be used primarily in environmental monitoring applications. The nodes communicate with one another by autonomously creating ad-hoc multihop networks which are subsequently used to gather sensor data. WSNs also process the data within the network itself and only forward the result to the requesting node. This is referred to as in-network data aggregation and results in the substantial reduction of the amount of data that needs to be transmitted by any single node in the network. We present a framework for WSNs which would allow optimised query execution plans to be generated in a distributed manner using only locally available information within the network thus preventing the need to transmit network metadata all the way to a central server. We also describe how this framework can be used to perform cross-layer optimisations and illustrate this by presenting a MAC protocol that adapts its operation according to data gathered by the query processing engine. Query plans also adapt continuously by monitoring varying network conditions to maintain energy-efficient operation thus maximising network lifetime.

Key Words: wireless sensor networks, distributed, adaptive, query processing

1. Introduction

Wireless sensor networks (WSNs) are formed of tiny, extremely low-powered sensor nodes that are equipped with built-in wireless transceivers. **Fig. 1** shows a picture of a development board of an EYES [3] sensor node together with some of its specifications. It is currently used for testing purposes and the final version of an EYES node will be around the size of a one Euro coin. These nodes may be deployed in large numbers and are capable of communicating with one another by autonomously creating multihop ad-hoc networks which are subsequently used to gather sensor data. Considering the fact that these battery-powered nodes are supposed to operate for months (possibly even years) and that it is assumed that battery replacement is not a viable option due to the large numbers of nodes involved, one of the primary concerns of wireless sensor networks is how to extend the longevity of the network to the furthest possible extent by devising novel ways of managing energy reserves.

Our framework describes a distributed and adaptive query processing engine for wireless sensor networks that addresses two main problem areas - making use of locally available information and providing a platform that allows various system components of the WSN architecture to adapt to varying network conditions. In existing systems [1), 7)] network metadata is relayed back to a central node by all the nodes in a sensor network at regular intervals. This information is used to obtain a global view of the network which in turn helps to generate optimised query execution plans. The central node subsequently sends out instructions to specific nodes to perform certain tasks, e.g. data aggregation, routing, etc. Moreover, to generate query execution plans that are up-to-date with the current network dynamics, it is essential that the central node always has a fresh view of the net-

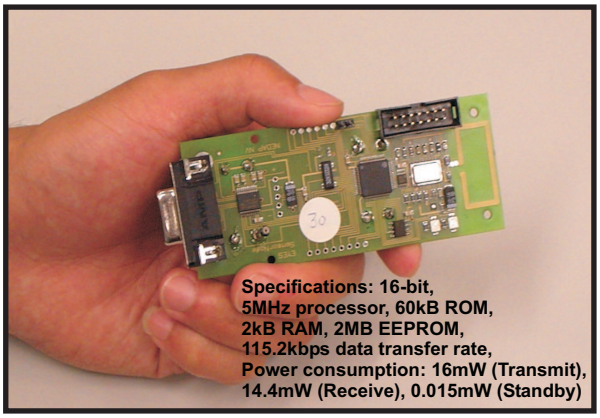

Fig. 1 A development board of an EYES sensor node.

† Presented at the First International Workshop on Networked Sensing Systems(2004.6)
(Received October 31, 2004)
(Revised April 28, 2005)

work. This mechanism has two fundamental problems. Firstly, the transmission of all network metadata by every node in the network to a central node at regular intervals is an extremely energy consuming task and secondly, in large networks, the collection of network metadata is a particularly high-latency process. Thus the generated query plan may be rendered useless due to backdated network metadata. In view of these problems, we suggest a framework where the generation of query execution plans is performed within the network based on locally available knowledge thus cutting back drastically on expensive transmissions to a central node. Consequently a node evaluates incoming queries and generates query execution plans based on the network metadata relayed by its neighbouring nodes.

Instead of "static" query execution plans where the central node instructs a certain node in the network to perform a specific task, an additional feature of our framework is that it allows query plans executing within the network to adapt to varying network conditions. These query plans are generated based on information that is captured locally. The information can also used be used by other system components such as the MAC, routing layer, operating system and query processor. These are explained in greater detail later.

In this paper we first give a brief overview of the related work in this area. We then present a framework for a distributed query processing engine for wireless sensor networks that will allow wireless sensor nodes to autonomously gather and analyse data on-site in an energy efficient manner. We also describe how the framework provides a platform which allows other system components to adapt their operation based on the locally-available information captured by the query processing engine. As an example of the benefits of such a system, we show how the operation of a MAC layer can benefit from being more "application aware" and present some preliminary results. We finally conclude the paper by stating the work that needs to be done in the future to build up on the framework presented.

Our work on WSNs is performed as part of the NWO funded CONSENSUS project [2] on self-organising and collaborative energy-efficient sensor networks. It addresses the convergence of distributed information processing, wireless communication and mobile computing.

2. Related work

The most energy consuming operation a node can perform is the transmission of data. In fact, transmitting just 1Kb of data a distance of 100 metres is approximately equal to the cost of executing three million CPU instructions [9]. Apart from simply collecting data, WSNs are designed to process data within the network itself and subsequently forward the result to the requesting node. This is referred to as in-network data aggregation and results in the substantial reduction in the amount of data that needs to be transmitted within the network as a whole which in turn translates into substantial energy saving. Data aggregation can be performed by intermediate nodes that lie between the sink (a node that injects a query into a network) and source (a node that responds to a query by sensing some physical parameter) node. These intermediate nodes carry out partial computation of the data obtained from sensor nodes thus ensuring that each node only has to transmit one data message. This also implies that bandwidth requirements between neighbouring nodes remain constant regardless of their position in the tree.

While performing data aggregation within the network may result in extending the operational lifetime of sensor nodes, it is important to note that there may be numerous ways to evaluate any particular query. Out of all these possibilities, only a handful might actually lead to energy savings. Thus it is important to develop a system that can analyse every incoming query and work out the optimal solution using the current network dynamics to ensure accurate decision making.

In certain existing query processing systems such as COUGAR [1] and TinyDB [7], network statistics (or network metadata) such as patterns of data produced by individual nodes, location information and energy reserves of nodes, etc. are sent back periodically to the central node (server) which originally injected a query into the network.

Using the centrally collected data, the server, which now has a detailed overview of the status of the entire network of nodes, calculates the optimal method in which the query may be evaluated. So the server generates a set of instructions that are then sent out to the individual nodes explaining the role individual nodes will play in evaluating the query, e.g. the server may stipulate which specific nodes would be required to perform aggregation of data for a particular query.

Another well known data-centric aggregation framework is Directed Diffusion [5] that is based on a publish/subscribe API. This however, does not use a database approach towards managing sensor data. Instead it uses a lower-level approach and does not hide the networking

specifics from the user thus making deployment of a system more difficult. DFuse [6] has a dynamic approach of placing fusion points but once again fails to provide the user with a SQL-like interface that is highly user-friendly. Also, DFuse is unable to handle data-centric queries and assumes that the specific addresses of nodes are known at the time of a query.

3. Design of the query processing engine

We have suggested a new framework for a completely distributed and adaptive query processing engine (QPE) for wireless sensor networks. The primary difference from the existing models is that in our framework, we transfer the task of generating query plans from a central node to the sensor nodes that lie within the network. Since nodes generate query plans using information that is available locally they greatly reduce the number to transmissions to the server. Every node would be able to detect changes in its vicinity and make necessary changes to the operation of its system components almost immediately. Also, the query evaluation procedure is carried out autonomously by the nodes and is completely transparent to the user. The user need not be concerned about the current network dynamics.

In this section, we highlight two of key features of our framework. First we describe its architecture which primarily deals with the individual building blocks of the QPE that help in the efficient processing of an incoming query. The second feature illustrates how the QPE provides the foundation that allows various system components to adapt. To demonstrate the benefits of our design, we describe how a MAC protocol adapts its operation using our framework as an example. Both these features would help to extend the network lifetime.

3.1 Architecture of framework

The QPE has a hybrid design involving a combination of centralised and distributed architectures. It consists of two sections - one residing on the main server (i.e. the central node) and the other residing on every node within the network. The section residing on the server carries out optimisations based solely on incoming queries. It does not require any information about the distribution of data within the network.

The second section of the QPE lies on top of the operating system in every node within the network. It is structured as shown in **Fig. 2**. The six components shown in Fig. 2 can be separated into two sections based on their functionality. The following subsections describe the role of each block in Fig. 2. The following is a brief description

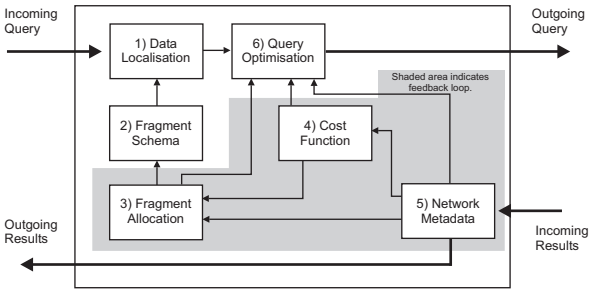

Fig. 2 Framework of the distributed query processing engine.

of each block:

• Data localisation - The main role of the data localisation block is to localise the query's data using data distribution information which is obtained from the Fragment Schema block. Thus data localisation examines the incoming query from the query decomposition block and determines which fragments of data are involved in the query. It also retransforms incoming queries into simpler and more optimised forms by using different reduction techniques depending on how the data has been fragmented. For example, when selections on fragments are made that have a qualification contradicting the qualification of a fragmentation rule, empty relations (or redundancies) are generated. Reduction rules ensure that such empty relations are eliminated.

Suppose we have a relation *sensors* containing readings of light and temperature of different floors in a building which is split into three horizontal fragments $sensors_1$, $sensors_2$, $sensors_3$ and is defined as follows:

(1) $sensors_1 = \sigma_{Floor \leq "4"}\ (sensors)$

(2) $sensors_2 = \sigma_{"4" < Floor \leq "8"}\ (sensors)$

(3) $sensors_3 = \sigma_{"8" < Floor \leq "12"}\ (sensors)$

Thus the localisation program for such a horizontally fragmented relation would be as follows:

(4) $sensors = sensors_1 \cup sensors_2 \cup sensors_3$

Such horizontal fragmentation can help to simplify certain operations as it is possible to produce empty relations and subsequently eliminate them. This can be illustrated in the case of a selection operation as shown in the following SQL statement:

```
SELECT nodeid, temp, light
FROM sensors WHERE Floor= 7
SAMPLE PERIOD 1s for 60s
```

The statement above gathers temperature and light readings from the 7th floor every second for a whole minute. The naïve way to process this query would be to replace every instance of sensors with ($sensors = sensors_1 \cup sensors_2 \cup sensors_3$). It is evident that

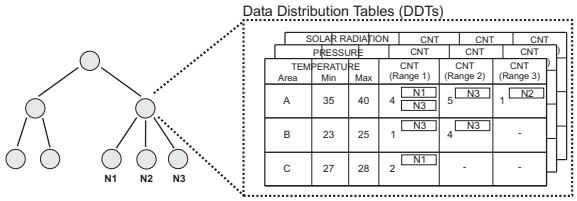

Fig. 3 Format of the Data Distribution Table (DDT).

$sensors_1$ and $sensors_3$ would then produce empty relations which are undesirable. Thus the reduced query is only applied to $sensors_2$. Such query simplifications can subsequently be used to limit the broadcast of a query to a particular section of the network thus resulting in energy consumption due to the reduction of transmissions.

• Fragment schema - The Fragment Schema describes how it is possible to reconstruct a relationship from the various fragments. It extracts the information from the Fragment Allocation Block. In other words it may describe how to apply a set of UNION operations to recreate a relation from the fragments.

• Fragment allocation - The task of the Fragment Allocation block is to decide at which node a certain fragment of a relation should be stored. Suppose there are a set of fragments F = F_1, F_2,.., F_N and a cluster of sensor nodes, S = S_1, S_2,.., S_N, the Fragment Allocator needs to find the optimal distribution of F to S. There are numerous parameters that need to be considered during the optimisation process, e.g. cost of communication between any two nodes, S_i and S_j, varying access patterns of various nodes, mobility patterns, cost of storing each F_i at a site S_j, cost of querying F_i at a site S_j, remaining energy reserves of a node and memory and performance parameters such as throughput and response time.

• Cost function - The Cost Function block stores the cost of some static parameters such as cost of polling a sensor and are generally defined prior to deployment of the sensor network.

• Network metadata - This block gathers network statistics based on every single query or result that the node hears, i.e. the query or result may not be addressed to it specifically but it might overhear a certain message from a neighbouring node that is within its transmission range, e.g. sizes of relations, patterns of query flow, energy and memory reserves of a particular node. One of the main components of the Network metadata block is the Data Distribution Table (DDT), **Fig. 3** . The DDT is responsible for gathering information about the data that flows through a particular node. The information collected and stored in the DDT is used by other system components to adapt their operation according to the network dynamics.

Overview - Once a node receives a query it looks up its Data Distribution Tables (DDTs) to deduce how many of its children are going to respond to the particular query. Every node maintains its own set of DDTs each of which is a table representing a particular type of sensor that is present within its set of child nodes. So if for instance a node and its children possess temperature and air pressure sensors, then the node would have two separate DDTs. A DDT is built over time by monitoring the data that flows through it. In other words, statistics about the data flowing through the network is collected without incurring any extra overhead. It is simply based on the data that already needs to flow from the leaf nodes to the root. Thus statistics are collected using only locally available information.

Upon receiving a reading from a child node, the parent node updates the entry in the appropriate DDT depending on a number of variables: the type of sensor that originally generated the reading, the region where the reading originated from and the value of the reading itself. Additionally, the DDT also keeps track of the maximum and minimum readings obtained and also which particular immediate neighbour sent it the reading. As shown in Fig. 3, apart from the first three columns of a DDT, the remaining columns contain the number of readings received that fall within a particular range.

Active children It is important to highlight that the numbers or rather the "count" found in the DDT is not simply a count of the total number of children a node has. It only includes the number of active children. By active children, we are solely referring to the number of nodes that are actually involved in servicing a particular query. Knowledge of the number of active children for a certain query would help in making rough estimates on how much data can be expected and this information can then be used to adapt the various components of the WSN architecture accordingly. For example nodes in areas which are generating larger amounts of data can be made to work more aggressively.

Directed dissemination using multiple DDTs and ranges Using multiple tables and categorising the incoming data into ranges has several advantages. Instead of flooding the entire network with queries, it would be possible to have a directed dissemination of queries.

Thus the dissemination of queries could only be limited to certain sections of the network. However, instead of disseminating queries only depending on region, the DDT allows more complex query dissemination schemes to be derived from queries that are injected into the network by the end-user. These complex schemes could span across multiple DDTs.

For example, the user could inject a query that requests for solar radiation readings in a particular region only when the temperature reading for the same region is above a certain threshold and the air pressure readings are within a particular range. This would involve parsing both the temperature and air pressure DDTs. The DDT also allows query optimisation to be performed on an incoming query. Suppose a query asks for all temperature readings above say 35C. If the DDT indicates that all readings above 35C only appear from a certain region, then the query dissemination could be limited only to that region even though it was not specified by the user. Thus in both examples, it is obvious that the number of nodes that would be involved in the query dissemination process could be greatly reduced.

Using only local information As can be seen from Fig. 3, every DDT includes a list of the immediate neighbours that have contributed to readings that fall within a certain range. This is used once again during query dissemination to forward incoming queries to the right node, i.e. the node that has been relaying data within the range that the query is interested in. Since the information contained in every DDT of every node is purely based on the immediate neighbourhood, the size of the DDT is not dependent on the depth of the node within the tree. In other words the size of the DDT does not increase depending on the number of active children a node possesses. The size of the DDT however, is largely dependent on the number of different types of sensors present in the child nodes, the number of regions defined within the network and the number of ranges the sensor readings are classified into.

Keeping DDTs up-to-date All All DDTs would have an expiry time attached to them since the range information will not remain accurate through out the entire lifetime of the network. Thus it is important to occasionally re-flood the entire network so that the latest ranges of readings are reflected in the DDT. The frequency at which the DDT needs to be refreshed depends on the rate of variability of the physical phenomenon being measured, and this will be defined by environmentalists during network deployment.

Adapting to varying network and query dynamics The dynamics within a WSN can be classified into two sections: network dynamics and query dynamics. By network dynamics we refer to the ability to continue operation when certain nodes die out and when nodes with new sensors are attached to the network. As mentioned earlier, we do not consider mobile nodes at this stage. If a node dies out, or is removed, this change is reflected in the corresponding DDT after the predefined expiry time has expired. Also, if a node with a new sensor is added to the network, the immediate parent node and all the related ancestral nodes automatically create a new DDT to reflect this new addition.

Using the information collected in the DDT, components of the WSN can also change their operation to adapt to varying query dynamics. Query dynamics refer to the situation when the number of queries passing through a particular node at any point of time varies. We mentioned earlier that we expect multiple queries to be injected into the network distributed both spatially and temporally. Thus various system components, such as the MAC for instance, can adapt its operation based on a not just a single query but based on the net requirements of all queries being served at a certain point of time.

Query Optimisation The Query Optimisation block receives several execution strategies (or operator trees) for a single query from the Data Localisation block. It is within this block that the QPE actually takes into account the distribution of data fragments, various costs involved and the current network dynamics in order to generate a query execution plan.

3.2 Adapting to Network Dynamics

The framework we have presented up to now provides a foundation that will allow the various components of the WSN architecture to adapt to the varying network dynamics. We now briefly describe how certain components can make use of the QPE to adapt their operation accordingly.

Medium Access Control Protocol (MAC) Instead of having a static duty cycle through out the network, the MAC can take advantage of the knowledge gained about the queries injected into the network by the application and dedicate more bandwidth to the relevant parts of the network. Thus while the MAC may work aggressively in certain areas, in other areas it will remain relatively dormant. This will ensure that bandwidth is allocated fairly and thus also prevent energy from being wasted in areas where data need not be collected.

We give a more detailed analysis of how the MAC can benefit in the following section.

Routing Queries are disseminated only to areas where they are more likely to return readings which meet the criteria set by queries, e.g. why send queries to sector A of the network when the user is only interested in sectors B and C?

Operating system Scheduling of tasks to be executed by the CPU can be prioritised based on the data traffic rates that have been predicted by the QPE. This will make the system more responsive to changes within the network.

Query optimisation Queries can be processed more efficiently if a node has knowledge of the distribution of data in the network. Consider a nested query as an example which states that Obtain the reading of Parameter A, only if Parameter B and Parameter C meet the conditions X and Y respectively. If Parameter B has a high selectivity, i.e. small number of active child nodes, and Parameter C has a low selectivity, i.e. large number of active child nodes, it would obviously be a lot more energy efficient to check Parameter B before Parameter C.

4. AI-LMAC: An example illustrating how a component can adapt using the QPE

The Adaptive and Information-aware, Lightweight Medium Access Protocol (AI-LMAC) is a TDMA-based protocol that is an adaptive and information-aware version of the LMAC protocol [4].

Time is divided into *time slots*, which nodes can use to transfer data without having to contend for the medium or having to deal with energy wasting collisions during transmissions. A time slot consists of two parts: the Control Message (CM) which is always transmitted by a node in the time slot(s) it controls and the Data Message (DM), which contains the higher protocol layer data. The CM has a fixed length and contains control information. The DM can have a length up to the end of the time slot or can even be omitted when the node has no data to send. For energy-efficiency reasons, nodes will turn-off their energy consuming transceiver when they are not the intended receivers of a DM, or when the transmission of a DM has finished before the end of the time slot. To be able to maintain the protocol, nodes will always listen to CMs of their neighbouring nodes.

To limit the number of time slots necessary in the network, we allow time slots to be reused at a non-interfering distance. Unlike traditional TDMA-based systems, the time slots in AI-LMAC are not divided among nodes by a central manager. Instead, the nodes use an algorithm that is based on local information only to choose time slots to control. Every node transmits a table in the CM that specifies which time slots the node considers to be occupied by itself and its one-hop neighbour nodes. This information can be efficiently encoded by a number of bits equal to the number of time slots in a frame. A node can occupy the appropriate slots when the required number of slots is considered to be free by all its neighbours. This method ensures that a time slot is only reused after at least three hops and that no collisions will occur. In fact, most MAC protocols ensure a similar distance between simultaneous transmissions. For example, in the SMAC protocol, this distance is assured by the exchange of RTS- and CTS-messages.

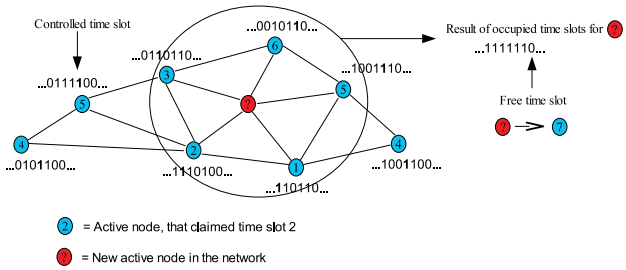

Fig. 4 A new active node in the network can pick a time slot when it has discovered all its neighbour nodes.

Fig. 4 gives an example of how a new node in the network can pick a time slot after it has discovered all its neighbours. When a node picks a time slot to control, it will control the same time slot in consecutive frames. Currently, we are considering frames of 32 time slots. Note that nodes will only use their own time slots to transmit data to their neighbouring nodes. In the AI-LMAC protocol, nodes are allowed to control multiple time slots in a frame. To ensure a connected network, every node controls a minimum of one time slot.

- Control message of AI-LMAC - The Control Message has a fixed size and is used for several purposes. It carries the ID of the time slot controller, indicates the distance of the node to the gateway in hops for simple routing to a gateway in the network, addresses the intended receiver(s) of the Data Message, reports the length of the DM and it carries acknowledgements to successfully received messages. The control data will also be used to maintain synchronisation between the nodes and therefore the nodes also transmit the sequence number of their time slot in the frame. The

transmission of the control data is carefully timed by the nodes, although we do not assume that the nodes have clocks with high accuracy. We assume that the clock drift is negligible in a single frame, even for clocks with low accuracy. All neighbouring nodes will ensure they receive the control messages of their neighbouring nodes. When a node is not addressed in that message or the message is not addressed as a broadcast message, the nodes will switch off their power consuming transceivers only to wake up at the next time slot.

◦ Network setup - When nodes are powered on, they are all unsynchronised. Also nodes are unaware of the number of slots they need to control. In order to get synchronised, the gateway takes the initiative to start controlling a time slot. The control messages of the gateway are received by its one-hop neighbours. These neighbours then synchronise their clocks to the gateway. After one frame, the one-hop neighbours are aware of all time slots that are owned by possible multiple gateways in their reception range. Next, the recently synchronised nodes will pick a random time slot. Slots already occupied will be excluded from this random slot selection process.

Once the nodes have been synchronised, the next step is to assign the right number of slots to a node depending on the expected traffic of a particular query as described in the following section. For details of the support for routing to the gateway node, we refer the reader to [4]. Note that the protocol also supports ID-based routing techniques, without adaptation of the protocol.

4.1 Adapting AI-LMAC using the data management framework

We now describe how AI-LMAC can adapt its operation using the information provided in the DDT. Unlike LMAC, which allows every node within the network to own only one slot [4], AI-LMAC allows a node to own multiple slots. Also, AI-LMAC is able to vary the number of slots a particular node owns depending on the amount of data that is expected to flow through it. This ensures fairness in the sense that the bandwidth allocated to a node corresponds to the traffic it is expected to encounter. For example, it would be pointless to allocate a large number of slots to a particular node that is not generating or relaying significant amounts of data.

In AI-LMAC, we assume that a parent-child relationship exists between all the nodes in the network, such that the root of the network can be considered to be the highest parent in the hierarchy. Using the DDT, every node would know how much "importance" to give every one of its immediate children.

Using the DDTs a node cannot decide by itself, how much importance it should give itself to transmit. This is because the DDTs only contain information about a node's active child information. A node is not aware of the data generated by the other children of its own parent node as they may not be in range. Thus, the parent is the only node that has knowledge of the proportion of data that will be contributed by each of its immediate children. The idea here is that if a node realises that a subset of its immediate children is going to transmit large quantities of data, then more attention Figure 4. A new active node in the network can pick a time slot when it has discovered all its neighbour nodes needs to be paid to this particular subset of child nodes. In this case, when we say more attention, we actually refer to assigning multiple slots to a particular child.

However, even though a parent node knows which child node deserves more slots to be assigned to it, it cannot send such a rigid instruction to its children as in LMAC. This is because in LMAC, when a node performs slot assignment, it has knowledge of the slot ownership of its first and second order nodes. In this case, the parent node would not know slot ownership information about the slot assignments of its child node's second order nodes since they are three hops away. Thus, the responsibility of the parent node is simply to "advise" the child, i.e. the parent node sends a message to every one of its children indicating the ideal number of slots that a particular node should take up under the current conditions. It is then up to the child node to follow the advice as closely as possible. This naturally depends on the number of empty slots available.

The process of giving advice starts at the root node of the tree when a query is first injected into the network. This process then percolates down the branches of the tree towards the leaf nodes. If however, the process of giving advice started at an intermediate node, this would increase the chance of performing unfair slot allocations. This is because a node assigning slots would not be aware of the bandwidth requirements of all its sibling nodes which are not within its direct range. From this argument, it is obvious that if we apply this rule repeatedly, the root

node is the only node which can assign slots fairly at the beginning. We term this as *horizontal fairness* as the mechanism ensures that all sibling nodes (i.e. at the same level) under a certain parent are allocated slots fairly.

Apart from establishing a horizontal relationship between nodes, we also introduce a mechanism to include *vertical fairness*. In order to prevent buffer overflow problems, our mechanism ensures that that the total number of slots assigned to the immediate children of a certain parent node, does not exceed the number of slots owned by the parent. This reduces the likelihood of data packets being dropped due to lack of bandwidth. Furthermore, leaf nodes are prevented from being allocated excessive bandwidth using this mechanism.

Thus introducing *two-dimensional fairness* ensures that the number of slots taken up by a node does not only depend on its siblings but on its parent as well.

Once a node has received the ideal number of slots it should take up, it checks to see which slots are free within its 2nd order neighbourhood. Just like in LMAC, once a node decides to take up a certain slot, it "marks" the slot using a "1" to indicate that the slot has been taken up.

In LMAC, a node transmits its control message (CM) section at the beginning of every slot it owns. Therefore every occupied slot in the frame has a CM section. However, in the case of AI-LMAC where a node may own multiple slots, the CM section is only transmitted during the very first slot it owns. Subsequent slots owned by the node only contain the Data section. This reduces the overhead incurred. In addition to the information contained in the CM section stated in [4], it also contains a list of all the slots that it owns. Suppose a node B receives a CM section from node A, and node B realises that it is supposed to receive the data that node A is going to send, node B makes sure that it is in receive mode during every slot that is specified in the CM section of node A. Other nodes which are not addressed in node A's CM section, only restrict their listening to the initial section of every slot in a frame which is usually reserved for the CM section.

4. 2 Experimental Analysis

Our framework provides a mechanism to assign more bandwidth to those parts in the network that encounter more data traffic than others. In fact, the assigned bandwidth is proportional to the expected traffic. Hence our framework is able to minimise the overall latency in the network and also the number of messages which need to be buffered in the nodes. **Fig. 5** presents these measurements for the single slot scenario. These results are obtained by simulation using the discrete event simulator OMNeT++ [8]. Results are averaged over five different network topologies consisting of 49 nodes and one root. Ten different runs were carried out per topology.

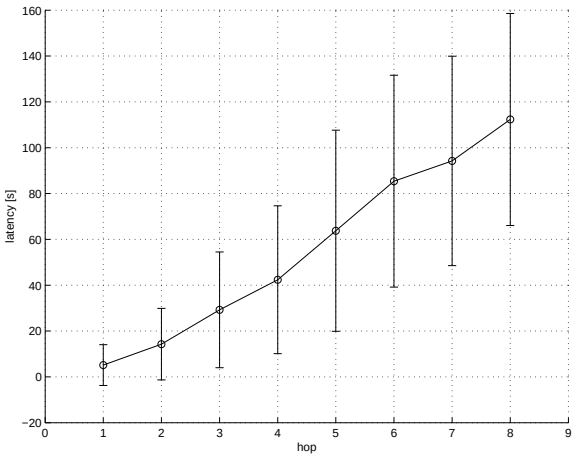

Fig. 5 Average latency and standard deviation when all nodes control one time slot.

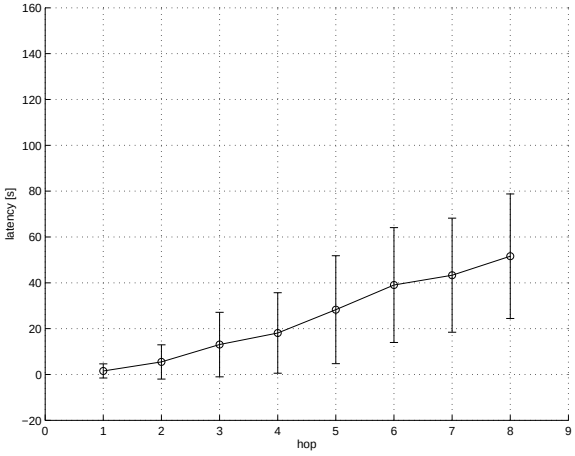

Fig. 6 Average latency and standard deviation; maximum given advice: 2 slots.

In **Figs. 6**–**10**, the advice for the maximum number of allowable time slots is varied from 2 to 16. The results clearly indicate that latency is proportionally reduced with the maximum number of controlled slots. However, this holds true only until eight slots, Fig. 8. For the twelve and sixteen slot scenarios, the number of free slots in the network rapidly decreases with every hop from the root and thus the nodes are not able to comply with the advice. Consequently, a bottleneck is created at a few hops (6 to 8) from the root, resulting in higher latency for messages created

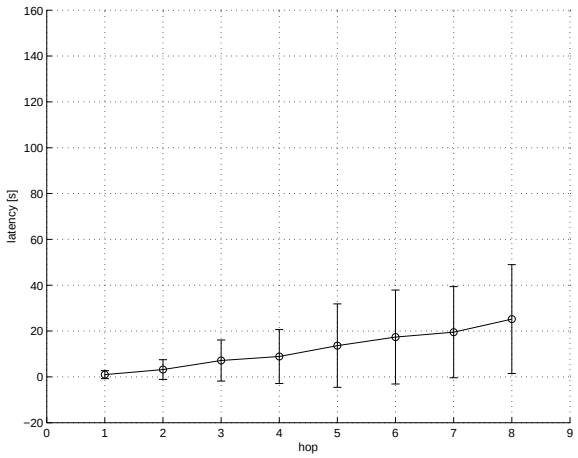

Fig. 7 Average latency and standard deviation; maximum given advice: 4 slots.

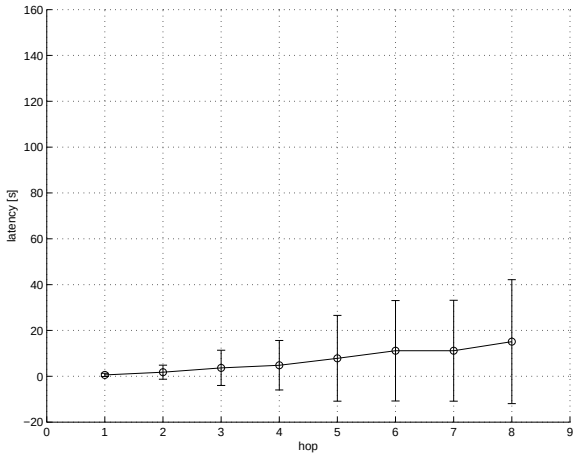

Fig. 8 Average latency and standard deviation; maximum given advice: 8 slots.

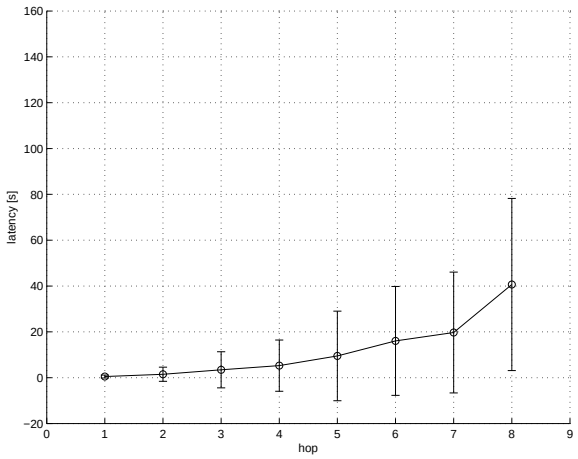

Fig. 9 Average latency and standard deviation; maximum given advice: 12 slots.

in those areas.

In this MAC protocol, nodes are able to receive up to 32 (=number of time slots) messages per frame. The number of messages that can be transmitted per frame is dependent on the number of time slots the

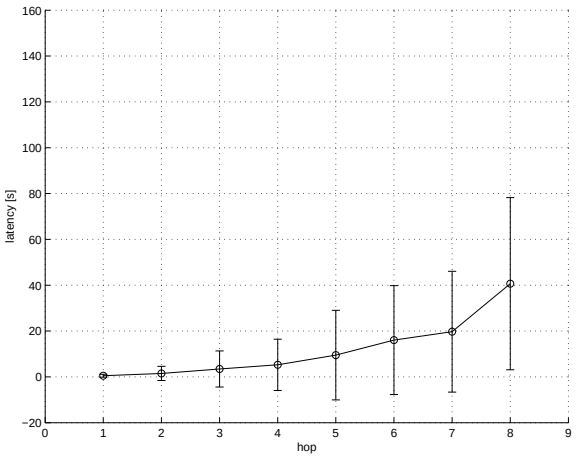

Fig. 10 Average latency and standard deviation; maximum given advice: 16 slots.

Table 1 Maximum number of back logged messages (worst case)

Scenario	Maximum messages
1 slot	105
2 slots	63
4 slots	34
8 slots	32
12 slots	54
16 slots	56

node controls. When the number of incoming messages exceeds the number of messages that can be transmitted during a frame, the additional incoming messages have to be buffered. For each of the scenarios, we have collected data about the maximum number of messages back logged in the worst case (Table 1). These results reflex the same trend as Figs. 5–10. Note that in real-life implementations, the capacity to hold back logged messages will be limited due to scarce memory resources in the sensor nodes.

5. Conclusion and future work

In this paper, we have described a framework for a distributed and adaptive query processing engine for wireless sensor networks where we concentrate on two primary problems (i) reducing transmissions by generating query optimisation plans based on local information, (ii) introducing a platform which allows other system components to adapt their operation according to network dynamics. We have also presented preliminary results showing how the performance of a MAC layer can benefit greatly by being more application-aware. Future work shall concentrate on extending the DDT to include information about data generation rates and degree of data aggregation. We shall also be looking into the memory

requirements of the framework.

References

1) P. Bonnet, J. Gehrke and P. Seshadri: Towards sensor database systems, 2nd International Conference on Mobile Data Management (2001)
2) CONSENSUS homepage: http://www.ctit.utwente.nl/research/projects/national/NWO/consensus.doc
3) EYES homepage: http://eyes.eu.org (1991)
4) L.F.W. van Hoesel and P.J.M. Havinga, "A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing Preamble Transmissions and Transceiver State Switches", In 1st International Workshop on Networked Sensing Systems, Tokyo, 2004
5) C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, pages 56-67, Boston, MA, USA, Aug. 2000. ACM
6) R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul and U. Ramachandran. DFuse: A Framework for Distributed Data Fusion. In 1st International Conference on Embedded Networked Sensor Systems, Los Angeles, CA, USA, Oct. 2000. ACM
7) S. Madden, R. Szewczyk, M. J. Franklin and D. Culler. Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks. In 4th IEEE Workshop on Mobile Computing Systems and Applications, Callicon, NY, June 2002
8) OMNeT++ discrete event simulator, http://www.omnetpp.org
9) G. Pottie and W. Kaiser, Wireless integrated network sensors. In Communications of the ACM, 2000

Supriyo Chatterjea

He is currently a full-time PhD student at the Department of Mathematics, Electrical Engineering and Computer Science at the University of Twente. After graduating from Nanyang Technological University, Singapore, with a Bachelor's degree in Electrical Engineering in 2001, he obtained a Master's degree with distinction in Computing and Internet Systems from King's College London, UK, in 2002. He was also awarded the "Best MSc Project" award at KCL for his work on service discovery in mobile ad-hoc networks carried out at IBM Research Laboratory, Zurich, Switzerland. His research interests lie in the field of distributed data management for wireless sensor networks.

Lodewijk vanhoesel

He holds a Masters degree in electrical engineering from the University of Twente, the Netherlands. In the recent past he has developed simulation models for software-defined Hiper- LAN/2 radio and has been working on a bone densitometer in a large company in the United States. He is currently working within the European project EYES on energy-efficient wireless communication mechanisms. Research topics include the physical layer, medium access protocols, wakeup radio, and signal processing.

Paul Havinga

He received his Ph.D. in mobile multimedia systems in 2000, and was awarded with the DOW Dissertation Energy Award for this work. Currently, he is within the Department of Computer Science of the University of Twente. His research interests are in the area of energy-efficient architectures and protocols, sensor networks, wireless communication, ubiquitous computing, personal communication systems, and (reconfigurable) hardware architectures. Currently, he is project leader of the Dutch project Smart Surroundings, on ambient intelligence, the Dutch project Featherlight on distributed operating system software, the European project EYES on energy efficient sensor networks, and the nationally funded project CONSENSUS on collaborative sensor networks. Besides these projects, he is also involved in several other projects, with a strong focus on sensor networks.

Reprinted from Trans. of the SICE
Vol. E-S-1 No. 1 58/67 2006