View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Portsmouth University Research Portal (Pure)

Efficient Time Triggered Query Processing in Wireless Sensor Networks

Bernhard Scholz[1], Mohamed Medhat Gaber[2]
Tim Dawborn[1], Raymes Khoury[1], and Edmund Tse[1]

[1] The University of Sydney
Sydney, NSW, Australia
[2] CSIRO
Hobart, TAS, Australia

Abstract. In this paper we introduce a novel system that comprises techniques advancing the query processing in wireless sensor networks. Our system facilitates time triggered queries that are scheduled in a distributed fashion among sensor nodes. Thus, time synchronisation is of paramount importance. Since accurate time synchronisation requires more energy, our system allows a trade off between precision of time and energy according to the user requirements. To minimize the communication overhead for query processing, our system employs new query execution mechanisms.

We have implemented our query processing system on Sun™ Small Programmable Object Technology (SPOT) sensor network platform. The system was entirely programmed in Java enabling an object oriented design and implementation. It provides a friendly graphical user interface for query management and visualisation of results.

Keywords: Wireless communications and ad hoc networks, distributed query processing, communication and energy optimisations, time triggered protocols.

1 Introduction

With the advent of smart sensor devices, wireless sensor networks are an emerging research field [1,2]. Wireless sensor nodes form a wireless ad-hoc network with a large number of nodes which operate without direct human interaction. The applications of wireless sensor networks are diverse and include environment and habitat monitoring [3], traffic control [4], health monitoring [5], supply-chain management [6], security and surveillance systems, and smart homes.

In this work we are concerned with distributed query processing [1,7] in sensor networks. Users are typically interested in continuous streams of sensed data from the physical world. Query processing systems [8,9,10] provide a high-level user interface to collect, process, and display continuous data streams from sensor networks. These systems are high-level tools that allow rapid-prototyping of wireless sensor network applications. In contrast, writing wireless sensor network applications in a systems language such as C is tedious and error-prone.

Y.-H. Lee et al. (Eds.): ICESS 2007, LNCS 4523, pp. 665–676, 2007.
© Springer-Verlag Berlin Heidelberg 2007

A query processing system abstracts from tasks such as sensing, forming an ad-hoc network, multi-hop data transmission, and data merging and aggregation. A state-of-the-art distributed query system for wireless sensor networks is TinyDB [8] that employs a subset of SQL as an underlying language for queries. In this system the user specifies declarative queries that perform the sensing tasks. For example the query `select avg(temp) from nodes` reports the average temperature of the area covered by the sensor network.

TinyDB forms a routing tree with all sensors in the network. The root of the routing tree is the base station (aka. gateway) that is connected to a PC. Every other node in the wireless sensor network maintains a parent node that is one step closer to the base station. Queries are flooded throughout the network and the query answers are collected and propagated through the routing tree. The query processing consists of three phases: (1) the query preparation phase inputs, parses, and optimises a query at the user's PC, (2) the broad-casting phase injects the sensing and collecting task into the sensor network, and (3) the data collecting phase makes results flowing up and out of the network to the PC where the results are displayed and stored in a disk-based DBMS for later access.

This paper describes a new query processing system for a new wireless sensor network platform Sun SPOT [11,12], that has been developed at Sun Research Labs. This new platform has a 32bit ARM Risc processor, an 11 channel 2.4GHz radio, and approx. 100 times more memory than a state-of-the-art platform such as Berkley Motes [13]. The platform is programmed in Java^{TM}and features a sensor board for I/O and an 802.15.4 radio for wireless communication. The Sun SPOT system runs "Squawk VM" that is a lightweight J2ME^{TM} virtual machine (VM). The VM executes wireless sensor network applications "on the bare metal", i.e., directly on the CPU without any underlying OS, saving overhead and improving performance. With more memory and a faster CPU alternative design decision can be made to minimise energy-costly communication by applying new time-triggered protocols for aggregation.

We have designed and implemented a time-triggered query engine for wireless sensor networks, called SSDQP, which is a distributed query processor that runs on each Sun SPOT. The new platform is programmed in Java. Hence, a clean object-oriented design of the engine was possible.

The contribution of our work is as follows:

- a new design of an acquisitional distributed query (ACQP) system that is time-triggered,
- a time synchronisation mechanism of the nodes that allows a trade-off between cost and accuracy,
- a new communication model for ACQP.

The paper is organised as follows: In Section 2 we survey the related work. In Section 3 we give an overview of our ACQP system. In Section 4 we discuss the trade-off between power consumption and time accuracy of the time synchronisation. In Section 5 we show the advantages of merging results at node level. In Section 6 we draw our conclusion.

2 Related Work

Distributed query processing in wireless sensor networks has been an active research area over the last few years. TinyDB [8] and Cougar [14] represent the first generation of query processing systems in wireless sensor networks. The main objective in these systems has been to preserve the limited power by attempting to reduce the communication overhead. This in turn prolongs the network lifetime.

TinyDB and Cougar [15] provide an SQL-like query language. Sensor data is viewed as a single virtual table. The data is appended at time intervals specified in the query termed as epochs. Results from every sensor find their way to the root node (the node that connects directly to the base station) through a routing protocol. Query lifetime has been introduced for the first time in query processing systems to serve the sensor network applications. The user can specify how long the query should be processed. Pushing computation is used in two forms: partial aggregation and packet merging. In partial aggregation, distributive query operators are used in-network. Intermediate results are then passed to the root to integrate the results. On the other hand, packet merging is used to reduce the communication overhead produced from sending multiple packet headers. Query optimisation is done locally at the central site. Once the query is optimised, the network is flooded through the routing tree to ensure every child node has heard the query. Multiple trees could be formed to allow simultaneous query processing. However, overlay among routing trees can lead to performance decay.

Open issues that have not been addressed in TinyDB and Cougar [15] include multi-query optimisation, storage placement and heterogeneous networks. In multi-query optimisation, the resource utilisation is an open research issue. Storage placement is how to choose nodes that are representative of in-network data and what fault tolerance techniques are required if the storage node fails. TinyDB and Cougar consider only homogeneous networks in which all nodes have the same power. Heterogeneous networks provide new research challenges to the community.

3 Sun SPOT Distributed Query Processing (SSDQP)

The Sun SPOT Distributed Query Processing system consists of two programs: (1) the *query engine* that is executed on the Sun SPOTs and (2) the *control system* on the user's PC that is connected to the base station.

The query engine is implemented as a set of time-triggered tasks. The task scheduler of the query engine executes a task if the start time of the task has been reached. The task scheduler maintains the active tasks in a time queue. Furthermore, tasks can be periodically executed with a fixed time period and the number of repetitions is parameterisable. Tasks can be added and removed from the time queue of the task scheduler. The start time of a task is "global" throughout the network such that sensing and communication can be done in

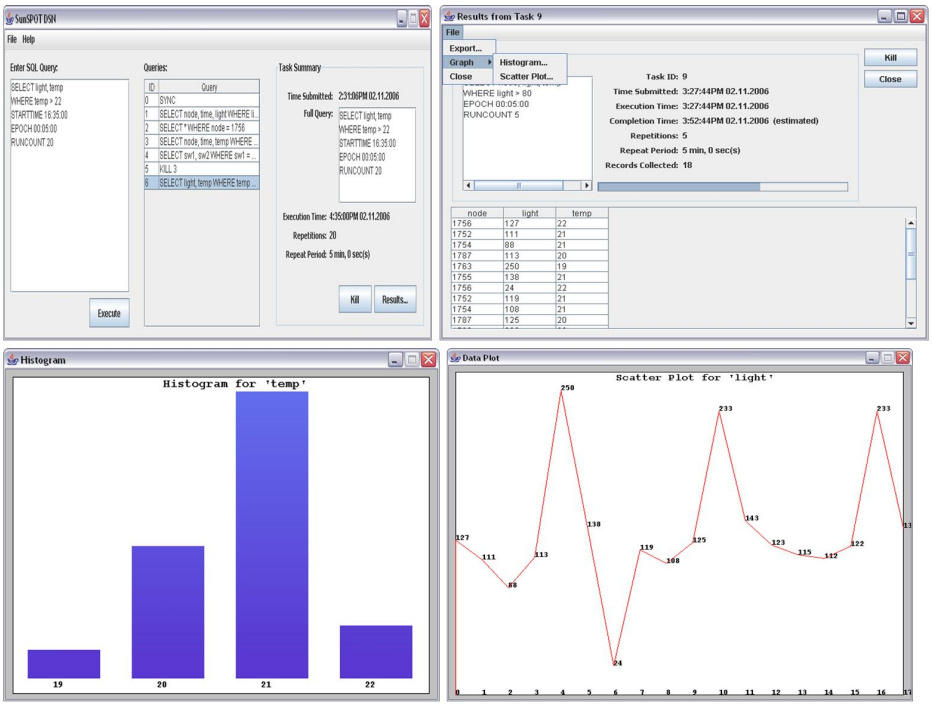


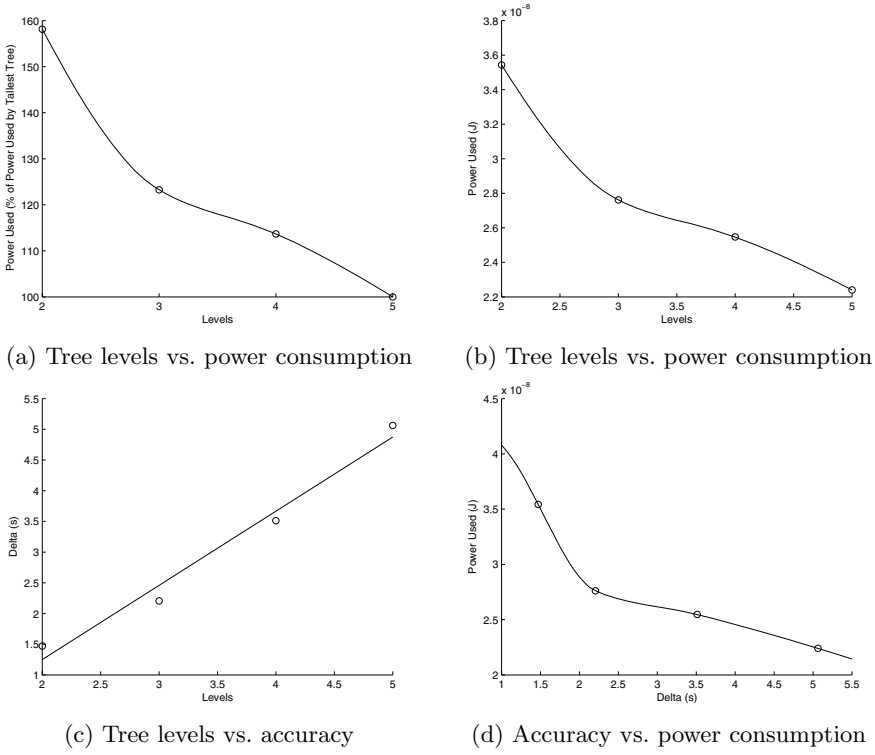
Fig. 1. Screenshots of SSDQP

a synchronised fashion. The query engine has a time synchronisation task that keeps the clocks of the Sun SPOTs in the network in sync.

Query tasks are composed of relational algebra operations that operate on relational tables. Since all sensor readings of the Sun SPOTs are integer values, the system does only support integer attributes in the relational tables. The query engine supports all the fundamental query operators including selection, projection, join and aggregation. In addition to these basic functionalities, there are

- the *sense operation* that reads the values of the sensors and creates a result table with the sensor readings,
- the *forward operation* that takes the input table and forwards the table to the parent node in the routing tree,
- the *merge operation* that receives result tables from the children in the routing tree, merges the tables, and gives as a result the merged tables.

The query operations are represented as expression trees in the query engine. A string representation of the expression tree is used for its contruction, which is sent from the control system to a Sun SPOT node. To minimise the size of



(a) Tree levels vs. power consumption

(b) Tree levels vs. power consumption

(c) Tree levels vs. accuracy

(d) Accuracy vs. power consumption

Fig. 2. Power Consumption

messages (and therefore energy), a basic data compression method is used. The data compression achieves compression rates of about 62% in practice.

The control system runs on the user's PC that is connected to the base station. It is also written in Java. The control system

- inputs and parses SQL queries, optimises the queries, and translates them into distributed relational query operations, which are deployed in the network,
- collects the data from deployed queries,
- manages deployed queries, (i.e., status of deployed queries, termination of deployed queries, etc.),
- provides the global time to all nodes in the network,
- displays and depicts results of queries.

The control system has also a friendly graphical user interface for query input and result visualisation as shown in Figure 1. The system is fully written in Java following true object-oriented software engineering practices. This gives our system the advantage of simple system maintenance and extension.

4 Accuracy Guaranteed Efficient Time Synchronisation

Wireless senor networks are dynamic. New nodes join the network and others die frequently. Static time synchronisation is infeasible in such computing environments. Consequently, time synchronisation techniques run frequently consuming the network energy and shortening its lifetime. Accurate time synchronisation leads to accurate query results due to the low time shifts among network nodes. The performance of time synchronisation techniques degrades with the increase in the network size. This occurs due to the increase in the number of hops to reach distant nodes from the base station. This problem could be overcome by increasing the power level of network nodes. This in turn decreases the number of hops to reach distant nodes. Thus, accurate time synchronisation is achieved at the cost of higher energy consumption. We have developed a parameterised optimiser in our SSDQP system that makes a trade-off between accuracy of time synchronisation and consumed energy. The user inputs an acceptable level of time shift (Δ) between the system time at the base station and the system time at a node in the network. Depending on the application the time shift Δ varies. Our optimiser chooses a network tree topology that minimises the consumed energy (E) and achieves a level of time shift $\Delta' \leq \Delta$. Let us assume that the number of hops of node u to the base station is denoted as h_u and the energy that is consumed for a single hop is e. The total energy E for time synchronisation is given by

$$E = \sum_{u \in T} e \cdot h_u \tag{1}$$

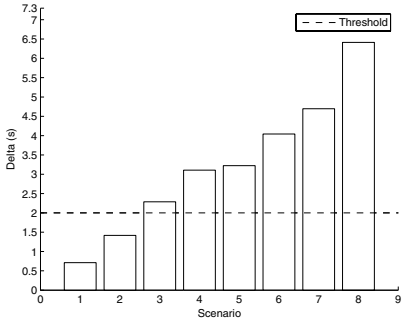
where T is the topological tree used for time synchronisation. The achieved time accuracy Δ' depends on the maximum h_u in the network, i.e., $\Delta' = \mu \max_{u \in T} h_u$ where μ is the time shift introduced by a single hop. We seek for a topology such that E becomes minimal and $\Delta' < \Delta$.

4.1 Experimental Study

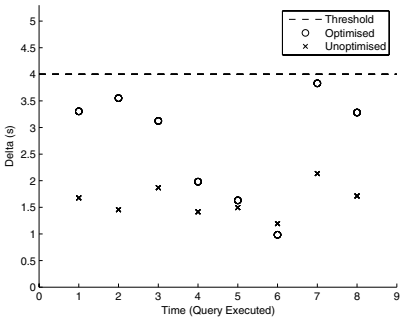
The aim of our experimental study is to provide simulation-based evidence of the significance of our efficient time synchronisation approach described earlier. The experimental setup is described in the following: Considering a full binary tree of height (number of levels)= 5. The radio power setting(I) to reach a parent

- 1 level above is $I = p = p$
- 2 levels above is $I = p + \frac{p}{3} = \frac{4p}{3}$
- 3 levels above is $I = p + \frac{2p}{3} = \frac{5p}{3}$
- 4 levels above is $I = p + p = 2p$

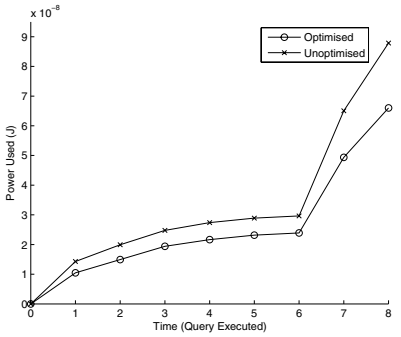
where p is the power setting required to reach a parent one level above. The delta between levels is calculated with a normal distribution, $\mu = 0.4s$. The goal of our first experiment is to show the trade-off between accuracy of time synchronisation and the energy consumed. By varying the number of levels in the routing tree, depending on the accuracy of time synchronisation required, it can be shown that a



(a)Delta Variations



(b) Optimisation on the accuracy



(c) Power consumption over time

Fig. 3. Accuracy vs. Power Consumption

trade-off between time synchronisation and power consumption of a node can be achieved. A reduction in the number of levels of the tree is accomplished by increasing the radio power level of a node so that it can transmit data to its grandparents, great-grandparents, etc. effectively skipping levels. For a tree with 5 levels, trees with 4, 3 and 2 levels can be constructed from the nodes of the original tree, given that all nodes are able to transmit to one another with a high enough radio power level. For a given delta in time synchronisation required by the users query, one routing tree from these 4 trees can be selected in order to achieve the delta, with a trade off in power. As shown in Figures 2(a)-(d), a reduction in levels of the tree increases power consumption (for one time synchronisation of the entire tree) substantially, however will yield a smaller delta.

In the first experiment we provide evidence that with no optimisation of the routing tree, the accuracy of time synchronisation achieved can exceed the accuracy pre-specified by the user. Assuming a system operating with a fixed routing tree which has been designed to achieve maximum power efficiency, the tree which is being operated on has 5 levels and various subtrees may be synchronised depending on the query scenario executed. Suppose that the user requires a maximum delta in time synchronisation of 2s. As shown in 3(a), although for

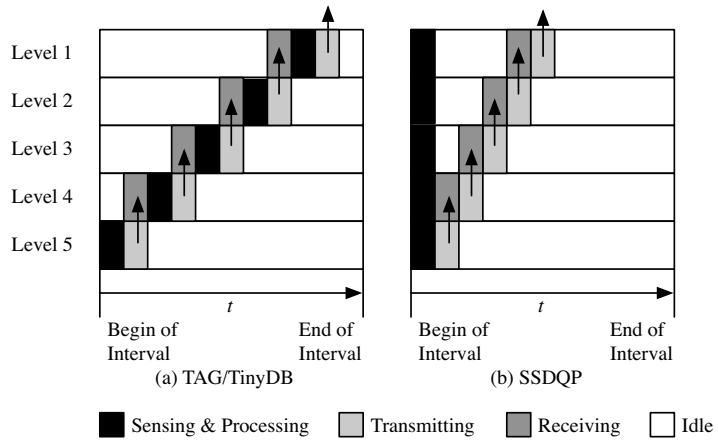


Fig. 4. Communication Model

some small subtrees the accuracy of the synchronisation is achieved, for queries executed over larger subtrees, the delta exceeds the desired threshold delta. This is done at the cost of unnecessary power consumption, which, in time, can substantially reduce the lifetime of the network. Take again a system with a fixed routing tree however this time it has been designed to achieve maximum accuracy in time synchronisation. Assuming that the user requires a maximum delta of 4s, as shown in 3(b), the unoptimised system always achieves a delta substantially below the threshold in all scenarios. Our system, which selects a tree based on the delta, also achieves a delta below the threshold however it is much closer to the threshold. 3(c) shows the power consumption over time of the network as a whole with each system, as various queries are executed. As shown, there is a significant difference in power consumption, particularly when large subtrees are being operated on (queries 7 and 8).

5 Communication Model

In the design of SSDQP we had the choice to either adopt the communication model of TinyDB [8] (and TAG [16] respectively) or to create a new communication model. Since the Sun SPOTs have more memory and more computational power than Berkley Motes; we designed a new communication model. This new communication model is optimised for repetitive queries and has following properties:

- timeliness of sensing, i.e., all nodes in the network sense at the same time, and
- minimised communication overhead achieved by a synchronised merge of results. Therefore, the new communication model uses less energy.

The communication model of TinyDB and TAG [16] is illustrated in Fig. 4(a). The partial information of a query flows up the network toward the root node.

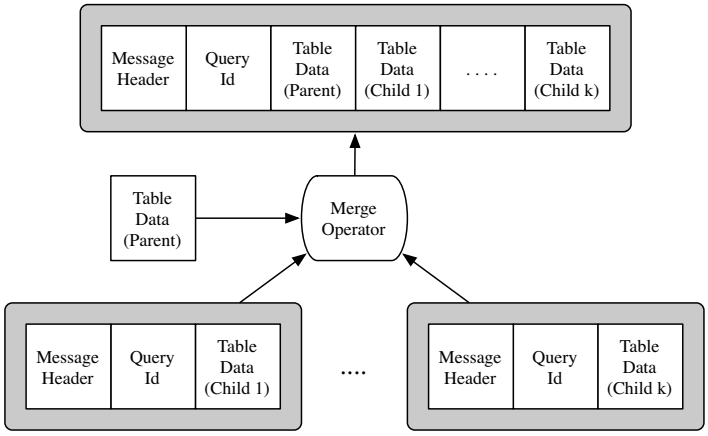
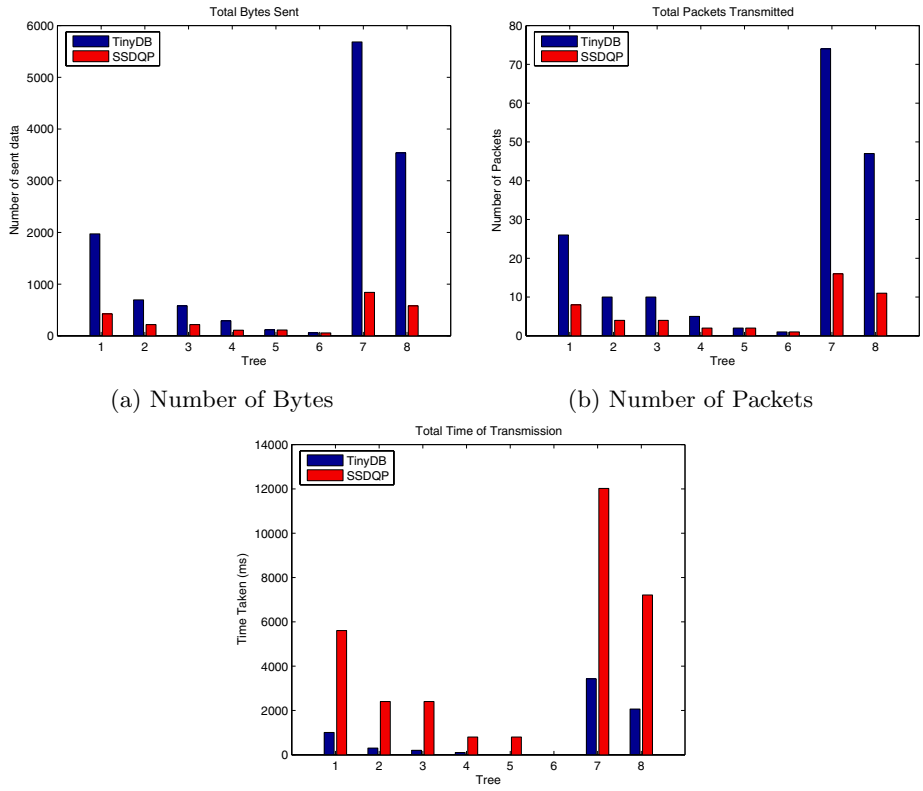


Fig. 5. Merge Operator

In a sensing interval (aka. epoch) a sensor node has four different states: a sensing and processing state, a sending state, a receiving/listening state, and an idle state. If the node is an inner node in the tree, the sequence of states is as follows: (1) receiving state where all the information of the children is gathered, (2) sensing and processing state, (3) sending state in which the information is forwarded to the parent node in the tree network and (4) followed by the idle state. If the node is a leaf node, then the receiving state is omitted because there are no children attached to the sensor node.

The disadvantage of the TinyDB/TAG model is that the point in time when the sensing and processing is performed depends on the tree level in the network. Note that for queries that do *not* have aggregation (e.g. AVG, SUM, etc) the sensed data is directly forwarded to the root node without aggregation. The information is "bubbled up" the network tree. In contrast, the SSDQP communication model de-couples sensing from the aggregation as illustrated in Fig. 4(b). The task scheduler of a sensor node performs two tasks for a single query: the first task performs sensing, and the second task performs the aggregation and the forwarding to the parent node. Both tasks are time-triggered. The second-task needs to be scheduled such that there is enough time for the children to provide their aggregated information. Therefore, the point in time of the second task depends on the child that needs the longest time span to provide the information, i.e. the child whose sub-tree has greatest depth. Even for simple queries without aggregation the partial information of query is merged at all levels before forwarded to the parents in the network tree.

Partial information collected by several sensor nodes is sent in a packet structure consisting of three parts: message header, a query identification, and the actual partial result of a query. The packet structure imposes communication overhead stemming from the message overhead of the Sun SPOT network as well as book-keeping information for the query system. We seek for a communication model that minimises the total number of packets to reduce the communication



(a) Number of Bytes (b) Number of Packets

(c) Latency

Fig. 6. Comparison with TinyDB communication model

overhead. We achieve this in our model by a *synchronised merge* operation. This means that a node merges the information of its own data and the data of its children before forwarding it to its parent node. If there are n nodes in the network, we need for one epoch exactly n messages whereas TinyDB/TAG forwards the information without merging the partial information or an on the fly merging in the network layer is used. The consequence is that in the worst case TinyDB has $O(n^2)$ messages for a single query.

The merge operation of a node is depicted in Figure 5. The disadvantage of the merge operation is that more memory is needed in a sensor node and that the latency of a query increases because a time slack for merge operations is to be taking into account for.

A simulation of both communication models was conducted on 8 network trees of various depth and density. For the simulation we used the query `select * from sensors`. The comparison of both communication models is shown in Fig. 6. The first bar-chart in Fig. 6(a) shows the total amount of bytes sent for a single epoch. The information sent in the SSDQP communication model is

significant less because there are significant less packets sent in total (cf. Fig. 6(b)). The total number of sent bytes is proportional to the energy used for the transmitter of the radio and has a great impact on the longevity of the nodes in the network. Especially for trees with larger depth the SSDQP communication model is superior to the communication model of TinyDB because the communication overhead for a single message packet is large in comparison to the data length of a sensor reading.

However, establishing well defined merge points for an inner node increases the latency (cf. 6(c)), i.e. the time span between the begin of an epoch and the point in time when the base station receives the result of a query. Because the information is immediately streamed from the nodes, the TinyDB model is more responsive.

6 Conclusion

In this paper, we have presented our novel distributed query processing system (SSDQP). The system is built on the new Sun SPOT platform from Sun Microsystems. Special considerations in the system design have been paid to preserve energy by minimising the required communication overhead. This paper has proven experimentally that our time synchronisation optimiser can achieve the required accuracy while minimising the required energy. An experimental comparison between our system and TinyDB has shown that our system outperforms TinyDB in terms of communication overhead.

References

1. Zhao, F., Guibas, L.: Wireless Sensor Networks – An Information Processing Approach. Elsevier / Morgan-Kaufman, Amsterdam (2004)
2. Culler, D.E., Hong, W.: Introduction. Commun. ACM **47**(6) (2004) 30–33
3. Szewczyk, R., Osterweil, E., Polastre, J., Hamilton, M., Mainwaring, A., Estrin, D.: Habitat monitoring with sensor networks. Commun. ACM **47**(6) (2004) 34–40
4. Chen, L., Chen, Z., Tu, S.: A realtime dynamic traffic control system based on wireless sensor network. In: ICPPW '05: Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW'05), Washington, DC, USA, IEEE Computer Society (2005) 258–264
5. Lu, K.C., Wang, Y., Lynch, J.P., Lin, P.Y., Loh, C.H., Law, K.H.: Application of wireless sensors for structural health monitoring and control. In: Proceedings of KKCNN Symposium on Civil Engineering, Taiwan (2005)
6. Liu, W., Zhang, Y., Lou, W., Fang, Y.: Managing wireless sensor networks with supply chain strategy. In: QSHINE '04: Proceedings of the First International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks (QSHINE'04), Washington, DC, USA, IEEE Computer Society (2004) 59–66
7. Woo, A., Madden, S., Govindan, R.: Networking support for query processing in sensor networks. Commun. ACM **47**(6) (2004) 47–52
8. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. **30**(1) (2005) 122–173

9. Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., Silva, F.: Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Netw. **11**(1) (2003) 2–16
10. Demers, A., Gehrke, J., Rajaraman, R., Trigoni, N., Yao, Y.: The cougar project: A work in progress report (2003)
11. Simon, D., Cifuentes, C., Cleal, D., Daniels, J., White, D.: Java on the bare metal of wireless sensor devices: the squawk java virtual machine. In: VEE '06: Proceedings of the 2nd international conference on Virtual execution environments, New York, NY, USA, ACM Press (2006) 78–88
12. Microsystems, S.: (Sun spot world) http://www.sunspotworld.com/.
13. Hill, J., Horton, M., Kling, R., Krishnamurthy, L.: The platforms enabling wireless sensor networks. Commun. ACM **47**(6) (2004) 41–46
14. Yao, Y., Gehrke, J.: The cougar approach to in-network query processing in sensor networks. SIGMOD Rec. **31**(3) (2002) 9–18
15. Gehrke, J., Madden, S.: Query processing in sensor networks. IEEE Pervasive Computing **03**(1) (2004) 46–55
16. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. In: OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, New York, NY, USA, ACM Press (2002) 131–146