Dynamic Approximative Caching Scheme for energy conservation in Wireless Sensor Networks

Nils Hoeller, Christoph Reinke, Jana Neumann, Sven Groppe
Florian Frischat, Volker Linnemann
Institute of Information Systems
University of Luebeck
Luebeck, Germany
lastname@ifis.uni-luebeck.de

ABSTRACT: *The crucial issue in Wireless Sensor Networks is saving energy so that the lifetime of in-field deployments can be extended. It was documented that the communication is generally the most energy consuming task and needs to be reduced in order to build resource-efficient long-term applications. It is evident that the communication demand for retrieving query results from deep within the sensor network is typically high. As a result, frequent non-continuous data acquisition consumes a lot of energy and shortens the lifetime of the sensor network significantly. We address the issue of optimizations for processing high amounts of unique queries by using a dynamic adaptive caching scheme: the DACS. In DACS query results can be retrieved from caches that are placed nearer to the query source instead of sending queries deep into the network. The communication demand can be significantly reduced and the entire network lifetime is extended. To verify cache coherence in sensor networks with non-reliable communication channels, an approximative update policy is used. To localize the adequate cache adaptively, model-driven queries including a degree of demanded result quality can be defined. The entire logic is thereby processed by DACS and hidden to the user. We have shown that the significant energy conservation is proven in evaluations that include real sensor node deployments.*

Keywords: Wireless sensor networks, Dynamic adaptive caching scheme, Energy conservation, Sensor nodes

Received: 18 September 2010, Revised 20 October 2010, Accepted 28 October 2010

1. Introduction

Currently, the new forms of highly distributed networks consisting of tiny sensory processing units denoted as Wireless Sensor Networks (WSNs) using the evolving techniques of microprocessor and communication technology are emerged. Researchers characterize the WSNs as large scale, wireless ad-hoc networks that are formed by tiny sensor nodes / motes typically for monitoring and environmental analyzation tasks [1], [2], [3]. in WSNs Each sensor node is equipped with an energy efficient microcontroller, a radio unit, diverse sensor units to analyze the environment and a limited energy supply. Sensor nodes are non-reliable platforms as they are limited in their lifetime due to the energy constraints and lack of robust communication due to interference. Previous work in data management has therefore been focussed on energy efficient data acquisition to extend the lifetime of the networks and energy efficient communication usage with additional improvements for more robust data transmission.

Significantly, the main data acquisition scenario in WSNs is to share measured information over non-reliable communication channels. Queries are inserted in the network by gateway nodes. The evaluation of these queries and the transmission of results need to be optimized regarding the present resource constraints. The main research goal is to extend the lifetime of the sensor networks by reducing the communication overhead during query processing. In-network aggregation techniques

therefore have been introduced. While initial approaches [4], [5] rely on fixed routing schemes, advanced solutions question the usage of a fixed topology with regards to the non reliable communication and propose approximative aggregation schemes using broadcast messages for enhanced reliability [6]. Moreover, optimizing the sensor nodes activity, e.g. adjusting the actual sampling rates to the acquisition demand, by analyzing active queries has been suggested [4].

Together, the previous acquisition optimization strategies presume the query result retrieval from deep within the network at the data sources. Nevertheless, the communication demand can be further reduced by using only a limited part of the entire network to evaluate the query. One possible approach is to use data caches instead of sending queries to each data source. As a result the communication demand can be significantly reduced.

In this work we introduce the **D**ynamic **A**pproximative **C**aching **S**cheme: **DACS**. Previous work on data caching in WSNs require the usage of hierarchical communication topologies with deterministic data routes, e.g. [7]. Unlike these approaches, **DACS** works without topology assumptions and is robust against communication limitations. The framework consists of a dynamic distribution of data caches in the sensor network. In order to extend the lifetime of the network and take care of the unreliable communication channels, a weak cache coherence is discussed that is based on an approximative update policy. By attaching a result quality requirement to a query, **DACS** automatically retrieves query results from caches nearer to the data sink and further ensures that the degree of result quality is not violated. As a result, queries do not need to be sent to all sensor nodes deep within the network which reduces the communication overhead. Evaluations show that by using **DACS** and by accepting a minimal deviation in the query result the network's lifetime can be significantly enhanced.

In Figure 1 we give a simplified example for a network running DACS. In an uniform distributed network data source nodes are cached throughout multiple layers. Gateway nodes are used to send queries into the network. In this example we define a data source node (*SN*) and four gateway nodes (*GW*). Every other node is a possible cache node. However, unlike this simpified example, **DACS** supports unlimited, randomly placed data source nodes, gateways and cache nodes, e.g. each data source node is cached separately by **DACS** and each data source node acts also as a cache for other nodes. Thereby, the maximum memory consumption can be limited. Due to the implicit redundancy on the update layers we verify that each source node can be cached.

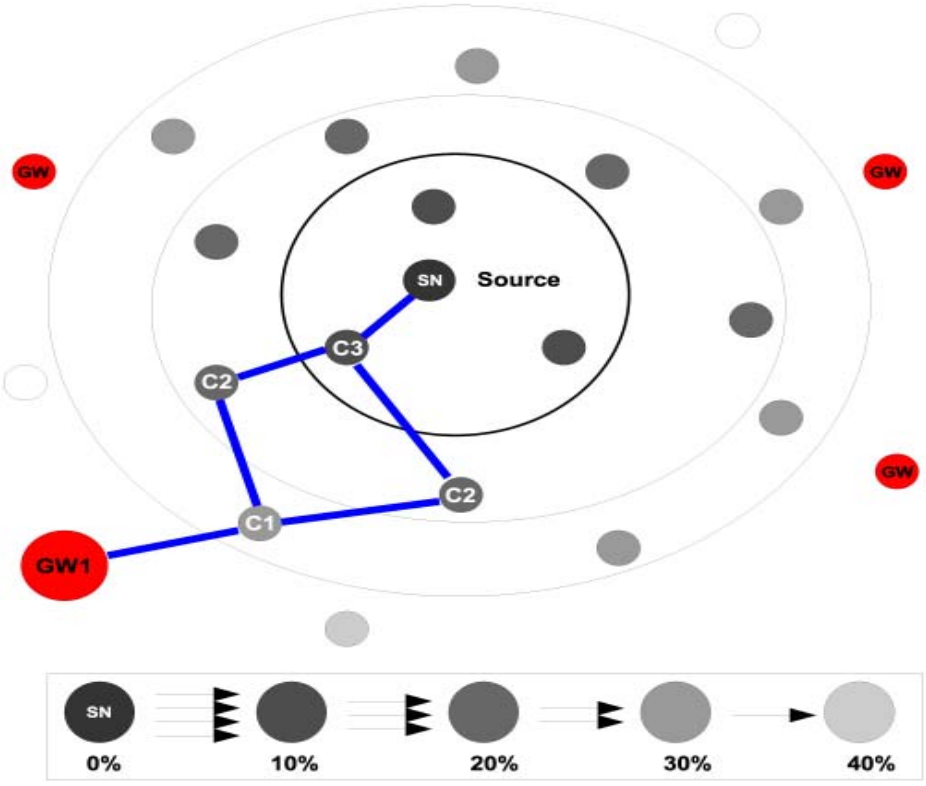

Figure 1. Overview DACS Simple Network Model Example

The basic concept of DACS is to use cache nodes instead of the actual data source nodes for retrieving information, e.g. temperature values. Hereby, the communication demand can be significantly reduced by using caches that are layered nearer to the gateway than the actual data source, e.g. using the caches C1, C2 or C3 instead of the data source SN. However, cache nodes need to be kept coherent. Sensor node communication in WSNs is generally unreliable. As a result, strict coherence or even consistency is not possible. Besides, keeping every cache node coherent comes at high communication costs for rapidly changing data at the data sources. To overcome these limitations, DACS uses an approximative update policy. As shown in Figure 1, the cached data can deviate from the actual source data based on the distance between cache and data source. For a simplified example, we can define a maximum deviation of the cache value for each source-cache distance as shown in Figure 1, e.g. using a linear increasing deviation of 10% per hop.

As a result, the estimated communication demand for cache updates decreases with increasing distance from the data source as highlighted by the number of arrows. In this work, we show how DACS implements this approximative cache coherence. Using approximative caches requires using modeldriven queries since it is not obvious how deep queries should be sent in the network to accordingly reach caches with lower deviation. DACS hides the cache localization process to the user by allowing to issue queries with additional demand of result quality, e.g. get temperature values that might include an overall maximum deviation / error of 30%. In this paper, we discuss this localization approach for unlimited data sources and data caches.

The remaining parts of the paper are organised as follows: In the next section we give an overview on important previous and related work. In Section III we give a detailed introduction into **DACS** covering issues of Cache Placement (Section III-A), Cache Coherence (Section III-B) and Cache Localization (Section III-C). In Section IV we evaluate the framework and show results on the communication efficiency and the robustness of the framework. We conclude our paper in Section V and show possible areas of future work.

2. Related Work

In this Section we present previous work related to data caching and data management in WSNs. Previous work in data management in WSNs was initially focussed on deep in-network aggregation and data acquisition optimizations to save energy during processing query results [4], [5]. Using caching structures to optimize in-network query processing in WSNs was firstly discussed for the query engine TinyDB in [7]. While there has been a clear recommendation for using caches to evaluate queries closer to the actual data sink and hence to save energy, the used strategies provide only a simple round-based caching scheme.

In this approach, in a static aggregation tree aggregation values of child nodes are cached by their parents for a determined number of rounds. The actual aggregation values are not regarded and the caching scheme is strictly connected to the TAG tree topology. Beside this approach other caching strategies for WSNs have been presented in [8], [9], [10], [11], [12]. Most of these these approaches require the usage of hierachical communication topologies with deterministic data routes, e.g. TAG [7]. Hence, we denote them as hierarchical caching strategies. The strategies can be further divided in nonapproximative and approximative approaches. In detail Shashi et al. describe an optimal cache placement strategy in tree topologies by finding nodes in Steiner Data Cashing Trees in a scenario where multiple subscribers are receiving data from one source [8]. Multi-source scenarios like described in this work are left out for future work. Chand et al. describe a cooperative caching scheme to improve data access performance and availability in mobile ad-hoc networks [9]. Their work is focused on a new utility based cache replacement strategy in contrast to a usage based policy. However this approach is not optimized for energy constrained WSNs and therefore seems not to be applicable in the presented way. In [10], Rahman et al. propose strategies for improving the energy efficiency of WSNs. The presented caching strategy is focussed on avoiding unnecessary sensing by estimating the data change frequency, comparable to the acquisitional data processing strategy of TinyDB. Strategies for in-network caching to optimize query evaluation like described in this paper are not presented. Jung et al. focus on an external cache-based sensor network bridge to avoid querying the entire WSN by using cached results [13]. The approach works for non-constrained external devices (e.g. gateways) and hence is not applicable for in-network query optimization.

3. Contribution

In this Section we give a detailed description of the **DACS** Framework. The basic concept of **DACS** is to provide a general

data cache solution that does not rely on any given topology assumptions. The design goal of **DACS** is to reduce the communication overhead by letting queries be evaluated by data caches on the route to the actual data source. Reducing the communication overhead increases the lifetime of the network significantly.

For setting up a general caching scheme for WSNs, the following issues need to be reviewed:

• **Cache Placement (Section III-A):** A general strategy on where to place data caches on the communication routes needs to be resolved. A general network model is the basis of this strategy.

• **Cache Coherence (Section III-B):** Data caches need to be updated when new data occurs at the data source.

• **Cache Localization (Section III-C):** Queries need to be redirected to adequate data caches for evaluation. The localization process should be hidden to the user, e.g. by a black box behaviour.

3.1 Cache Placement and Network Model

WSNs are highly dynamic networks. The exact position of nodes after deployment, e.g. out of a plane, is often not precise and the lack of communication robustness does not guarantee fixed data routes in general. As a result, the placement of data caches cannot be verified before deployment and the caching structure needs to be set up during runtime autonomously. **DACS** is able to adapt the distribution and placement of data caches dynamically in case of changing network conditions. Additionally, no assumptions on the network topology are made. The organization of the cache structure is a collaborative decision of the network itself.

The following general network model is defined to make **DACS** suitable for most application scenarios:

• A network consists of a uniform distribution of N nodes.

• In the network each node except the gateway nodes can produce data, e.g. measure environmental information. Measurements are taken on predefined intervals continuously.

• The general **DACS** approach supports multiple gateways, whereby each gateway is managed separately in identical manner. However, in the following discussion, we assume the network to have one dedicated gateway for better understanding.

• To avoid bottlenecks on the routing path, **DACS** relies on using broadcast communication. We desist from using fixed topologies to improve failure tolerance and to support a maximum number of deployment scenarios. However, to avoid energy unefficient flooding of the network we use ring-oriented, directed communication as proposed in [6]. As a result the energy demand is only minimal higher than existing static topology approaches with unicast delivery but the fault tolerance is significantly improved. As porposed in [6] this routing technique is optimal for sensor networks that generally are sensitive to node and communication faults resulting in unpredictable network conditions, e.g. network partitioning.

In **DACS** we desist from using dedicated cache placements. Instead, every node beside its own measurement task is a potential cache for other nodes. Thereby, nodes decide which data they cache based on their distance to the data source that is determined by the hop count of update messages. Nodes can then be classified concerning their membership to distance layers. We therefore define two different layers:

1) *Cache Layer:* The cache layer defines the distance between the gateway and a cache node, denoted by d_{GC}. For example, in Figure 1 *C1* is on cache layer 1 for Gateway *GW1*.

2) *Update Layer:* The update layer defines the distance between the data source and a cache node, denoted by d_{SC}. For example, in Figure 1 *C1* is on update layer 3 for the source node *SN*.

As denoted previously, we use a ring-oriented broadcast communication as proposed in [6], e.g. cache results are sent to the corresponding gateway over decreasing cache layer and update messages are sent to the caches over increasing update layer. The actual placement of the cache is now adjustable by placement rules, e.g. caches are placed every second or third layer, depending on the memory and energy resources of the application scenario. The actual update logic is defined by the cache coherence protocol that is described in the next section. Finally, by avoiding dedicated caches and instead using cache layers we further introduce an implicit data replication which optimizes the node failure tolerance and makes the networks more stable concerning communication path failures.

3.2 Cache Coherence

Cache consistency predefines that for each point in time the cache is consistent to the data source whereas cache coherence demands that the cache is consistent to the data source during the evaluation of queries. Cache coherence therefore significantly reduces the effort of keeping the cache up-to-date and hence the communication demand. However, both claims can mostly not be satisfied in WSNs because of the unreliable communication and the energy demand of continuous cache updating that conflicts with the general hardware restrictions in WSNs. In detail, the times when queries have to be evaluated are generally not predictable resulting in a continuous updating process of the caches which significantly reduces the lifetime of the entire WSN due to the communication overhead.

To overcome these problems, **DACS** introduces an approximative cache update policy. Each cache node does not store the actual value of the data source but rather stores a value for that a maximum deviation / error is guaranteed. An update is only processed if the maximum deviation is exceeded. The maximum deviation for each cache item is set based on its distance to the actual data source.

In Figure 1 we give an example for an approximative update policy with linear increasing error (10% per hop on distance d_{SC}). In the following, we give a detailed description on how **DACS** supports this policy.

As described in the previous section, nodes are classified based on their membership to cache and update layers. Each logical update layer consists of nodes with the same distance d_{SC} to the data source and includes a maximum deviation / error regarding the cached values and the values of the actual data sources.

This error is defined by a function $\Upsilon(d_{SC}) \rightarrow e_x$ that calculates the maximum error e_x for a given cache data source distance $x = d_{SC}$. The function is initially known to all nodes based on the application scenario and the efficiency predefi- nitions. For Figure 1 it can be defined as $\Upsilon(d_{SC}) = d_{SC}/10$. In general, the higher the steepness of the function, the lower the estimated communication demand in general for updates. The high impact of choosing the error function according to efficiency predefinitions that can result into significantly lower communication demand is also shown in the evaluation of this work in Section IV. Moreover, the function can be adjusted concerning the estimated number of queries per update as also shown in Section IV.

On incoming update messages, nodes decide based on the function $\Upsilon(d_{SC})$ whether an update message needs to be processed, e.g. the value needs to be cached and the message needs to be forwarded to higher update layers. **DACS** currently supports three forward policies, that can easily be extended for future work:

1) *Decision at Source Node:* The source node tracks the progress of cache values based on a virtual layer scheme and determines itself how many hops an update needs to be sent in order to verify the update policy. This approach prevents policy violation and is denoted as stable approach.

2) *Forward If Updated:* Cache nodes forward as long as the update policy results into cache updates. The forward process is a dynamical decision on the update layer path. However, monotonic changes in source values can produce temporary violation of the demanded deviation gradation, e.g. higher update layers exceeding the maximum deviation temporary. It can be shown that the maximum error can be nearly squared. Nevertheless, this simple solution performs well in average without policy guarantees.

3) *Weighted Forward If Updated:* The problem of the error policy violation of the *Forward If Updated* forward strategy can be solved by dynamically weighting the function $\Upsilon(d_{SC})$. However, this approach requires additional logic on the routing path and is out of scope of this paper due to its complexity.

In the experiments of this work we have used the forward policies 1) and 2) which both ensure a stable performance. In summary, we have shown in this section how cache coherence can be guaranteed by allowing an average error gradient in the network based on an adjustable error function. In the next section we discuss how caches can be localized for given user queries.

3.3 Cache Localization

In **DACS**, caches can be localized using model-driven approximative queries [14]. The query issuer defines a quality demand that needs to be resolved by **DACS** autonomously. **DACS** estimates the distribution of the nodes in the network and derives

the error distributions of the caches in the network. Based on the distributions **DACS** extrapolates the next cache layer that fulfils the quality requirements. This entire process is hidden to the query issuer which makes **DACS** an optimal solution for non expert sensor network users.

As described in the previous section, cache nodes are classified by their membership to cache layers, e.g. the distance d_{GC} between a gateway and the cache nodes. In **DACS** the user defines an overall maximum average error requirement $\in$ along his query, e.g. get all temperature values with an overall maximum average error of $\in$. **DACS** automatically retrieves a corresponding cache layer, e.g. the maximum hop for the query messages, so that the requirement can be satisfied.

We therefore define a function $C(\in) \rightarrow c \in \mathbb{N}$ that looks up a corresponding cache layer c that fulfils $\in$. The resulting cache layer guarantees that the overall average error of the results does not exceed $\in$. Hereby, it is important that not onlythe maximum deviation of the cache layer is relevant to the overall average error but also the amount of nodes between the cache layer and the gateway that will send exact results. On the other side, **DACS** chooses the cache layer as close as possible to the gateway without violation of the requirement $\in$ to optimally reduce the communication overhead.

The localization of the cache nodes depends on two factors:

1) the error function $\Upsilon(d_{SC})$

2) the distribution $F(X)$ of the nodes on the hop layers of the network starting from the gateway. The distance from the gateway to a node is defined by d_{GN}. The density function is denoted as $f(X)$. In Figure 2 we show a possible gaussian distribution which will be used throughout this section as an example an is, as we show later in this section, representative for many network scenarios.

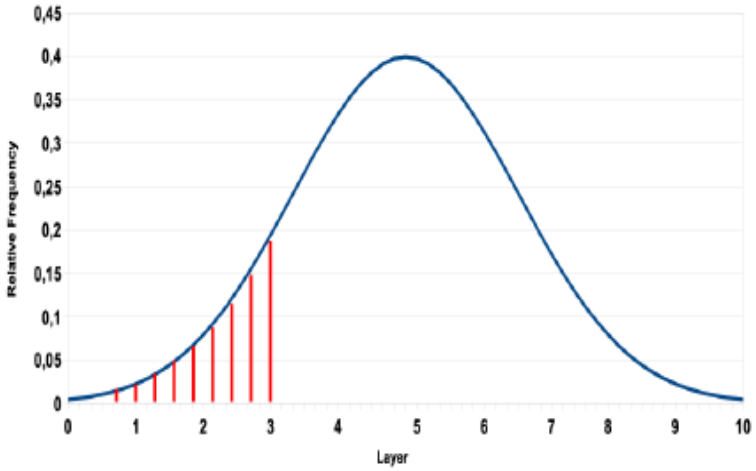

Figure 2. Network Node Distribution Density $f(x)$ on Hop Layer

Localization based on Node and Error Distributions

The function $C(\in)$ can be derived inductively over the used cache layers as shown in the following cases:

CASE 1: Cache Layer $C = 0$

In this initial case, we assume that the gateway acts as a cache. By using the approximative update policy the maximum error of the cached value of a node n with distance $x = d_{SC} = d_{GN}$ is defined by $e_x = \Upsilon(d_{SC}) = \Upsilon(d_{GN})$. In example, a node that is 5 hops away from the gateway will be cached by allowing a maximum deviation of $\Upsilon(5)$.

Accordingly, by reviewing all nodes in the network we can derive an error distribution for a given cache layer C denoted as $G(C, e_x)$ (with density $g(C, e_x)$) that determines the amount of nodes that are cached by the cache layer C allowing a maximum error ex. For the present case (C=0) the error distribution is now directly determined by the given node distribution density $f(X)$:

$$g(0, e_x) = f(\Upsilon^{-1}(e_x)) \qquad (1)$$

In detail, the amount of nodes that are cached with a maximum error of *ex* is the amount of nodes that are on a layer with distance d_{GN} and for that the equation $\Upsilon(d_{GN}) = ex$ is true. We can retrieve this layer by resolving the function $\Upsilon^{-1}(e_x)$ and get the amount of nodes on this layer from the density function $f(\Upsilon^{-1}(e_x))$.

As shown in Figure 3, this theoretical derivation means that for the initial case ($C = 0$) we can directly determine the amount of nodes that are cached with a certain error e_x from the general node distribution.

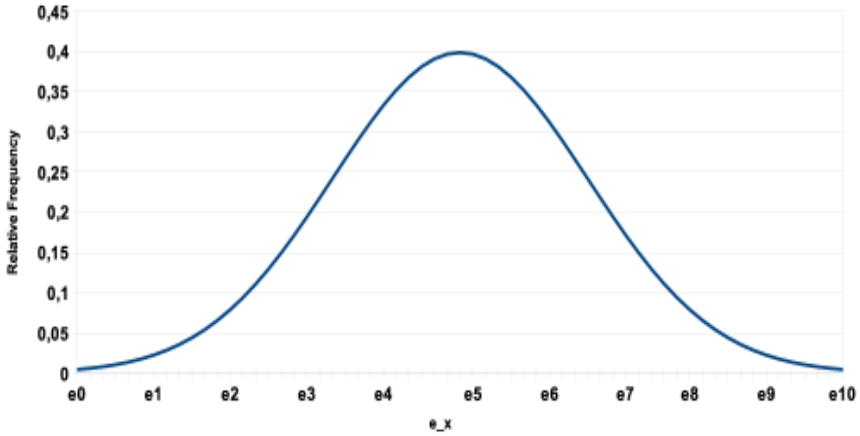

Figure 3. Network Error Distribution for Cache Layer 0: $g(0, ex)$

The estimation value of the derived error distribution for cache layer 0 is the worst case overall average error of **DACS** . As denoted previously, we now have to review the case that we retrieve results from cache layer deeper in the network whereby nodes between the gateway and the cache will answer with exact results without error.

CASE 2: Cache Layer $C > 0$

We continue the inductive derivation of $C(\in)$ by reviewing the usage of caches deeper in the network ($C > 0$). The overall average error of a query result can again be determined by the expectancy value of an error distribution $g(C, e_x)$. Nevertheless, we have to take care of the nodes that are positioned between the cache layer *C* and the gateway, as they will send exact results.

We give an example in Figure 2. By using cache layer $C = 3$, all nodes on previous layers will answer with exact results. These nodes are marked by the red area and the number of these nodes is directly determined by $F(3)$.

In general, by using cache layer *C*, $F(C)$ nodes send exact results ($e_x = 0$) and hence we can derive

$$g(C, 0) = F(C) \qquad (2)$$

In the following, we need to determine the error distribution of the rest of the nodes with error $e_x > 0$ that are actually cached by the cache layer *C*, e.g. the nodes from layer $C+1$. As shown previously, the amount of nodes on layer $C + 1$ is determined by the node distribution $f(C + 1)$. Accordingly, because these nodes are only one hop away from the cache layer the maximum error $e1$ of the cached values is $\Upsilon(1)$. Based on the node distribution, we then retrieve the density of error e_1 as

$$g(C, e_1) = f(\Upsilon^{-1}(e_1) + C) \qquad (3)$$

By reviewing all layers of cached nodes, we retrieve the general error distribution for a used cache layer *C* ($C > 0$):

$$g(C, e_x) = \begin{cases} F(C) & , for\ e_x = 0 \\ F\left(\Upsilon^{-1}(e_x)\right) & , for\ e_x > 0 \end{cases} \qquad (4)$$

In other words, we retrieve the actual error distribution for a cache layer *C* ($C > 1$) by left shifting the error distribution of Equation 1 by *C*. In example, we show the resulting error distribution for the usage of cache layer $C = 3$ in Figure 4.

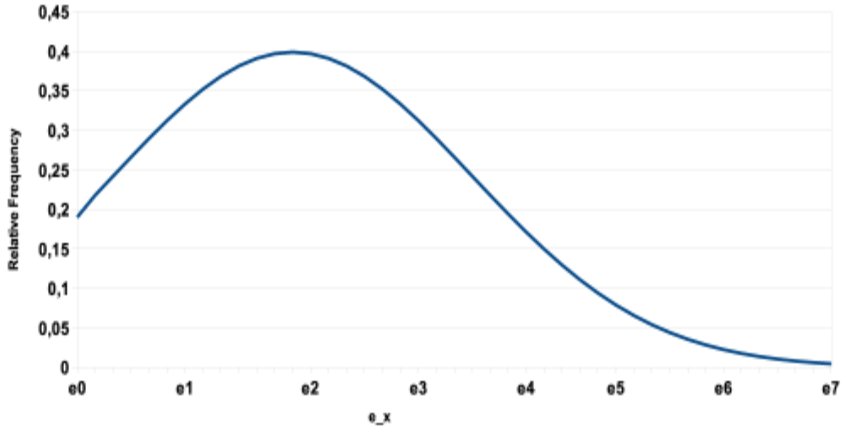

Figure 4. Error Distribution in Network for Cache Layer 3

Equation 1 and Equation 4 now form the general error distribution of **DACS** for variable cache layer *C*:

$$g(C, e_x) = \begin{cases} F\left(T^{-1}(e_x)\right) & , for\ C = 0 \\ F(C) & , for\ e_x = 0 \wedge C > 0 \\ F\left(T^{-1}(e_x) + C\right) & , for\ e_x > 0 \wedge C > 0 \end{cases} \quad (5)$$

As denoted previously, we can now retrieve the overall average error by calculating the expectancy value E of the error distribution: $E(g(C, e_x))$.

Finally, $C(\in)$ is defined as

$$C(\in) = min\left(C \middle| (\in - E(g(C, e_x))) \quad ; \forall C : E(g(C, e_x)) > \in\right) \quad (6)$$

This function retrieves the cache layer *C* whereby the overall average error of this layer ($E(g(C, e_x))$) is as close as possible to the error requirement without violating it. In **DACS** the function $C(\in)$ is solved using interval-valued approximation.

Remarks on the Distribution of Nodes

In the derivation of $C()$ we assume the knowledge of the general node distribution $F(X)$. This distribution of nodes can be known for dedicated deployments, e.g. square deployments with equidistant nodes. However, for general deployments, e.g. deployments out of air, the exact position of nodes and their relative position to each other can not be fully verified. For this purpose, we have investigated the node distribution of randomly distributed networks to find distribution classes that can be used in **DACS** . We therefore randomly placed nodes of large scale networks consisting of n nodes (n>1000) and issued an analyzation query from a gateway to retrieve the amount of nodes on each hop layer.

As a result, we retrieved that randomly deployed networks tend to be gaussian distributed. In Figure 5 we show a histogram of the distribution of randomly placed nodes based on an average of 100 evaluations for 100 nodes. The red curve denotes the density function of the gaussian distribution with a mean / median of 17 and a variance of 8. The gaussian distribution was verified running statistical verifications, e.g. the Kolmogorow-Smirnow Test [15]. The previous described localization strategy in general is independent of the actually used distribution. However, based on these statistical tests, a gaussian distribution can be assumed as a general case. Only an estimation of the network diameter has to be done before using **DACS**.

4. Evaluation

In this section we give an extended overview on evaluation results of running **DACS** in WSNs. Hereby, **DACS** has been evaluated in real sensor node deployments and simulations to test the scalability for very large deployments. The measurement data is based on real temperature measurements in Friedberg, Germany. As proposed throughout this work, the intention of

DACS is to save energy on the communication path. Hereby, it is not only important that **DACS** actually reduces the communication overhead but also that the error requirement of a given query is never violated. Therefore the evaluation covers the following most important aspects:

1) *Communication Efficiency*: The impact on the communication demand of using various error functions is shown in Section IV-A.

2) *Query per Update Trade-off* : The evaluation in Section IV-B covers aspects on when to use **DACS** in relation to the ratio between expected updates and queries.

3) *Validity and Robustness*: The error requirement of a given query has to always be guaranteed. Therefore the deviation gradiation on the update layer path that is defined by the used error function needs to be adhered. An evaluation for this aspect is given in Section IV-C.

As sensor node hardware we use Pacemate nodes [16], based on a Philips LPC 2136 Processor, and iSense core modules, based on a Jennic 32bit RISC Controller [17]. The available RAM was 96kByte shared for program and data (heap memory was ≈ 15kByte, program memory was ≈ 81kByte). We hereby point out that **DACS** is fully applicable on real sensor nodes and has been tested in an indoor application scenario as shown in Figure 6 whereby each node sends measurements continuously and one node acts as a gateway. The network consisted of up to 30 nodes that where either placed randomly or in a pairwise linear topology.

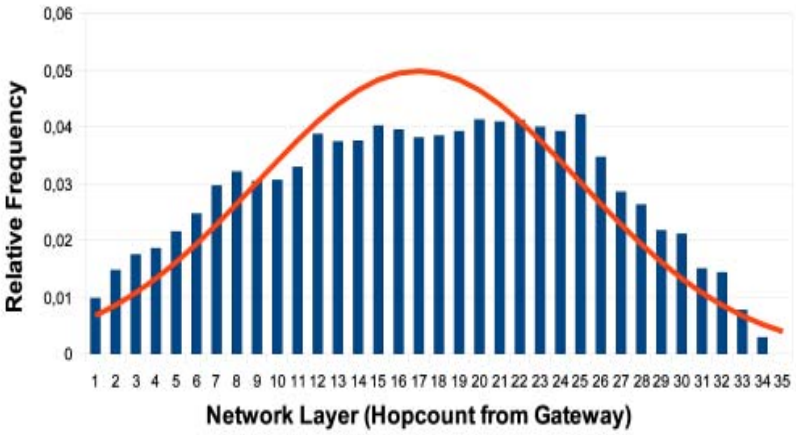

Figure 5. Statistical Node Distribution for uniform randomized Deployments

4.1 Energy and Communication Efficiency

We first test the communication efficiency of **DACS** based on two quarter temperature measurements and different error functions. The results are shown in Figure 7. Hereby, the xaxis shows the chosen linear error function for the coherence protocol and the y-axis denotes the communication demand in update messages. As a result, linearly increasing the error function significantly reduces the communication demand. Both measurements show a logarithmic decrease. Using **DACS** in the network therefore significantly reduces the update demand if the user can accept more deviation in the cache results.

4.2 Query per Update Trade-off

In the next evaluation, the usability of **DACS** in relation to the expected amount of queries was tested. As denoted previously, the usability of a caching scheme depends on the amount of unique queries that are sent in the network. Generally, caching becomes interesting if a high amount of independent queries is estimated and the update rate is lower. In this evaluation we show how this trade-off can be determined for **DACS** . Thereby, we have to compare the cache layers that are actually used. In Figure 8 we show the trade-off results for the message demand for the temperature evaluation scenario based on a network of 100 nodes with a diameter of 6 hops over 91 update cycles (one quarter). The x-axis hereby denotes the numer of queries that occur during the 91 update cycles (query per update ratio). The y-axis denotes the overall communication demand in messages. The message demand for direct querying acts as a reference, where all nodes need to be reached without using caches and hence the results need to be forwarded through the entire network. The different curves show the usage of cache layer one to six. The higher the cache layer the deeper it is in the network. As a result the intersection between the direct querying curve and the cache layer curve determines the point of inflection at which caching reduces the communication

demand. We show the inflection points of this scenario in Table I. For caches closer to the gateway the tradeoff is significantly small, e.g. for cache layer 1 0.14 queries per update, which shows the benefits of approximative data caching in WSNs.

Figure 6. **DACS** Pacemate Indoor Deployment

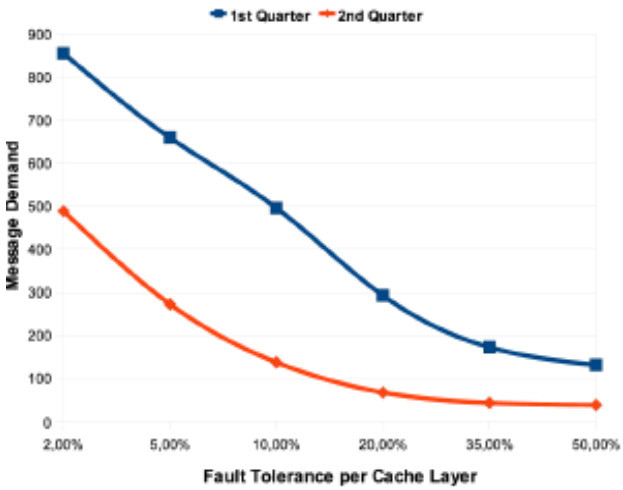

Figure 7. Message Demand for varying Error Function Update Policies

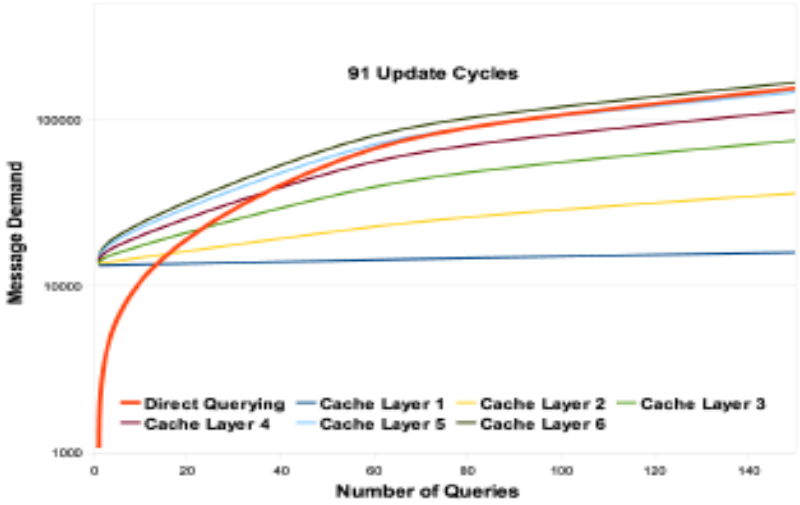

Figure 8. Update per Query Trade-Off

Cache Layer	Ratio Query per Update
1	0,14
2	0,16
3	0,23
4	0,38
5	0,98
6	36,6

Table 1. Point of Efficiency: Query Per Update

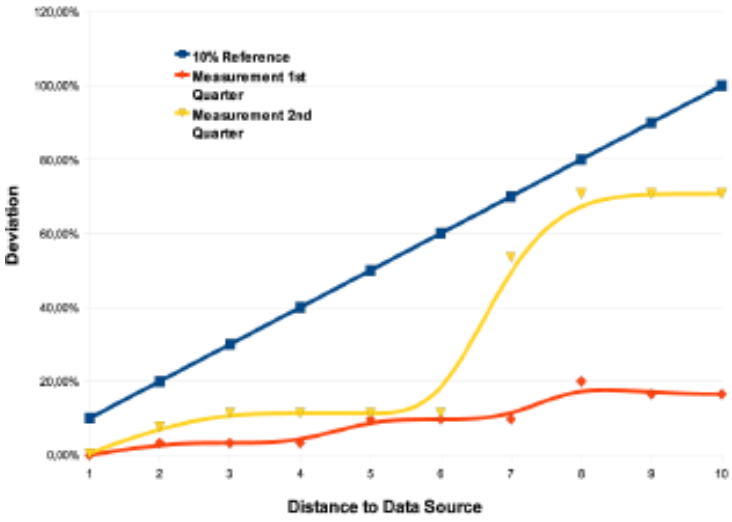

Figure 9. Validity of the Update Policy for a 10% per Hop Error Function

To verify the results, we show the corresponding experimental results for using a 5% and a 20% error function in Figure 10 whereby an average of both quarters is shown.

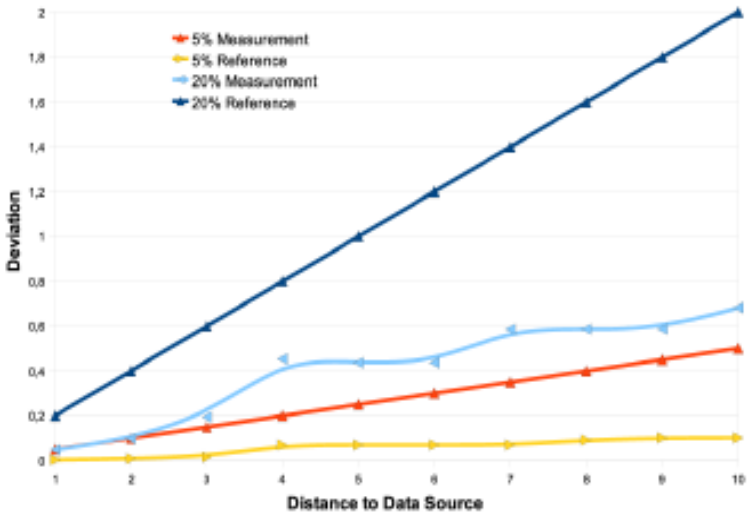

Figure 10. Validity of the Update Policy for a 5% / 20% per Hop Error Function

Again, the update policy can be guaranteed for both error functions. Hence, **DACS** provides a flexible stable update management with stable error gradiation which ensures the correctness of the cache localization process.

5. Conclusion

This paper proposes a dynamic approximative caching scheme for wireless sensor networks to optimize model driven query evaluation. An approximative update policy has been introduced to support a weak cache coherence. Based on a deviation tolerance a query is issued to the network and the corresponding caches are used adaptively. Evaluations have shown that this concept performs significantly better than traditional query evaluation when a minimal deviation in the query results can be accepted. For future work, the adaptation of transactional techniques in the cache update process is reviewed to further optimize the approximate cache coherence update policy.

References

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

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

[3] Hun Lee, M., Bok Eom, K., Joong Kang, H., Sun Shin,C., Yoe, H (2008). Design and implementation of wireless sensor network for ubiquitous glass houses, *ICIS*, V. 0, p. 397–400.

[4] Madden, S. R.. Franklin, M. J.. Hellerstein, J. M.. Hong, W. (2005). Tinydb: an acquisitional query processing system for sensor networks, *ACM Trans. Database Syst.*, 30 (1) 122–173.

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

[6] Nath, S., Gibbons, P. B.,Seshan, S., Anderson, Z. R. (2004). Synopsis diffusion for robust aggregation in sensor networks," in *Proceedings of SenSys '04*. New York, NY, USA: ACM, p. 250–262.

[7] Madden, S., Franklin, M. J., Hellerstein, J. M.,and Hong, W. (2002). Tag: a tiny aggregation service for ad-hoc sensor networks, *SIGOPS Oper. Syst. Rev.*, 36 (SI). p. 131–146.

[8] Prabh, K. S.,Abdelzaher, T. F. (2005). Energy-conserving data cache placement in sensor networks, *ACM Trans. Sen. Netw.*, 1 (2) 178–203.

[9] Chand, N. Joshi, R. C., Misra, M. (2007). Cooperative caching in mobile ad hoc networks based on data utility, *Mob. Inf. Syst.*, 3 (1) 19–37.

[10] Rahman, M. A., Hussain, S. (2007). Effective caching in wireless sensor network, *Advanced Information Networking and Applications Workshops, International Conference on*, 1, p. 43–47.

[11] Li, Y., Ramakrishna, M. V., Loke, S. W. (2007). An optimal distribution of data reduction in sensor networks with hierarchical caching, *In: EUC*, 2007, p. 598–609.

[12] Anonymous. (2006). Approximate query answering in sensor networks with hierarchically distributed caching, *Advanced Information Networking and Applications, International Conference on*, 2. p. 281–285.

[13] Jung,E.-H., Park, Y.-J. (2008). Tinyonet: A cache-based sensor network bridge enabling sensing data reusability and customized wireless sensor network services, *Sensors*, 8 (12) 7930–7950, 2008. [Online]. Available: http://www.mdpi.com/1424-8220/8/12/7930

[14] Deshpande, A.,Guestrin, C., Madden,S. R., Hellerstein, J. M.,and Hong, W. (2004). Model-driven data acquisition in sensor networks, *In*: VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases. VLDB Endowment, p. 588–599.

[15] Bronshtein, I. N., Semendyayev, K. A. *(*1997). Handbook of mathematics (3rd ed.), K. A. Kirsch, Ed. London, UK: Springer-Verlag.

[16] Lipphardt, M.H., Hellbr¨uck, D., Pfisterer, S., Ransom, Fischer, S (2007). Practical experiences on mobile inter-body-area-networking, *In*: BodyNets '07. ICST, 2007, p. 1–8.

[17] Coalesenses. Coalesenses isense core module, http://www.coalesenses.com