ZeDDS – Fault-Tolerant Data Management in Wireless Sensor Networks with Mobile Users

Jens Kamenik*
OFFIS
Escherweg 2
26121 Oldenburg, Germany
Email: jens.kamenik@offis.de

Christoph Peuser, Volker Gollücke, Daniel Lorenz,
Roland Piechocki, Merlin Wasmann, Oliver Theel
Carl von Ossietzky University of Oldenburg
26111 Oldenburg, Germany
Email: theel@informatik.uni-oldenburg.de

***Abstract*—Ubiquitous wireless sensor networks (WSNs) consist of sensor nodes which may communicate with each other via unreliable communication links. Furthermore, the sensor nodes themselves may fail. Ubiquitous WSNs may be used in application scenarios where they autonomously monitor the environment and are only sporadically visited by the mobile user for harvesting the collected sensor data. Thus, high availability of the measured data is of paramount priority. But how can the mobile user formulate this QoS requirement and how can a WSN – honoring such a QoS requirement – be efficiently implemented? We propose ZeDDS[1] a middleware and control framework for providing high available data storage in WSNs. In ZeDDS, we assume that the WSN is meant for collecting and dependably storing measured data until the mobile user contacts the WSN for data harvesting. ZeDDS enables the mobile user to explicitly specify a particular replication strategy exhibiting a certain data availability and energy consumption. At run-time, ZeDDS is appropriately configured and replicates the measured sensor data according to the replication strategy specified. We evaluate our ZeDDS implementation in terms of write operation availability measurements of a WSN consisting of TelosB sensor nodes using three different well-known replication strategies.**

***Keywords*-wireless sensor networks; distributed data storage; data replication**

I. Introduction

In application scenarios where sensor data is temporarily stored within the WSN – instead of transferring the data directly to a base station – data availability is an important factor. Depending on the application scenario of the WSN, the required level of data availability may differ. A powerful concept for increasing data availability is *data replication*. Every sensor node of a WSN that collects sensor data owns a data object. This data object may be realized by multiple copies of the sensor data located at different nodes, including the sensor node that owns the object. Such a copy is called a *replica*. Additional to its own data object and a replica belonging to it, a sensor node may host replicas of sensor data objects of other nodes of the WSN. There exist different *replication strategies* for managing the replicas of a data object. The replication strategies differ in the level of data availability they provide and the communication costs they generate.

Unfortunately, often, an increase in data availability also leads to an increase in communication costs. Therefore, the mobile user of such WSNs needs to adjust the level of data availability of the measured data to find a good trade-off between data availability and communication costs. In one of our previous works [1], we proposed a method to optimize data availability and communication cost according to the query workload of a WSN. This method can be used as a decision support for the mobile user to find a suitable replication strategy (or a combination of replication strategies) that meets the availability and energy requirements of the WSN application. To the best of our knowledge, up to now, there has been no implementation of a framework for WSNs that allows an end-user requirement-driven customization of the replication strategy – without changing the underlying implementation and that is flexible enough to support many and even completely new replication strategies. Additionally, the framework supports switching between replication strategies at run-time, for instance, in order to react to modified application requirements. In a typical ubiquitous scenario, a mobile user deploys a WSN in the environment for monitoring an environmental phenomenon (Figure 1). A sensor node of such a WSN consists of a micro-controller, a limited amount of memory, multiple sensors, energy supply, and a wireless transceiver. The wireless transceiver allows the sensor nodes to communicate with each other and with a unique mobile base station. The mobile base station, e.g., a hand-held device or notebook, is carried by the mobile user for the purpose of harvesting the sensor data of the WSN. Thus, the mobile base station is only sporadically connected to the WSN, for example, as long as the mobile user moves within the communication range. The mobile base station is the primary interface of the mobile user to the WSN. With a particular client running on it, the mobile user has the ability to configure the WSN in terms of a suitable replication strategy. The user

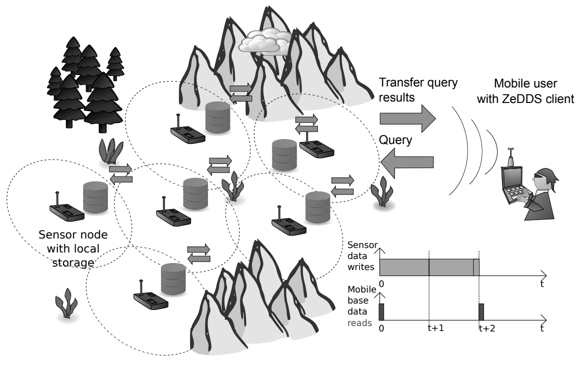


Fig. 1: Application scenario with mobile base station

* Supported by the German federal state of Lower Saxony with funds of the European Regional Development Fund (ERDF), research project CogniLog.

[1]ZeDDs is the German acronym for ”Dependable & energy-efficient data management in WSNs”

can further initiate measurements by (1) instructing the WSN what sensor data must be recorded and how long the recording should last (i.e., the user issues a corresponding query for sensor data). Then, the mobile user "disconnects," i.e., he or she leaves the communication range of the WSN. On return, he or she can (2) gather the results that were recorded during his or her absence. The WSN's task in the meantime is to dependably store all collected sensor data during that time frame and keep it "ready to be harvested" by the mobile user with high probability at any time. Thus, the data must be managed in a highly available fashion until the mobile base station reconnects. More precisely, our application scenarios exhibit the following system requirements:

1) replication strategies can be specified and even changed at run-time without altering the framework implementation, i.e., "reprogramming the whole WSN,"
2) the framework should be able to support multiple sensors per sensor node,
3) sensor data should be stored on the sensor nodes themelves,
4) sensor data can be erased after harvesting, and
5) at least one query should be allowed to be effectuated at every sensor node at any time.

In fulfillment of these requirements, we propose ZeDDS: a middleware and control framework for providing available data storage in WSNs. In ZeDDS, the WSN dependably stores measured data as long as the mobile user has not harvested it. ZeDDS enables the mobile user to specify a replication strategy that exhibits a desired, sufficiently high data availability and sufficiently low energy consumption. At run-time, ZeDDS replicates the measured sensor data according to the specified replication strategy. The evaluation of our implementation is done by measurements of the write operation availability on a WSN consisting of TelosB sensor nodes using three different replication strategies whose availabilities and message costs are well-known.

The remainder of this paper is organized as follows: In Section II, we introduce basic concepts and terminology associated with data replication and (data) replication strategies as it is used throughout the paper. In Section III, we review related work on different approaches for distributed data storage in WSNs employing data replication. In Section IV, we describe the ZeDDS architecture and explain adapted SQL command syntax and semantics used for controlling the ZeDDS framework. Furthermore, in Section V, we validate our implementation by measurements of write operation availabilities of three different replication strategies on a WSN existing of TelosB sensor nodes. Finally, Section VI concludes the paper.

II. Preliminaries

In this paper, we consider WSNs with n sensor nodes and a unique (mobile) base station. At any particular point in time, a sensor node is either in failed state (i.e., "down") or in working state (i.e., "up"). The average behavior of a sensor wrt. to up and down time periods, the so-called *node availability* p, is given by as follows:

$$p = \frac{MTTF}{MTTF + MTTR}, \tag{1}$$

where MTTF is the *mean time to failure* and MTTR is the *mean time to repair*. For simplifying purposes, we assume that the node availability of every sensor node is identical. Operations are either read or write operations. A read operation is the reading of stored sensor data from the WSN and a write operation writes measured data to the WSN data storage. Operations are initiated by a query, that specifies on which sensor node operations are executed and how long and how often they are executed. The availability of the write operation can be approximated by A_w, calculated as follows:

$$A_w = \frac{\text{number of successful write operations}}{\text{total number of write operations tried}}. \tag{2}$$

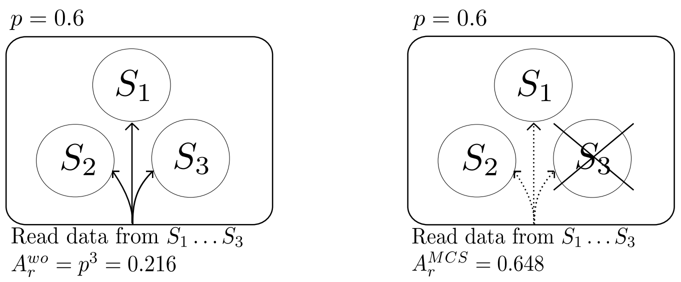
The availability of the read operation can be approximated by A_r, calculated correspondingly. For different replication strategies closed formulas for the operation availability are known (see [2]). They, in general, depend on a strategy-specific parameter set, the total number of nodes n and the node availability p

$$A = f(\textit{parameter set}, n, p). \tag{3}$$

As an example, a read operation for extracting all sensor data should be issued by a base station to a WSN consisting of $n = 3$ sensor nodes. Within the WSN, the sensor data is read and written according to the Majority Consensus Strategy (MCS) [3]. The operation availabilities are compared with the corresponding operations of a WSN not using replication. With MCS, at least two sensor nodes must be read (written) in order to guarantee consistency of the sensor data read (written). Without replication, all sensor nodes have to be read because every sensor node hosts only its local data. The availability of the read operation with MCS replication can be calculated by the following formula

$$A_r^{MCS} = \sum_{k=\lceil \frac{n+1}{2} \rceil}^{n} \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}. \tag{4}$$

For the availability of the read operation without replication A_r^{wo}, all three sensors must be available. Thus, the availability is calculated by $A_r^{wo} = p^3$. Figure 2 shows that using MCS,



(a) Sensor read without replication

(b) Sensor read with MCS replication

Fig. 2: Comparison of operation availabilities

the availability of the read operation (Figure 2b) is three times higher than without replication (Figure 2a). The reason is that using MCS, one sensor node is allowed to fail (Figure 2b) and the read operation still remains available. Without replication all sensor nodes must be available for data harvesting.

For evaluating ZeDDS, the MCS [3], the Grid Protocol [4], and the Tree Quorum Protocol (TQP) [5] are used.

The Grid Protocol arranges the sensor nodes as a (logical) grid structure, e.g., a 3x3 grid. Then, for the read operation, a complete column must be locked. For a write operation, a complete column and at least one node of every column must be locked (i.e., five nodes for a 3x3 grid).

TQP uses a tree for logically arranging the sensor nodes. For a read operation, the root node must be contacted (and locked). If this fails, then a majority of its children must be contacted. If some of the children have failed, then the majority of the children of the failed children must be contacted. This process is repeated recursively until the leaf nodes are reached. For a write operation, the root node *and* a majority of its child nodes must be contacted as well as a majority of the child nodes children. This process is also recursively repeated until the leaf nodes are reached and a majority on every level of the tree has been contacted.

Every set of nodes that, depending on the particular replication strategy used, fulfills the conditions for replica collection is called a *read quorum* or *write quorum*, respectively. Furthermore, note that closed solutions for the calculation of write and read operation availabilities of MCS, Grid and TQP exist (see, for example, [2]).

To support a variety of replication strategies, *General Structured Voting* (GSV) [6] has been introduced. With GSV, replication strategies are modeled as directed acyclic graphs. These graphs, called *voting structures*, consist of physical nodes representing a replica each and virtual nodes for grouping purposes. Each node is associated with a number of votes. With the help of votes, quorums for read and write operations can be derived. To derive a quorum from the voting structure, votes are collected from the nodes. A physical node directly provides its vote if it is up and not locked in a conflicting manner. For a virtual node, votes are first collected among its children and a vote is only provided if enough votes could be collected among them. If enough votes could be collected for the voting structure's root node, then all the replicas of the participating physical nodes form a quorum of the requested operation. Figure 3 shows a voting structure for MCS. The root

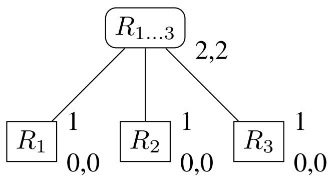


Fig. 3: A voting structure for MCS with three nodes

node requires two votes for both a read and a write quorum, while each of the physical nodes has a single vote assigned.

III. Related work

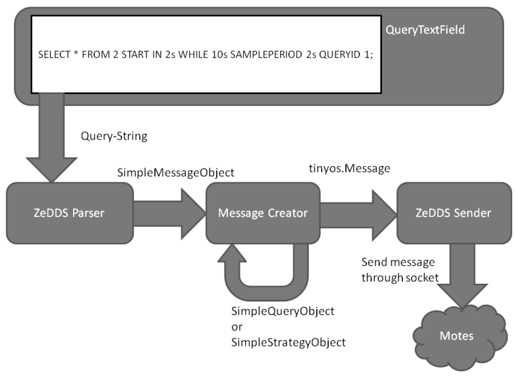
The first implementations of SQL-like query mechanisms for WSNs were Cougar [7] and TinyDB [8]. Cougar and TinDB model a WSN as a database and implement filters on the sensor nodes. This allows processing of the sensor data close to the source of the sensor data and reduces the need to transfer raw sensor data that is filtered out in a later step across the network. TinyDB implements a so-called Acquisitional Query Processor (AQP) that reduces the energy consumption of queries. For example, the AQP composes multiple queries into one query in order to save communication overhead. Alternatively, the sequence of operations having different energy costs is reordered in a way that the cheapest operation is used first to decide whether an expression is valid. For example, the expression (IF expensive sensor $>$ 100) AND (cheap sensor $<$ 50) THEN read out sensor X) is reordered such that the cheap sensor is queried first.In the middleware Sensceive [9] sensing is separated from the network management to allow the end user to focus on the sensing instead of being concerned with networking details. Sensceive also uses SQL-like query mechanisms. SwissQM [10] and Corona [11] implement a virtual machine on the sensor nodes. SwissQM provides a generic high-level framework for tasks typical in WSNs, e.g., event-processing, data pipelines and finite state automata. SQL can be uses as query language but other query languages are also possible, e.g., XQuery. Corona [11] uses an adapted SQL and runs within a Java virtual machine on the Sun SPOT platform. Corona is multi-user and multi-query capable and minimizes sensor activations in WSNs by caching sensor values and attributing them with a freshness value. Corona is able to guarantee a freshness level of read senor data. Senceive and SwissQM are available as TinyOS 2.x implementations. Corona requires a Java-VM.

Data replication strategies are well-understood in classic distributed systems. In WSN research, though, data replication is still in its infancy. For example, in [12], [13], and [14], only simple Read One Write All (ROWA) strategies are used. In [15] growth codes are used instead of replication to increase the persistence of sensed data. A Quorum-based approach for replication of service directories in WSNs is shown in [16]. A middleware comparable to ZeDDS is shown in [17]. Here, a WSN is used to store data persistently.

IV. Architecture

In this section, we describe the layout of our middleware and control framework ZeDDS.

A so-called **client application** is running on the mobile base station. It is responsible for parsing a user-initiated query, interpreting it and sending the resulting commands to the sensor nodes addressed in the query. The **client application** parser internally generates simple message objects from the query. From these objects a Message Creator generates the necessary messages (Figure 4) to control the **node application**. The ZeDDS-BNF (BNF stands for Backus Naur Form) is



```
SELECT * FROM 2 START IN 2s WHILE 10s SAMPLEPERIOD 2s QUERYID 1;
```

Fig. 4: The client application parsing process

comparable to the BNF of the Corona Project [11] or TinyDB [8] – but adapted to our requirements, i.e., to our need of specifying and controlling replication strategies. A query is specified using a variant of the classic SELECT-FROM-WHERE statement having the usual semantics. Our adaption includes commands and options for

- spec. an interval in which sensors should be sampled,
- filtering sensor data,
- killing queries,
- switching replication strategies,
- collecting statistical data, and
- receiving the results of a completed query at the mobile base station.

The most important command is STRATEGY used for replication strategy switching. Using this command and followed by the name of the chosen replication strategy as well as the keyword TO accompanied by a node id, the strategy for the particular sensor node is configured. The strategies are defined in the form of voting structures (refer to Section II), that are stored in separate text files. With this mechanism, new replication strategies can be easily integrated. The STATISTIC command, followed by a node id, allows the gathering of statistical data, e.g., the battery state of the node or the number of messages sent and received. The KILL command followed by QUERYID allows to stop a running query and deletes the related, stored data. Results of a completed query can be harvested using the GET command followed by a particular QUERYID.

A query is constructed similar to a SQL query. As an example, a query involving all sensors of node with id 2, a sampling period of 2 seconds and an overall sampling duration of 10 seconds is shown in Listing 1.

```
SELECT * FROM 2 START IN 2s WHILE 10s
SAMPLEPERIOD 2s QUERYID 1;
```

Listing 1: QuerySet 1

The *START IN* option allows an additional specification of a start delay (being 2 seconds in the example). Using *QUERYID*, a unique id is associated with the query. This id is subsequently used for managing the query, be it in the scope of a *KILL* command or for harvesting the collected data using a *GET* command. A query can additionally be attributed by a filter or an aggregation operator. Such a filtering query is shown in Listing 2.

```
SELECT * FROM 1 START IN 2s WHILE 32s
SAMPLEPERIOD 5s QUERYID 1 ADVANCED
FILTER 22.3 < temp;
```

Listing 2: QuerySet 2

Here, the keyword *ADVANCED FILTER* followed by a value and a comparison operator configures the filter to only deliver values from the temperature sensor lower than 22.3 degree. An aggregation query is shown in Listing 3.

```
SELECT temp FROM 1 START IN 2s WHILE 30s
SAMPLEPERIOD 2s QUERYID 2 ADVANCED SUM temp 2;
```

Listing 3: QuerySet 3

For this kind of query, SQL aggregation operators like sum (SUM), average (AVG), minimum (MIN) and maximum (MAX) can be chosen. Additionally, the type of the sensor must be specified as well as the size of the window, i.e., the size determines the number of sensor measurements to be considered for the aggregate operation. In our current implementation, the window size is limited to eight measurements.

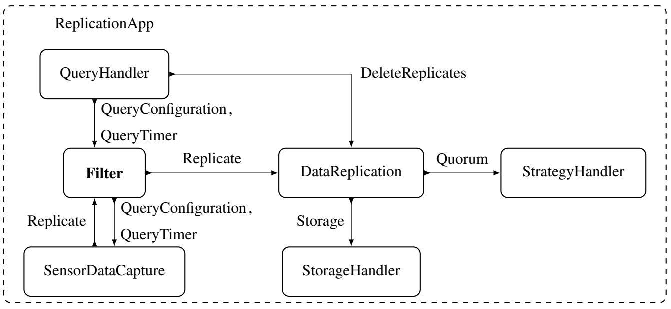


Fig. 5: Architecture of the mote application

The Sensor Node Application

The **sensor node application** is implemented as a TinyOS application and is therefore running on sensor nodes, such as TelosB. As communication layer, the (multi-hop capable) Blip [18] (Berkeley IP v1.0) stack of TinyOS 2.1.1 has been used. Due to the high complexity of the sensor node application, the application was divided into “task-oriented” TinyOS components connected via predefined interfaces. The components are the **QueryHandler**, the **Filter**, the **SensorDataCapture**, the **DataReplication**, the **StorageHandler**, and the **StrategyHandler** (see Figure 5). An additional advantage of the high modularization degree is, that we were able to develop variants of the modules with different memory footprints (note that TelosB is limited to 48KByte of Flash memory). For example, the filter component exist in two versions, one having extended functionality and a large memory footprint whereas the other one exhibits reduced functionality but a small memory footprint. Components have different tasks to perform in the different phases of ZeDDS, as decribed next.

Query Phase: If a query is sent from the client, then it will (Step 1) be received by the node’s **QueryHandler** component. This component is responsible for starting, stopping and killing queries. The **QueryHandler** passes the sampling period information to the **Filter** component via the *QueryTimer* interface, and the sensor types and the filtering options via the *QueryConfig* interface (Step 2). The **Filter** component configures its filter or aggregation functions with these options and forwards all the other information to the **SensorDataCapture** component (Step 3). This component starts the measurement with the requested sensor types and the configured sampling period information (Step 4). For each set of sensor data, it hands back the acquired measurement data to the **Filter** component via the *Replicate* interface (Step 5). After the data is filtered, it is handed on via the *Replicate* interface to the **DataReplication** component (Step 6). The **DataReplication** component assigns an index to the sensor data and requests a quorum from the **StrategyHandler** component via the *Quorum* interface (Step 7). The **StrategyHandler** then builds a quorum based on the configured strategy (Step 8) and hands it back to the **DataReplication** component (Step 9). Then, this component attempts to replicate the data (Step 10) using a two-phase commit protocol. If the attempt fails, then it returns to Step 7. If the attempt is successful, then the node application starts the next measurement by resuming Step 4. These measurement steps are repeated until the query ends. If a node is configured for replication and is part of a quorum, then its **DataReplication** component will store measurement data received from other nodes as well as local measurement data by using the *Storage* interface from the **StorageHandler** component. This component is responsible for the memory management on the sensor nodes.

Query Result Phase (Harvesting): If all sensor data measurements specified in a query have been completed, then the results can be gathered by the base station. The GET command, issued by the base station, is received by the **QueryHandler** component. The **QueryHandler** component acknowledges that the query has ended. If such an acknowledgement has been received by the client application, then a read quorum is constructed by the client application. The client application then sends a request for data to the nodes of the read quorum. These request are handled by the **StorageHandler** component which sends the collected sensor data back to the client application.

Maintenance Phase: If no query is running, then the replication strategy may be changed by sending a new voting structure to the sensor nodes **StrategyHandler** component. The voting structure specifies the quorum building process. If the **QueryHandler** receives the kill command, then it reinitializes the **DataReplication** component and terminates all running measurements in the **SensorDataCapture** component. Subsequently, the client application sends a quest to all nodes with replicated data for deleting their stored measurement data.

V. Evaluation

For the evaluation of ZeDDS, we extended the ZeDDS architecture by a software component (the so-called **StatisticHandler**) that lets the sensor nodes fail with an adjustable probability of $(1-p)$. Failures were simulated by the nodes on-the-fly meaning that a failed node did not take part in the quorum building operation for a particular number of rounds determined by the random number generator (RNG) of a sensor node. For synchronization reasons, in the scope of our experimental evaluation, we measured time in rounds. In particular, we used one round as the minimal time period for which a sensor node might fail if failed at all. The functionality was as follows.

The client application starts the round by triggering the RNG of every sensor node, except the node in charge of writing. The latter one does not fail in the scope of the evaluation. The RNG decides if the sensor node is in failed state or in working state. If the node is in working state, then it takes part in the replication process – otherwise it does not. After that, the client application submits one query with one write operation to the unique writing node. Depending on the success of the operation, the writing node increments a counter for the number of failed write operations w_{failed} that belongs to the statistical data collected. Furthermore, a counter of the total number of write operations w_{total} (statistical data) is incremented. Finally, the replicated data is deleted and this particular round is finished. After a number of those rounds, the statistical data is retrieved from the sensor nodes and the write availability is calculated according to the following formula.

$$A_w = \frac{w_{total} - w_{failed}}{w_{total}} \tag{5}$$

We assumed exponential distributed time periods between the points of repair and the points of failure. The exponential distribution was derived through inverse transformation of a normal distribution [19]. In our experiments, MTTR was set to 1 round for minimizing overall measurement time. Having a fixed MTTR, using Equation (1), different values of p were be obtained simply by varying the MTTF value.

Measurement Setup: For the measurements, we used a setup with 11 TelosB nodes and a PC with the ZeDDS client application running on it. The 11 TelosB nodes subsumed one sensor node that acquired sensor data and replicated them according to the specified replication strategy (the writing node had no local replica), nine sensor nodes that were used by the replication strategies as replica storage and one IP-Basestation that was needed as IP-Gateway for Blip. The measurements were done using multi-hop communication being enabled.

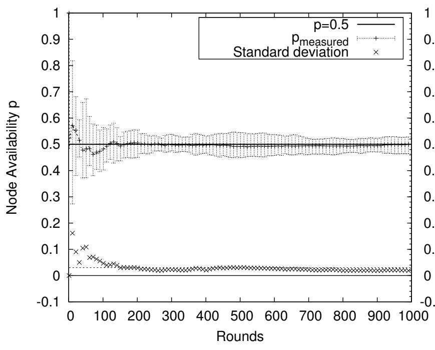


Fig. 6: Stabilization measurement of the random number generator

Before we started the measurements, the number of rounds necessary to “stabilize the RNG” was determined, i.e., the number of rounds needed to get the standard deviation of the RNG for a node availability $p = 0.5$ down to a stable level, was measured. For this, we let the RNG on one node run for 1000 rounds and repeated the measurement 10 times. Then, we calculated the mean, minimum and maximum values (Figure 6, upper graph) as well as the standard deviation (Figure 6, lower graph). It can be seen that the RNG of TinyOS needs approximately 200 rounds “to get stabilized to a standard deviation of 0.03.” For accounting the stabilization of the RNG, we let the measurements run for 300 rounds and took the write availability A_w of the last round as a stable value. The measurements were done for the strategy MCS with nine nodes, for TQP with four nodes and for Grid with, again, nine nodes. We varied the node availabilities p from 0.1 to 0.9 (in

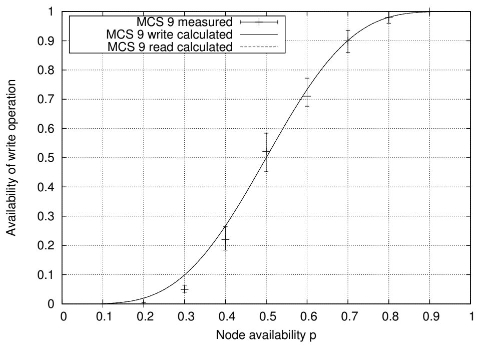


Fig. 7: Majority consensus with nine nodes

0.1 steps). For every step, the write availability was measured within 300 rounds. The measurements for every strategy were repeated five times and from the results, we calculated the mean, the minimum and the maximum value and plotted them as error bar together with the corresponding analytical values for write (and read) availabilities (as shown in Figures 7, 8 and 9). For MCS with nine nodes, the read and write availabilities are identical (see Figure 7).

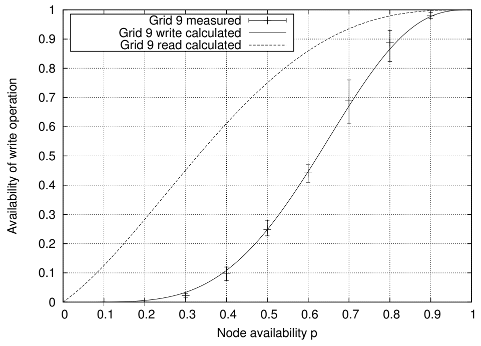


Fig. 8: GRID protocol with nine nodes

The measurement for a single run of a single replication strategy took five hours. Five repetitions accounted for 25 hours and for all three strategies an overall measurement time of 75 hours was needed. An additional measurement of the read availabilities would have doubled the measurement time – thus, we decided to restrict our analysis to the write operation in the scope of this paper. The measured write availabilities A_w for all three strategies correlate very well with the analytical results and we consider them sufficient to validate our ZeDDS framework. The deviation from the theoretical values stem from the still existing residual of the node availability at 300 rounds (Figure 6, lower graph) and the systematic error that is introduced by the fact that our time unit is a round (and thus, time must be natural multiplier thereof): RNG delivers fractional round numbers that are mapped to multiples of one round. Furthermore, occasional communication errors during the measurement may further introduce inaccuracies.

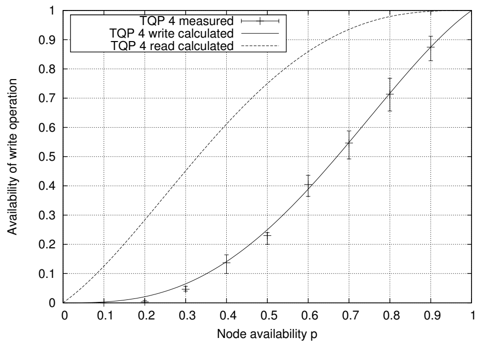


Fig. 9: TQP with four nodes

VI. Conclusion and Future Works

In this paper, we presented ZeDDS: a middleware and control framework for providing high available data storage in WSNs. In ZeDDS, a WSN has the task to dependably store the measured data as long as the mobile user has not returned to the WSN for data harvesting. ZeDDS enables the mobile user to specify a replication strategy that has a known availability and energy consumption. At run-time, ZeDDS replicates the measured sensor data according to a specified replication strategy. We described the ZeDDS architecture and the adapted SQL commands to control the ZeDDS framework. Furthermore, we successfully validated our implementation by measurements of the write operation availability on a WSN consisting of TelosB sensor nodes using the MCS, TQP and Grid Protocol replication strategies. As future work, we plan to extend the measurements to read operations. Furthermore, we plan to also measure of the overall energy consumption per replication strategy.

References

[1] J. Kamenik and O. Theel, "Optimized data-available storage for energy-limited wireless sensor networks," in *Proc. of the 6th IEEE Int. Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2010)*. Bonn, Germany: IEEE Computer Society, 2011, To apper in.

[2] H.-H. Koch, "Thesis (doctoral): Entwurf und Bewertung von Replikationsverfahren," *Technische Hochschule Darmstadt*, 1994.

[3] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases," *ACM Trans. Database Syst.*, vol. 4, no. 2, pp. 180–209, 1979.

[4] S. Y. Cheung, M. H. Ammar, and M. Ahamad, "The grid protocol: A high performance scheme for maintaining replicated data," *IEEE Trans. on Knowl. and Data Eng.*, vol. 4, no. 6, pp. 582–592, 1992.

[5] D. Agrawal and A. El Abbadi, "The generalized tree quorum protocol: an efficient approach for managing replicated data," *ACM Trans. Database Syst.*, vol. 17, no. 4, pp. 689–717, 1992.

[6] O. Theel, "General structured voting: A flexible framework for modelling cooperations," in *Proc. of the 13th Int. Conf. on Distributed Computing Systems, Pittsburgh, PA*, 1993, pp. 227–236.

[7] P. Bonnet, J. Gehrke, and P. Seshadri, "Querying the physical world," *Personal Communications, IEEE*, vol. 7, no. 5, pp. 10–15, Oct 2000.

[8] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[9] C. Hermann and W. Dargie, "Senceive: A middleware for a wireless sensor network," in *Proc. of the 22nd Int. Conf. on Advanced Information Networking and Applications*. Washington DC, USA: IEEE Computer Society, 2008, pp. 612–619.

[10] R. Müller, G. Alonso, and D. Kossmann, "Swissqm: Next generation data processing in sensor networks," in *Proc. of the 3rd Biennial Conf. on Innovative Data Systems Research*. Asilomar CA,USA: www.crdrdb.org, 2007, pp. 1–9.

[11] R. Khoury, T. Dawborn, B. Gafurov, G. Pink, E. Tse, and Q. Tse, "Corona: Energy-efficient multi-query processing in wireless sensor networks," in *DASFAA (2)*. Springer, 2010, pp. 416–419.

[12] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, and R. Govindan, "Ght: A geographic hash table for data-centric storage in sensornets," in *Proc. of the First ACM Int. Workshop on WSNs and Applications (WSNA)*. ACM, 2002, pp. 78–87.

[13] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, and L. Yin, "Data-centric storage in sensornets with ght, a geographic hash table," *Mob. Netw. Appl.*, vol. 8, no. 4, pp. 427–442, 2003.

[14] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Data-centric storage in sensornets," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 137–142, 2003.

[15] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: Maximizing sensor network data persistence," in *Proc. of SIGCOMM 2006*, Pisa, Italy, September 2006, pp. 255–266.

[16] V. Raychoudhury, "Efficient and fault tolerant servicediscovery in manet using quorum-based selective replication," in *Proc.of the 2009 IEEE Int. Conference on Pervasive Computing and Communications*. Washington DC, USA: IEEE Computer Society, 2009, pp. 1–2.

[17] J. Neumann, C. Reinke, N. Hoeller, and V. Linnemann, "Adaptive quality-aware replication in wireless sensor networks," in *Proc. of the 2009 Int. Workshop on Wireless Ad Hoc, Mesh and Sensor Networks (WAMSNET09)*, ser. Communications in Computer and Information Science (CCIS), 2009, vol. 56, pp. 413–420.

[18] S. Dawson-Haggerty, "Blip (Berkeley IP implementation for low-power networks)," (Last access 14.07.2011) 2008. [Online]. Available: http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip

[19] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling*. John Wiley & Sons, 1991.