MoMi - Model-Based Diagnosis Middleware
for Sensor Networks

Adriaan de Jong
Embedded Software group
Delft University of Technology
The Netherlands
adriaan@ch.tudelft.nl

Matthias Woehrle[*]
Computer Engineering and
Networks Lab
ETH Zurich, Switzerland
woehrle@tik.ee.ethz.ch

Koen Langendoen
Embedded Software group
Delft University of Technology
The Netherlands
k.g.langendoen@tudelft.nl

ABSTRACT

Numerous deployments of Wireless Sensor Networks (WSNs) have shown that intricate problems are introduced by deploying sensor nodes in an unknown and dynamic environment. Consequently, a vital part of a WSN deployment is the supervision of the sensor nodes and detection of abnormalities in their operation. Previous approaches have devised hand-crafted solutions to detect application-specific problems. This work proposes a generic middleware component, named MoMi, to detect such problems in a systematic way. Given a description of normal system behavior MoMi uses a Model-Based Diagnosis (MBD) framework to present the likely causes of system abnormalities to an administrator. This paper demonstrates a proof of concept implementation of MBD on sensor nodes and presents examples of its applicability for typical WSN applications.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging — *Diagnostics*

Keywords

Failure detection, WSN, MBD

1. INTRODUCTION

Wireless sensor networks have experienced numerous unsuccessful deployments [1, 5, 14]. The underlying problems are diverse and affect different parts of the system [10]: node problems concerning hardware, e.g., the packaging or the energy supply; link problems due to high variability in the wireless channel, e.g., through environmental changes; or ill-behaving protocols creating congestion or routing path oscillations. This is just a small sample of the encountered

problems. Up to now, the common approach to handle these types of problems has typically been very problem-specific: (i) Hand-crafted health-monitoring protocols [7, 8, 11] relying on expert knowledge and/or limited application scope or (ii) sending (a subset of) raw health data to the common data back-end and using standard network monitoring tools and monitoring rules for problem detection [2] or (iii) using special handheld devices for enabling interactive on-site diagnoses by experts [1, 6].

In contrast, this paper presents a generic middleware component - named MoMi - for identifying causes of failures in sensor network deployments. System health is diagnosed in a decentralized manner based on a middleware abstraction using the Model-Based Diagnosis framework [4]. Utilizing MBD avoids hand-crafted solutions for diagnoses in sensor networks. MBD uses a formal model, independent of hardware and operating system idiosyncrasies, which provides a separation of concerns. The user provides a concise model of normal behavior of system components rather than an elaborate failure model specifying how each individual failure can be traced to a fault in a given component. The MBD middleware takes the behavioral model and performs a diagnosis, i.e., infers probable causes, in a transparent yet resource-efficient manner and presents a minimal set of potential faulty components to a system administrator as soon as a problem is detected. MoMi has been implemented for TinyOS 2 and evaluated both in simulations and on a testbed verifying that MBD is a feasible approach for WSNs.

This paper provides the following contributions:
- A formulation of supervision tasks for WSN deployments using a Model-Based Diagnosis framework.
- Details on performing Model-Based Diagnosis in a decentralized manner on resource-scarce sensor nodes.
- A case study and evaluation is provided, which shows the feasibility of our approach on a 24-node testbed and in simulation, based on real deployment data.

In the following, we present a general introduction to MBD and discuss related work in WSNs. Section 3 presents diagnosis examples that are used in describing the MoMi architecture. Finally, we show a proof of concept implementation using multiple testcases, before concluding the paper.

2. BACKGROUND AND RELATED WORK
2.1 Background

Model-Based Diagnosis (MBD) is a framework for troubleshooting systems, which has its origins in the artificial intelligence community. As depicted in Figure 1, it uses formal models of a system, in particular its *components (COMP)*
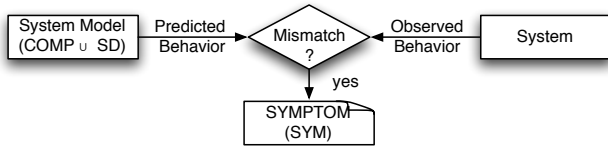
Figure 1: Model based diagnosis: Comparing observed and model predicted behavior

and a *system description (SD)* of their composition. Component models specify normal behavior, i. e., that there is no abnormality. The standard notation of $\neg AB(c), c \in COMP$ describes that a component c is functioning not abnormal, i. e., as specified. These models are applied to *observations (OBS)* of system behavior. *Predictions (PRED)* of behavior are inferred based on observations collected during runtime. When predictions mismatch with observations, a *symptom* is detected, which constitutes a failure w. r. t. the specified behavior. The MBD framework infers the set of components that potentially have induced the failure, so-called *conflicts*. A *diagnosis* is generated from a set of conflicts as the minimal set of faulty components that can explain failures. Informally, the difference between a conflict and a diagnosis is that conflicts describe a set of components for which at least one is assumed to be faulty, while a diagnosis details exactly the set of components that are all assumed to be false (broken) [4]. Formally a diagnosis is formulated as determining a minimal set of faulty components $\Delta \subset COMP$ such that the following is consistent:[1]

$$SD \cup OBS \cup \{AB(c)|c \in \Delta\} \cup \{\neg AB(c)|c \in COMP - \Delta\}$$

There has been considerable research on MBD, also in the context of distributed systems; this work is based on the foundations of MBD by Reiter [9] and de Kleer [4]. This type of MBD focuses on state-based diagnosis, i. e., it diagnoses only the current state (snapshots) and hence considers models using state invariants. The resulting models are small, which is required for using MBD on memory scarce sensor nodes (e. g., with only 4KB of RAM).

MoMi features a hybrid architecture: It utilizes lean, localized models of the system for inferring conflicts on the sensor node itself. Conflicts are sent to the sink where they are forwarded to a *diagnosis host*, which performs the diagnosis from received conflicts. As energy is constrained and communication is a significant consumer, sending all observations to a sink, i. e., a centralized approach, is prohibitive. Local limitations in storage and processing power are prohibiting completely distributed diagnosis, i. e., require using a stronger diagnosis host.

2.2 Related Work

Most related to our work in terms of distributed, in-system, on-line monitoring is the work on health monitoring of sensor network applications [7, 8, 11]. All of these works are application-specific solutions. Hence, they target a fixed set of faults and use explicit failure models and rather focus on optimization of message complexity. The most comprehensive health monitoring tool is Sympathy [8], which utilizes specific expert knowledge on failure dependencies in data collection applications to generate an elaborate decision tree for root cause analysis. In contrast, MBD on sensor net-

works targets a generic middleware component, usable in different kinds of sensor network applications and allows for formulating diverse diagnoses of failures.

Davis et al. [3] provide an overview of MBD, its applicability and differences to diagnostics, fault dictionaries, decision tress and rule-based programming. In a nutshell, when using MBD there is no need for providing specific failure models, which would be needed for generic debugging platforms, such as Wringer [12]. Rather one can focus on a model of valid behavior and let the inference be performed by the MBD framework. This renders MBD suitable for a middleware abstraction for decentralized fault analysis.

3. MBD FOR WSNS

In the following, typical examples are described, which are used as guiding examples when discussing the MoMi architecture and in the case study.

3.1 Diagnosing WSNs

Two specific diagnoses in sensor networks are detailed concerning (a) sensed phenomena and (b) node and link health. Additionally, further examples of MBD usage are presented before detailing how these models result in diagnoses in MoMi.

3.1.1 Spatial phenomenon model

A fundamental diagnosis concerns the interpretation of sensed phenomena. The interpretations are manifold, yet highly dependent on the application context. This can be functional purposes, such as detecting correlations in sensed vibrations in bridge monitoring or between temperature of mountain rock for permafrost monitoring [2]. In the following we focus on health monitoring of the sensors, in particular, on those of the SensorScope project where corrosion created several problems. These could have been detected through abnormally large differences in humidity readings between sensor nodes [1]. This diagnosis is formulated in MBD as a predicate on sensor values of neighboring nodes; these values should be approximately the same. While in general a sophisticated model could be favorable, we utilize a simple, yet effective approximation invariant. The model is specified over the environment of a node (Eq. 1a), comparing its local sensor value with the sensed value of its neighbor (Eq. 1b). The observations used in these models are the locally sensed values $sensor(x)$.

$$\forall n_i \in NEIGHBORS(node): \tag{1a}$$
$$\neg AB(node) \wedge \neg AB(n_i) \rightarrow sensor(node) \approx sensor(n_i) \tag{1b}$$

The approximation operator $\approx$ is a user-defined predicate that assesses whether two humidity sensor values are within a certain range. In the following, sensor values are defined to be about equal when they are within 20% of each other.

3.1.2 Link versus node problems

A second fundamental supervision task is to observe failing nodes. As an example, safety-critical sensor network applications, e. g., for fire detection, require the detection of failing nodes to guarantee system integrity [7]. In MoMi, a model can be formulated that relies on each node sending messages periodically. Depending on the application, this may be data traffic or network beacons. The model is also specified by considering nodes in the neighborhood (Eq. 2a).

---

[1]Refer to Reiter [9] for formal definitions and a discussion on decidability of diagnoses.

Eq. 2b formulates that there is communication with each neighbor in both direction. It uses predicates on sent and received messages of a specific neighbor during a given time window.

$$\forall n_i \in NEIGHBORS(node): \tag{2a}$$
$$\neg AB(link(node, n_i)) \wedge \neg AB(n_i) \wedge \neg AB(link(n_i, node)) \rightarrow$$
$$send(node, n_i) \rightarrow receive(node, n_i) \tag{2b}$$

When no message is received from a given node n_i, the reason could be that the link to or from n_i is broken or that n_i has crashed. However, when looking at conflicts from neighboring nodes, MBD can determine whether only single links or all links to a given node are affected. If all links are affected, MBD determines node death as the likelier cause.

Through the predicate functions, the model has an implicit notion of time. It assumes that there is a given communication window, i.e., there is a period where communication must take place. This is dependent on the application context. In our case study, where network beacons are analyzed, the temporal context is already provided by (a multiple of) the beacon interval.

3.1.3 Further examples

Note that these are merely two out of many possible examples of utilizing MBD for WSNs. As an example, models for the PermaSense deployment [2], which temporarily suffered from a high number of parent switches, can be formulated. Eq. 3 diagnoses whether the network layer is working as expected by checking that the number of parent switches within an interval stays below a threshold.

$$\neg AB(network) \rightarrow parentswitches(node) < p, p \in \mathbb{N} \tag{3}$$

Interference/congestion in the environment of a node can be detected by monitoring backoffs of all neighbors as shown in Eq. 4.

$$\neg AB(interference) \rightarrow \sum_{n_i} backoffs(n_i) < k,$$
$$k \in \mathbb{N}, n_i \in NEIGHBORS(node) \tag{4}$$

Finally, networking relies on stable links and healthy nodes as modeled before. Networking has dependencies on node, link and interference components. Hence, MBD could diagnose the underlying faults of the detected networking issues.

3.2 Basic MoMi Architecture

The basic architecture of MoMi is presented in Figure 2. The MoMi middleware consists of a local component on each individual sensor node and a diagnosis host receiving information via the sink node.

The node local part of MoMi, the *MoMi engine*, features two distinct parts. First, it provides storage for observations, predictions, and the model. Predictions are composed data types in MoMi comprising predicted values and an *environment (ENV)* of the prediction. This environment represents the components that a prediction relies upon. Models are implemented as constraints on components. Each observation, prediction and constraint features a unique identifier to minimize memory use.

Second, the inference engine generates predictions based on observations and the model. As soon as a mismatch between an observation and a prediction, i.e., a symptom, is detected, the inference engine generates a conflict. Conflicts are subsequently sent to the sink. The sink forwards them
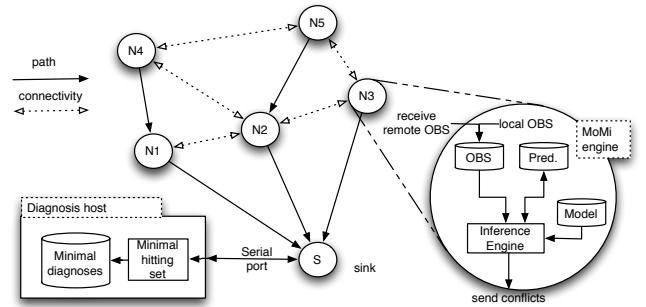

Figure 2: Overview of the MoMi architecture. Remote observations can be intercepted on their route or snooped from neighbors.

over the serial port to a diagnosis host, which performs the final diagnosis.

3.2.1 Local inference

In the following the functionality of the MoMi architecture is presented based on an example of the model of sensor correlation on the topology shown in Figure 2. In particular, we focus on node N1, which observes its two neighbors N2 and N4. N1 uses the abstract model in Eq. 1 and generates two concrete models for each of its observed neighbors as shown below.

$$\neg AB(N1) \wedge \neg AB(N2) \rightarrow sensor(N1) \approx sensor(N2)$$
$$\neg AB(N1) \wedge \neg AB(N4) \rightarrow sensor(N1) \approx sensor(N4)$$

This process is referred to as *model instantiation*. Now consider that in the example, the MoMi engine on N1 receives a local observation of a sensor value of 60 ($sensor(N1)$ = 60), which is stored in the observation database. In turn, the inference generates predictions for its two neighbors based on the models above:

$$sensor(N2) \approx 60, \{N1, N2\} \tag{5a}$$
$$sensor(N4) \approx 60, \{N1, N4\} \tag{5b}$$

As can be seen in these examples, each prediction includes an environment. For Eq. 5a, the environment denotes that only if both N1 an N2 are assumed to be working the prediction of the sensor value for N2 holds. Both predictions are stored in the prediction database of node N1.

On receiving an observation from N2 with a differing value ($sensor(N2)$ = 20) a symptom is detected with Eq. 5a, since the difference between the observation and the previous prediction is too large. The symptom shows that one of the components in the supporting environment of the conflicting predictions and observations does not hold. Hence, the MoMi engine determines the resulting conflict as the union of the environments, $\langle N1, N2 \rangle$, which signals that either node N1 or N2 is faulty, and sends it to the sink. The conflict is then archived and indexed for future reference. This is vital because each conflict is state-based, i.e., depends on the current state (snapshot) and hence may occur only temporarily. If a new observation of $N2$ purges the outdated conflict, an invalidation message is sent to the diagnosis host.

3.2.2 Centralized diagnosis

The diagnosis host computes minimal diagnoses, minimal w.r.t. to the number of faulty components, and presents

these to the system administrator. In order to determine minimal diagnoses, it needs to compute the minimum hitting set of all received conflicts. Given the fact that it is an NP-complete problem, it would be prohibitive to run this on a sensor node. Hence, a powerful diagnosis host, connected via serial port rather than the sink, is employed. For convenience, the final diagnoses are output in XML and can be visualized in a GUI.

The inference benefits from additional observations. Given the example above, an observation $sensor(N4) = 60$ on N1 refines the final diagnosis by generating a second conflict $\langle N2, N4 \rangle$. Based on these given conflicts $\langle N1, N2 \rangle$ and $\langle N2, N4 \rangle$, the diagnosis host generates a minimal diagnosis of $[N1, N4], [N2]$ indicating that either both the sensors on node N1 and N4 are faulty or merely the sensor on node N2. In turn it presents $[N2]$ as the primary diagnosis based on its cardinality i.e., based on the higher likelihood that fewer components fail concurrently.

3.3 MoMi Engine in Detail

The MoMi engine follows the original idea of de Kleer for diagnosing multiple faults [4]. A fundamental difference due to the distributed nature of the diagnosis is that the MoMi engine sends raw conflicts of individual models to the sink. There is no further dependency-tracking performed between different conflicts on the sensor node. Since global consistency is determined on the diagnosis host, MoMi saves considerable computational and storage overhead for a very light increase in traffic load, as conflicts are only sent when symptoms are detected.

Additionally, the temporal nature of the diagnosis requires (frequent) updates to facts, i.e., observations. When observations change, predictions have to be re-computed and symptoms may disappear. This requires re-running the local inference in order to maintain *local belief*: predictions, symptoms and conflicts. MoMi currently deals with changing observations by purging the previous observation and any inferred prediction or conflict. Since, observations may stay (nearly) constant over time, MoMi features a threshold for differences in consecutive observation values. The threshold is a user-defined parameter of MoMi, which trades off accuracy versus update frequency. Only if it is exceeded, the new observation is processed, while the previous one is removed.

3.3.1 Inference engine

The MoMi node engine features an inference engine, which uses a recursive algorithm for generating predictions, maintaining local belief and detecting symptoms and conflicts. For a new observation all constraints of a model are checked whether they include the given observation and hence may induce new predictions. When computing a new prediction, it is checked against all existing observations and predictions in the database. This results in one of the following cases:

1. The prediction mismatches a given observation or previous prediction of another model. Hence, a symptom is detected. The environments of the mismatched predictions and observations describe the conflict. The conflict is archived and sent to the sink. No further propagation of the prediction is required: Any additional symptom results in conflicts that are supersets of the found conflict. Since MBD is interested in minimal conflicts, these additional conflicts can be ignored.

2. The prediction matches a given observation or prediction. In turn, when one of the environments is a subset of the other, only the one with the smaller environment is maintained. When the new prediction is maintained, it requires a recursive call to propagate the change.

3. Similarly, if no prediction or observation of the same type is found in the local database, a recursive call is required as well to further process the new prediction.

3.3.2 Implementing additional models

MoMi facilitates integrating novel models. Implementing a new model requires:

- Allocation of a unique identifier for each new observation type.
- An indication when an observation is sufficiently different to a previous observation to re-start the inference with the new value.
- A function that computes a prediction given some observations.
- A mismatch function, which determines whether a prediction should create a symptom given some observations or predictions from other models.

As an example, for our environment sensor model (cf. Eq. 1), a significant difference in observations is a change by 5% in value of sensor readings. The prediction function merely assigns the locally read value to all its observed neighbors. The mismatch function checks whether the difference between sensor values is larger than 20%.

4. PROOF OF CONCEPT

This section presents an proof of concept implementation on the TinyOS operating system diagnosing on a typical sensor network application: Data collection from a set of distributed sensors, e.g., as used for environmental monitoring. In particular, we use the MultihopOscilloscope application based on the Collection Tree Protocol (CTP). We performed experiments with different diagnoses formulated in MoMi (a) in the TinyOS simulator TOSSIM, and (b) on a 24-node testbed of TNOdes [5].

The MoMi engine has about 1700 lines of NesC code (LoC) in 6 components. Given the MoMi engine, the two models introduced in Sec. 3.1.1 and 3.1.2, which are evaluated in dedicated testcases, each require approximately 100 LoC, including data structures and glue code.

On the host node attached to the sink, MoMi features the diagnosis host, some 2000 lines of Java code, including a graphical user interface for on- and off-line inspection of diagnoses. Diagnoses are stored as XML documents facilitating integration with the sensor network back-end.

4.1 Integration

Using MoMi in a sensor network application should incur minimal overhead. Hence it is vital to integrate it into the application and reuse given software components when possible. In particular, MoMi requires two communication mechanisms from an application:

- Snooping on neighborhood traffic for collecting remote observations.
- Sending conflict packets to the central sink.

Additionally, for neighborhood-dependent models and model instantiation, MoMi requires information about its neighbors, which is only available at runtime. All of these mechanisms are typically provided by the communication stack.

	Model	OBS/PRED	Total
Sensor	15	54	69
Link/node	5	29	34
1 neighbor	20	83	103
2 neighbors	40	146	186
8 neighbors	160	641	801

Table 1: MoMi RAM requirements in bytes for observation and prediction databases and the models.

In our proof of concept, MoMi is integrated with CTP; it has access to the neighbor table of CTP to determine its neighbors, and uses CTP's snoop and intercept methods for overhearing measured sensor data in the neighborhood. Additionally, MoMi uses CTP to route packets (best-effort) to the sink.

4.2 Memory Requirements

One of the biggest challenges for using MBD is memory overhead on the sensor node. The MoMi architecture itself is very lean and can be easily integrated with a given sensor network application. Hence, the memory overhead is determined by the complexity of the models. This complexity has different sources:

- The complexity of the model itself.
- The number of observations required.
- The number of model instantiations (per neighbor).

Table 1 presents the memory requirements for the presented models. The employed models can be formulated in a simple manner as can be seen in the first column. As shown in the second column, the models also rely on a small number of observations. As such, the memory overhead can be kept low, i.e., for both models and a single neighboring node just 103 bytes are needed. The critical issue for memory conservation is the model instantiation for neighboring nodes. RAM requirements increase about linearly with the number of neighbor models as can be seen in the bottom rows of Table 1. Observations/predictions scale sub-linearly, since they have an initial overhead for identifier databases and depend on the number of observations to store. Hence, instead of allocating a model for all neighbors, as specified in the original models, it is advisory to select only a subset of neighbors to be observed. In the following, only two neighbors are used in MoMi to verify that diagnosis is satisfactory even when faced with severe memory restrictions.

4.3 Testcases

Based on the previously presented models, the use of MoMi for detecting faults is demonstrated. First, diagnosis latency is investigated based on the link versus node problem model (cf. Sec. 3.1.2). Second, we show the use of MBD to find sensor-related problems on real deployment data. In all of our setups, topology control restricts the number of neighbors to two.

4.3.1 Node crashes in simulation and testbed

As a first testcase, node crashes are randomly injected into the application. Figure 3 shows the detection latency in three different simulations topologies in TOSSIM, differing in node density and environmental noise, and measured testbed results. Each scenario consists of ten runs of one hour. In the experiments CTP's adaptive beacon interval is upper bounded by 100s. Hence, the communication window
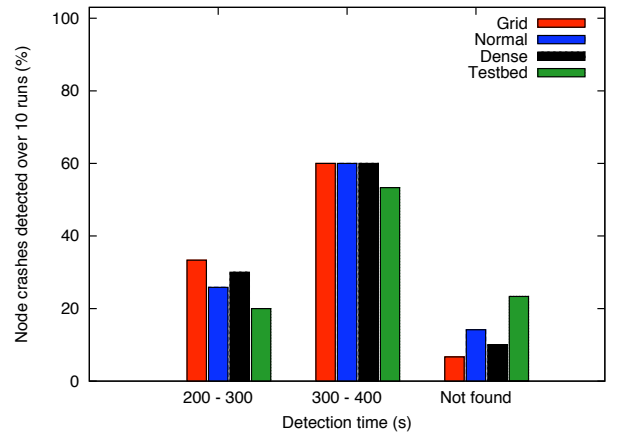

Figure 3: Latency of node crash detection in different simulation topologies and on the testbed. The are no diagnoses with the detection time being smaller than 200s or larger than 400s.

is set to 200s. The induced traffic is low; in all simulations the number of conflicts to be sent is on average 5.4 messages per induced failure.

As can be seen in Figure 3, in all tests scenarios MoMi either detects node crashes within 200s-400s, i.e., in the same or the consecutive observation window or they are not found at all. The detection latency is mainly determined by the length of the observation window, since routing of the messages takes only a minimal fraction. Undetected node crashes are false negatives; their primary cause is that MoMi uses the CTP routing protocol, i.e., conflicts are susceptible to packet loss as any routed message. Losing conflicts results in undetected defects since:

- A node is only observed by two neighbors and hence a node death creates two conflicts on each of them. If one conflict is lost, MBD assumes that the other conflict is due to a bad link rather than a node death.
- A conflict is only sent once. Since the state of a crashed node does not change, i.e., it remains dead, there are no newly generated conflicts that would be sent.

Hence, while topology control (i.e., using two neighbors) greatly reduces memory consumption, it reduces the fidelity of the diagnoses in cases of packet loss. Additionally, if diagnosis is critical further mechanisms such as end-to-end acknowledgments have to be considered for sending conflicts or a separate network stack as in SMNS [13]. Only in the noisy scenario, MoMi suffered from two false positives due to excessive packet drop. This cannot be accounted for but is rather a symptom of another fault: excessive contention.

4.3.2 Sensor failures

The second testcase is based on five days of humidity sensor data from the SensorScope deployment from the Génépi deployment [1]. In particular, the deployment faced corrupted measurements due to condensation of the connectors. We utilize the data of four humidity sensors to generate a (re-)simulation of a four-node (single-hop) network in TOSSIM. Figure 4 shows the sensor readings over the interesting fail period along with the diagnoses of MoMi. On the top, the readings of four sensors are shown. The corruption of the data at the 50th hour in the plot (in the morning of October 6th, 2007) can be clearly seen as the different
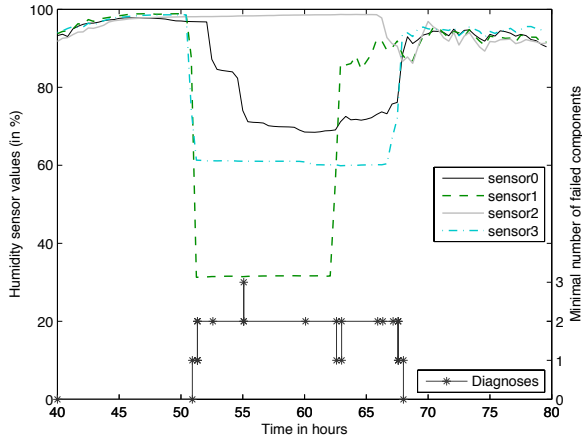
Figure 4: Humidity sensor readings and MoMi diagnoses using the Génépi deployment data.

sensors diverge substantially. The step function on the bottom indicates the minimum number of components, MoMi assumes to be defective.

As soon as the readings diverge, diagnoses are determined. This is a step-wise process. First, MoMi finds a single problem, i. e., the step function has a value of 1, between sensor0 and sensor3 and thus reports the diagnosis: $[sensor0]$, $[sensor3]$, i. e., a single component is assumed to be defective. However at around $55h$ it detects a mismatch between all sensors and as a consequence the minimal number of components being wrong is 3.

At the next step however, sensor0 changes to a value close to sensor3. Hence the diagnosis changes to a steady state of minimally 2 assumed broken components. This minimal case is that both sensor0 and sensor3 are working correctly and the other two are faulty, i. e., the minimal diagnosis that MoMi determines is: $[sensor1, sensor2]$.

Note that MoMi easily handles the dynamic changes in sensor values. At the end of Figure 4 all sensors are working properly again and the diagnoses shows that there is no assumed fault. Hence, MoMi can be used to detect sensor-related problems from multiple sensor nodes.

5. CONCLUSION

Supervision of a sensor network is a crucial part of a deployment. This work demonstrates a framework for allowing the formulation of supervision tasks (i) without incurring a large communication overhead and (ii) suitable for resource constrained sensor nodes. It presents the MoMi architecture of a middleware component for performing Model-Based Diagnoses in a sensor network in a decentralized manner. Local models use neighborhood observations to infer local conflicts and send them to a centralized diagnosis host, which performs a global diagnosis.

We detailed on the models that MoMi supports and how typical diagnosis can be formulated. A proof of concept implementation demonstrated in different testcases that MoMi diagnoses in a generic, transparent and resource efficient manner. Simulation as well as testbed results showed that timely delivery of diagnoses ($200s - 400s$) is achievable using the standard network stack (CTP) of the application.

In future work, we will look at a trade-off of more elab-

orate model formulations, while maintaining low memory requirements. In particular this concerns the modeling of temporal and dynamic context of the system. Due to the nature of invariants, sudden changes (observations) may induce spurious diagnoses as new local values are compared with stale neighborhood data. Temporal context is only implicitly handled through observations, e. g., network beacons. Ultimately, a language for a concise representation of models, which generates platform-specific code is desirable. However, observations are application- and sensor node-specific and typically need to be carefully integrated rather than synthesized.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] G. Barrenetxea et al. The hitchhiker's guide to successful wireless sensor network deployments. In *Proc. 6th ACM Conf. Embedded Networked Sensor Systems (SenSys '08)*, 2008.

[2] J. Beutel et al. PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes. In *ACM/IEEE Int'l Conf. on Information Processing in Sensor Networks (IPSN '09)*, pages 265–276, 2009.

[3] R. Davis and W. Hamscher. Model-based reasoning: troubleshooting. In *Readings in model-based diagnosis*. 1992.

[4] J. de Kleer and B. C. Williams. Diagnosing multiple faults. In *Readings in model-based diagnosis*. 1992.

[5] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *Proc. 20th Int'l Parallel and Distributed Processing Symposium (IPDPS 2006)*, pages 8–15, 2006.

[6] H. Liu, L. Selavo, and J. Stankovic. SeeDTV: deployment-time validation for wireless sensor networks. In *Proc. 4th IEEE Workshop on Embedded Networked Sensors (EmNetS-IV)*, pages 23–27, 2007.

[7] A. Meier et al. DiMo: Distributed node monitoring in wsns. In *Proc. 11th ACM In'l Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems MSWiM 2008*, pages 117–121, 2008.

[8] N. Ramanathan et al. Sympathy for the sensor network debugger. In *Proc. 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys '05)*, pages 255–267, 2005.

[9] R. Reiter. A theory of diagnosis from first principles. In *Readings in model-based diagnosis*. 1992.

[10] M. Ringwald and K. Römer. Deployment of sensor networks: Problems and passive inspection. In *Proc. of the 5th Workshop on Intelligent Solutions in Embedded Systems (WISES '07)*, 2007.

[11] S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. In *Proc. 3rd IEEE Communications Society Conf. Sensor, Mesh and Ad Hoc Communications and Networks (SECON '06)*, 2006.

[12] A. Tavakoli et al. The case for predicate-oriented debugging of sensornets. In *Proc. of the 5th Workshop on Hot Topics in Embedded Networked Sensors (HotEmNets '08)*, 2008.

[13] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *2nd European Workshop on Wireless Sensor Networks (EWSN '05)*.

[14] M. Wachs et al. Visibility: A new metric for protocol design. In *Proc. 5th ACM Conf. Embedded Networked Sensor Systems (SenSys '07)*, 2007.