

DiMo: Distributed Node Monitoring in Wireless Sensor Networks

Andreas Meier[†], Mehul Motani^{*}, Hu Siquan^{*}, and Simon Künzli[‡]

[†]Computer Engineering and Networks Lab, ETH Zurich, Switzerland

^{*}Electrical & Computer Engineering, National University of Singapore, Singapore

[‡]Siemens Building Technologies, Zug, Switzerland

ABSTRACT

Safety-critical wireless sensor networks, such as a distributed fire- or burglar-alarm system, require that all sensor nodes are up and functional. If an event is triggered on a node, this information must be forwarded immediately to the sink, without setting up a route on demand or having to find an alternate route in case of a node or link failure. Therefore, failures of nodes must be known at all times and in case of a detected failure, an immediate notification must be sent to the network operator. There is usually a bounded time limit, e.g., five minutes, for the system to report network or node failure. This paper presents DiMo, a distributed and scalable solution for monitoring the nodes and the topology, along with a redundant topology for increased robustness. Compared to existing solutions, which traditionally assume a continuous data-flow from all nodes in the network, DiMo observes the nodes and the topology locally. DiMo only reports to the sink if a node is potentially failed, which greatly reduces the message overhead and energy consumption. DiMo timely reports failed nodes and minimizes the false-positive rate and energy consumption compared with other prominent solutions for node monitoring.

Categories and Subject Descriptors

C.2.2 [Network Protocols]: Wireless Sensor Network

General Terms

Algorithms, Design, Reliability, Performance

Keywords

Low power, Node monitoring, Topology monitoring, WSN

1. INTRODUCTION

Driven by recent advances in low power platforms and protocols, wireless sensor networks are being deployed today to monitor the environment from wildlife habitats [1]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MSWiM'08, October 27–31, 2008, Vancouver, BC, Canada.
Copyright 2008 ACM 978-1-60558-235-1/08/10 ...\$5.00.

to mission-critical fire-alarm systems [5]. There are, however, still some obstacles in the way for mass application of wireless sensor networks. One of the key challenges is the management of the wireless sensor network itself. Without a practical management system, WSN maintenance will be very difficult for network administrators. Furthermore, without a solid management plan, WSNs are not likely to be accepted by industrial users.

One of the key points in the management of a WSN is the health status monitoring of the network itself. Node failures should be captured by the system and reported to administrators within a given delay constraint. Due to the resource constraints of WSN nodes, traditional network management protocols such as SNMP adopted by TCP/IP networks are not suitable for sensor networks. In this paper, we consider a light-weight network management approach tailored specifically for WSNs and their unique constraints.

Currently, WSN deployments can be categorized by their application scenario: data-gathering applications and event-detection applications. For data-gathering systems, health status monitoring is quite straight forward. Monitoring information can be forwarded to the sink by specific health status packets or embedded in the regular data packets. Administrators can usually diagnose the network with a helper program. NUCLEUS [6] is one of the network management systems for data-gathering application of WSN. Since event-detection deployments do not have regular traffic to send to the sink, the solutions for data-gathering deployments are not suitable. In this case, health status monitoring can be quite challenging and has not been discussed explicitly in the literature.

In an event-detection WSN, there is no periodic data transfer, i.e., nodes maintain radio silence until there is an event to report. While this is energy efficient, it does mean that there is no possibility for the sink to decide whether the network is still up and running (and waiting for an event to be detected) or if some nodes in the network have failed and are therefore silent. Furthermore, for certain military applications or safety-critical systems, the specifications may include a hard time constraint for accomplishing the node health status monitoring task.

In an event-detection WSN, the system maintains a network topology that allows for forwarding of data to a sink in the case of an event. Even though there is no regular data transfer in the network, the network should always be ready to forward a message to the sink immediately whenever necessary. It is this urgency of data forwarding that makes it undesirable to set up a routing table and neighbor

list after the event has been detected. The lack of regular data transfer in the network also leads to difficulty in detecting bad quality links, making it challenging to establish and maintain a stable robust network topology.

While we have mentioned event-detection WSNs in general, we accentuate that the distributed node monitoring problem we are considering is inspired by a real-world application: a distributed indoor wireless alarm system which includes a sensor for detection of a specific alarm such as fire (as studied in [5]). To illustrate the reporting requirements of such a system, we point out that regulatory specifications require a fire to be reported to the control station within 10 seconds and a node failure to be reported within 5 minutes [9]. This highlights the importance of the node-monitoring problem.

In this paper, we present a solution for distributed node monitoring called DiMo, which consists of two functions: (i) Network topology maintenance, introduced in Section 2, and (ii) Node health status monitoring, introduced in Section 3. We compare DiMo to existing state-of-the-art node monitoring solutions and evaluate DiMo via simulations in Section 4.

1.1 Design Goals

DiMo is developed based on the following design goals:

- In safety critical event monitoring systems, the status of the nodes needs to be monitored continuously, allowing the detection and reporting of a failed node within a certain *failure detection time* T_D , e.g., $T_D = 5$ min.
- If a node is reported failed, a costly on-site inspection is required. This makes it of paramount interest to decrease the false-positive rate, i.e., wrongly assuming a node to have failed.
- In the case of an event, the latency in forwarding the information to the sink is crucial, leaving no time to set up a route on demand. We require the system to maintain a topology at all times. In order to be robust against possible link failures, the topology needs to provide redundancy.
- To increase efficiency and minimize energy consumption, the two tasks of topology maintenance (in particular monitoring of the links) and node monitoring should be combined.
- Maximizing lifetime of the network does not necessarily translate to minimizing the average energy consumption in the network, but rather minimizing the energy consumption of the node with the maximal load in the network. In particular, the monitoring should not significantly increase the load towards the sink.
- We assume that the event detection WSN has no regular data traffic, with possibly no messages for days, weeks or even months. Hence we do not attempt to optimize routing or load balancing for regular data. We also note that approaches like estimating links' performance based on the ongoing data flow are not possible and do not take them into account.
- Wireless communications in sensor networks (especially indoor deployments) is known for its erratic behavior [2, 8], likely due to multi-path fading. We assume such an environment with unreliable and unpredictable communication links, and argue that message losses must be taken into account.

1.2 Related Work

Nithya et al. discuss Sympathy in [3], a tool for detecting and debugging failures in pre- and post-deployment sensor networks, especially designed for data gathering applications. The nodes send periodic heartbeats to the sink that combines this information with passively gathered data to detect failures. For the failure detection, the sink requires receiving at least one heartbeat from the node every so called *sweep interval*, i.e., its lacking indicates a node failure. Direct-Heartbeat performs poorly in practice without adaptation to wireless packet losses. To meet a desired false positive rate, the rate of heartbeats has to be increased also increasing the communication cost. NUCLEUS [6] follows a very similar approach to Sympathy, providing a management system to monitor the health status of data-gathering applications.

Rost et al. propose with Memento a failure detection system that also requires nodes to periodically send heartbeats to the so called observer node. Those heartbeats are not directly forwarded to the sink node, but are aggregated in form of a bitmask (i.e., bitwise OR operation). The observer node is sweeping its bitmask every sweep interval and will forward the bitmask with the node missing during the next sweep interval if the node fails sending a heartbeat in between. Hence the information of the missing node is disseminated every sweep interval by one hop, eventually arriving at the sink. Memento is not making use of acknowledgements and proactively sends multiple heartbeats every sweep interval, whereas this number is estimated based on the link's estimated worst-case performance and the targeted false positive rate. Hence Memento and Sympathy do both send several messages every sweep interval, most of them being redundant.

In [5], Strasser et al. propose a ring based (hop count) gossiping scheme that provides a latency bound for detecting failed nodes. The approach is based on a bitmask aggregation, being filled ring by ring based on a tight schedule requiring a global clock. Due to the tight schedule, retransmissions are limited and contention/collisions likely, increasing the number of false positives. The approach is similar to Memento [4], i.e., it does not scale, but provides latency bounds and uses the benefits of acknowledgements on the link layer.

2. TOPOLOGY MAINTENANCE

Forwarding a detected event without any delay requires maintaining a *redundant topology* that is robust against link failures. The characteristics of such a redundant topology are discussed subsequently.

The topology is based on so called *relay nodes*, a *neighbor* that can provide one or more routes towards the sink with a smaller *cost metric* than the node itself has. Loops are inherently ruled out if packets are always forwarded to relay nodes. For instance, in a simple tree topology, the parent is the relay node and the cost metric is the hop count.

In order to provide redundancy, every node is connected with at least two relay nodes, and is called *redundantly connected*. Two neighboring nodes can be redundantly connected by being each others relay, although having the same cost metric, only if they are both connected to the sink. This exception allows the nodes neighboring the sink to be redundantly connected and avoids having a link to the sink

as a single point of failure. In a (*redundantly*) *connected network*, all deployed nodes are (redundantly) connected.

A node's *level* \mathcal{L} represents the minimal hop count to the sink according to the level of its relay nodes; i.e., the relay with the least hop count plus one. The level is infinity if the node is not connected. The *maximal hop count* \mathcal{H} to the sink represents the longest path to the sink, i.e., if at every hop the relay node with the highest maximal hop count is chosen. If the node is redundantly connected, the node's \mathcal{H} is the maximum hop count in the set of its relays plus one, if not, the maximal hop count is infinity. If and only if all nodes in the network have a finite maximal hop count, the network is redundantly connected.

The topology management function aims to maintain a redundantly connected network whenever possible. This might not be possible for sparsely connected networks, where some nodes might only have one neighbor and therefore cannot be redundantly connected by definition. Sometimes it would be possible to find alternative paths with a higher cost metric, which in turn would largely increase the overhead for topology maintenance (e.g., for avoiding loops).

For the cost metric, the tuple $(\mathcal{L}, \mathcal{H})$ is used. A node A has the smaller cost metric than node B if

$$\mathcal{L}_A < \mathcal{L}_B \vee (\mathcal{L}_A = \mathcal{L}_B \wedge \mathcal{H}_A < \mathcal{H}_B). \quad (1)$$

During the operation of the network, DiMo continuously monitors the links (as described in Section 3), which allows the detection of degrading links and allows triggering topology adaptation. Due to DiMo's redundant structure, the node is still connected to the network, during this neighbor search, and hence in the case of an event, can forward the message without delay.

3. MONITORING ALGORITHM

This section describes the main contribution of this paper, a distributed algorithm for topology, link and node monitoring. From the underlying MAC protocol, it is required that an acknowledged message transfer is supported.

3.1 Algorithm

A monitoring algorithm is required to detect failed nodes within a given *failure detection time* T_D (e.g., $T_D = 5$ min). A node failure can occur for example due to hardware failures, software errors or because a node runs out of energy. Furthermore, an operational node that gets disconnected from the network is also considered as failed.

The monitoring is done by so called *observer* nodes that monitor whether the target node has *checked in* by sending a *heartbeat* within a certain *monitoring time*. If not, the observer sends a *node missing* message to the sink. The target node is monitored by one observer at any time. If there are multiple observer nodes available, they alternate amongst themselves. For instance, if there are three observers, each one observes the target node every third monitoring time. The observer node should not only check for the liveliness of the nodes, but also for the links that are being used for sending data packets to the sink in case of a detected event. These two tasks are combined by selecting the relay nodes as observers, greatly reducing the network load and maximizing the network lifetime. In order to ensure that all nodes are up and running, every node is observed at all times.

The specified failure detection time T_D is an upper bound for the *monitoring interval* T_M , i.e., the interval within

which the node has to send a heartbeat. Since failure detection time is measured at the sink, the detection of a missing node at the relay needs to be forwarded, resulting in an additional maximal delay T_L . Furthermore, the heartbeat can be delayed as well, either by message collisions or link failures. Hence the node should send the heartbeat before the relay's monitoring timer expires and leave room for retries and clock drift within the time window T_R . So the monitoring interval has to be set to

$$T_M \leq T_D - T_L - T_R \quad (2)$$

and the node has to ensure that it is being monitored every T_M by one of its observers.

The schedule of reporting to an observer is only defined for the next monitoring time for each observer. Whenever the node checks in, the next monitoring time is announced with the same message. So for every heartbeat sent, the old monitoring timer at the observer can be cancelled and a new timer can be set according to the new time.

Whenever, a node is newly observed or not being observed by a particular observer, this is indicated to the sink. Hence the sink is always aware of which nodes are being observed in the network, and therefore always knows which nodes are up and running. This registration scheme at the sink is an optional feature of DiMo and depends on the user's requirements.

3.2 Packet Loss

Wireless communication always has to account for possible message losses. Sudden changes in the link quality are always possible and even total link failures in the order of a few seconds are not uncommon [2]. So the time T_R for sending retries should be sufficiently long to cover such blanks. Though unlikely, it is possible that even after a duration of T_R , the heartbeat could not have been successfully forwarded to the observer and thus was not acknowledged, in spite of multiple retries.

The node has to assume that it will be reported missing at the sink, despite the fact it is still up and running. Should the node be redundantly connected, a *recovery message* is sent to the sink via another relay announcing being still alive. The sink receiving a recovery message and a node-missing message concerning the same node can neglect these messages as they cancel each other out. This recovery scheme is optional, but minimizes the false positives by orders of magnitudes as shown in Section 4.

3.3 Topology Changes

In the case of a new relay being announced from the topology management, a heartbeat is sent to the new relay, marking it as an observer node. On the other hand, if a deprecated relay is announced, this relay might still be acting as an observer, and the node has to check in as scheduled. However, no new monitor time is announced with the heartbeat, which will release the deprecated relay of being an observer.

3.4 Queuing Policy

A monitoring buffer exclusively used for monitoring messages is introduced, having the messages queued according to a priority level, in particular node-missing messages first. Since the MAC protocol and routing engine usually have a queuing buffer also, it must be ensured that only one single monitoring message is being handled by the lower layers at

the time. Only if an ACK is received, the monitoring message can be removed from the queue (if a NACK is received, the message remains). DiMo only prioritizes between the different types of monitoring messages and does not require prioritized access to data traffic.

4. EVALUATION

In literature, there are very few existing solutions for monitoring the health of the wireless sensor network deployment itself. DiMo is the first sensor network monitoring solution specifically designed for event detection applications. However, the two prominent solutions of Sympathy [3] and Memento [4] for monitoring general WSNs can also be tailored for event gathering applications. We compare the three approaches by looking at the rate at which they generate false positives, i.e., wrongly inferring that a live node has failed. False positives tell us something about the monitoring protocol since they normally result from packet losses during monitoring. It is crucial to prevent false positives since for every node that is reported missing, a costly on-site inspection is required.

DiMo uses the relay nodes for observation. Hence a possible event message and the regular heartbeats both use the same path, except that the latter is a one hop message only. The false positive probability thus determines the reliability of forwarding an event.

We point out that there are other performance metrics which might be of interest for evaluation. In addition to false positives, we have looked at latency, message overhead, and energy consumption. We present the evaluation of false positives below.

4.1 Analysis of False Positives

In the following analysis, we assume r heartbeats in one sweep for Memento, whereas DiMo and Sympathy allow sending up to $r - 1$ retransmissions in the case of unacknowledged messages. To compare the performance of the false positive rate, we assume the same sweep interval for three protocols which means that Memento’s and Sympathy’s sweep interval is equal to DiMo’s monitoring interval. In the analysis we assume all three protocols having the same packet-loss probability p_l for each hop.

For Sympathy, a false positive for a node occurs when the heartbeat from the node does not arrive at the sink in a sweep interval, assuming $r - 1$ retries on every hop. So a node will generate false positive with a possibility $(1 - (1 - p_l^r)^d)^n$, where d is the hop count to the sink and n the numbers of heartbeats per sweep. In Memento, the bitmask representing all nodes assumes them failed by default after the bitmap is reset at the beginning of each sweep interval. If a node doesn’t report to its parent successfully, i.e., if all the r heartbeats are lost in a sweep interval, a false positive will occur with a probability of p_l^r . In DiMo the node is reported missing if it fails to check in at the observer having a probability of p_l^r . In this case, a recovery message is triggered. Consider the case that the recovery message is not kept in the monitoring queue like the node-missing messages, but dropped after r attempts, the false positive rate results in $p_l^r(1 - (1 - p_l^r)^d)$.

Table 1 illustrates the false positive rates for the three protocols ranging the packet reception rate (PRR) between 80% and 95%. For this example the observed node is in a five-hop distance ($d = 5$) from the sink and a common

PRR	80%	85%	90%	95%
Sympathy ($n=1$)	3.93e-2	1.68e-2	4.99e-3	6.25e-4
Sympathy ($n=2$)	1.55e-3	2.81e-4	2.50e-5	3.91e-7
Memento	8.00e-3	3.38e-3	1.00e-3	1.25e-4
DiMo	3.15e-4	5.66e-5	4.99e-6	7.81e-8

Table 1: False positive rates for a node with hop count 5 and 3 transmissions under different packet success rates.

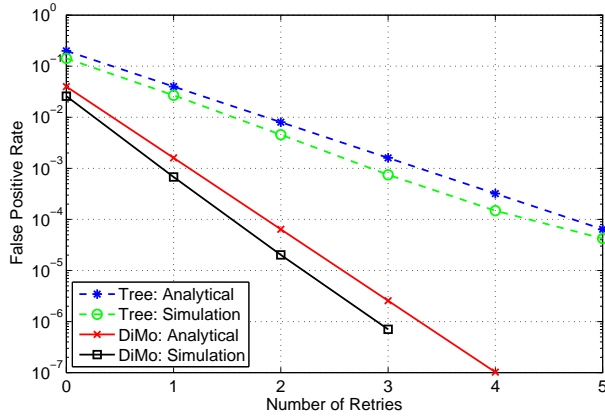
number of $r = 3$ attempts for forwarding a message is assumed. Sympathy clearly suffers from a high packet loss, but its performance can be increased greatly sending two heartbeats every sweep interval ($n = 2$). This however doubles the message load in the network, which is especially substantial as the messages are not aggregated, resulting in a largely increased load and energy consumption for nodes next to the sink. Comparing DiMo with Memento, we observe the paramount impact of the redundant relay on the false positive rate. DiMo offers a mechanism here that is not supported in Sympathy or Memento as it allows sending up to $r - 1$ retries for the observer and redundant relay. Due to this redundancy, the message can also be forwarded in the case of a total blackout of one link, a feature both Memento and Sympathy are lacking.

4.2 Simulation

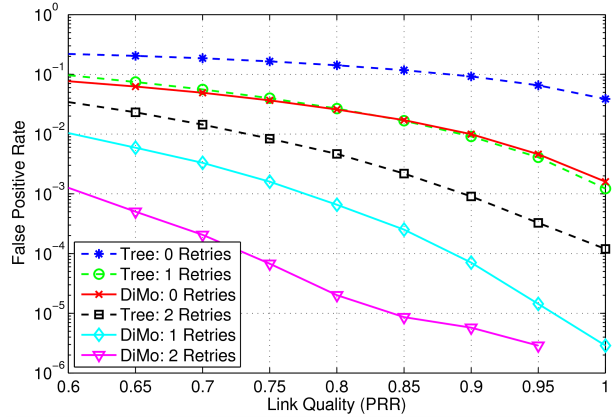
For evaluation purposes we have implemented DiMo in Castalia 1.3, a state of the art WSN simulator based on the OMNet++ platform. Castalia allows evaluating DiMo with a realistic wireless channel (based on the empirical findings of Zuniga et al. [8]) and radio model but also captures effects like the nodes’ clock drift. Packet collisions are calculated based on the signal to interference ratio (SIR) and the radio model features transition times between the radio’s states (e.g., sending after a carrier sense will be delayed). SpeckMAC [7], a packet based version of B-MAC, with acknowledgements and a low-power listening interval of 100 ms is used on the link layer. The characteristics of the Chipcon CC2420 are used to model the radio.

The simulations are performed for a network containing 80 nodes, arranged in a grid with a small Gaussian distributed displacement, representing an event detection system where nodes are usually not randomly deployed but rather evenly spread over the observed area. 500 different topologies were analyzed. The topology management results in a redundantly connected network with up to 5 levels \mathcal{L} and a maximum hop count \mathcal{H} of 6 to 8.

A false positive is triggered if the node fails to check in, which is primarily due to packet errors and losses on the wireless channel. In order to understand false positives, we set the available link’s packet reception rate (PRR) to 0.8, allowing us to see the effects of the retransmission scheme. Furthermore, this fixed PRR also allows a comparison with the results of the previous section’s analysis and is shown in Figure 1(a). The plot shows on the one hand side the monitoring based on a tree structure that is comparable to the performance of Memento, i.e., without DiMo’s possibility of sending a recovery message using an alternate relay. On the other hand side, the plot shows the false positive rate of DiMo. The plot clearly shows the advantage of DiMo’s redundancy, yet allowing sending twice as many heartbeats than the tree approach. This might not seem necessarily fair at first; however, in a real deployment it is always possible



(a) Varying number of retries; PRR = 0.8.



(b) Varying link quality.

Figure 1: False positives: DiMo achieves the targeted false positive rate of $1e-7$, also representing the reliability for successfully forwarding an event.

that a link fails completely, allowing DiMo to still forward the heartbeat. The simulation and the analysis show a slight offset in the performance, which is explained by a simulation artifact of the SpeckMAC implementation that occurs when the receiver’s wake-up time coincides with the start time of a packet. This rare case allows receiving not only one but two packets out of the stream, which artificially increases the link quality by about three percent.

The nodes are observed every $T_M = 4$ min, resulting in being monitored $1.3e5$ times a year. A false positive rate of $1e-6$ would result in having a particular node being wrongly reported failed every 7.7 years. Therefore, for a 77-node network, a false positive rate of $1e-7$ would result in one false alarm a year, being the targeted false-positive threshold for the monitoring system. DiMo achieves this rate by setting the numbers of retries for both the heartbeat and the recovery message to four. Hence the guard time T_R for sending the retries need to be set sufficiently long to accommodate up to ten messages and back-off times.

The impact of the link quality on DiMo’s performance is shown in Figure 1(b). The tree topology shows a similar performance than DiMo, if the same number of messages is sent. However, it does not show the benefit in the case of a sudden link failure, allowing DiMo to recover immediately. Additionally, the surprising fact that false positives are not going to zero for perfect link quality is explained by collisions. This is also the reason why DiMo’s curve for two retries flattens for higher link qualities. Hence, leaving room for retries is as important as choosing good quality links.

5. CONCLUSION

In this paper, we presented DiMo, a distributed algorithm for node and topology monitoring, especially designed for use with event-triggered wireless sensor networks. As a detailed comparative study with two other well-known monitoring algorithm shows, DiMo is the only one to reach the design target of having a maximum error reporting delay of 5 minutes while keeping the false positive rate and the energy consumption competitive.

The proposed algorithm can easily be implemented and also be enhanced with a topology management mechanism to provide a robust mechanism for WSNs. This enables its use in the area of safety-critical wireless sensor networks.

Acknowledgment

The work presented in this paper was supported by CTI grant number 8222.1 and the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. This work was also supported in part by phase II of the Embedded and Hybrid System program (EHS-II) funded by the Agency for Science, Technology and Research (A*STAR) under grant 052-118-0054 (NUS WBS: R-263-000-376-305). The authors thank Matthias Woehrle for revising a draft version of this paper.

6. REFERENCES

- [1] A. Mainwaring et al. Wireless sensor networks for habitat monitoring. In *1st ACM Int’l Workshop on Wireless Sensor Networks and Application (WSNA 2002)*, 2002.
- [2] A. Meier, T. Rein, et al. Coping with unreliable channels: Efficient link estimation for low-power wireless sensor networks. In *Proc. 5th Int’l Conf. Networked Sensing Systems (INSS 2008)*, 2008.
- [3] N. Ramanathan, K. Chang, et al. Sympathy for the sensor network debugger. In *Proc. 3rd ACM Conf. Embedded Networked Sensor Systems (SenSys 2005)*, 2005.
- [4] 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 (IEEE SECON 2006)*, 2006.
- [5] M. Strasser, A. Meier, et al. Dwarf: Delay-aware robust forwarding for energy-constrained wireless sensor networks. In *Proceedings of the 3rd IEEE Int’l Conference on Distributed Computing in Sensor Systems (DCOSS 2007)*, 2007.
- [6] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proc. 2nd European Workshop on Sensor Networks (EWSN 2005)*, 2005.
- [7] K.-J. Wong et al. Speckmac: low-power decentralised MAC protocols for low data rate transmissions in specknets. In *Proc. 2nd Int’l workshop on Multi-hop ad hoc networks: from theory to reality (REALMAN ’06)*, 2006.
- [8] M. Zuniga and B. Krishnamachari. Analyzing the transitional region in low power wireless links. In *IEEE SECON 2004*, 2004.
- [9] Fire detection and fire alarm systems – Part 25: Components using radio links. European Norm (EN) 54-25:2008-06, 2008.