# Designing a High-Reliability Low-Power Status Monitoring Protocol

Andreas Meier, Jan Beutel, Roman Lim and Lothar Thiele
Computer Engineering and Networks Lab, ETH Zurich
8092 Zurich, Switzerland
{a.meier,beutel,lim,thiele}@tik.ee.ethz.ch

***Abstract*— This paper describes the design and implementation of a high-reliability and low-power status monitoring protocol, e.g. for wireless alarm systems. The dual design approach followed consists of an in-depth device characterization with subsequent algorithm development. This in-depth understanding supports the conception and implementation of a correct design and leverages the optimization of the most critical algorithm components. The high-reliability status monitoring protocol operates based on monitoring windows with synchronous notification, subsequent acknowledgement rounds and periods of reduced activity, at regular intervals. This leads to deterministic and thus highly reliable operation of the status monitoring protocol implementation on the Tmote Sky platform.**

## I. Introduction

The monitoring of status, that is the monitoring of liveliness and failures, is a fundamental problem in distributed systems. Applications are abundant with prominent examples in the monitoring of computing resources, networks and assets as well as distributed safety and security. From a functional perspective, status monitoring requires to actively monitor the liveliness of both the nodes of a system and their respective communication channels in regular intervals while respecting timing constraints. In an active reporting scheme, an alarm is triggered on the detection of such a missing status report.

A practical example based on the (rather simple) case of a safety-critical, wireless fire-alarm application for a larger household can be characterized as follows: A central station supervises the operation of tens of nodes that are organized in a multi-hop network. Every node is required to report its availability to a single sink in regular intervals (e.g. within a few minutes). Since the application itself can typically not discriminate between an alarm and a (temporary) node or link failure, these events usually result in equally large overhead, e.g. an alert of the fire brigade or a manual inspection. Hence a reliability constraint on the order of a maximum of one false alarm per year and 20 nodes is a reasonable assumption translating into one critical packet failure per 20 node years. When operating on batteries a duty-cycle far below one percent is required to avoid the cumbersome change of batteries more than every few years.

Therefore the key requirement is ultra low-power operation while maintaining reliability and the respective latency constraints. Apart from using the most efficient hardware components and an appropriate architecture, the required low power operation can only be achieved by reducing activity in the system components, i.e. by using the lower power (sleep) states of the platform and the resulting reduced reactivity.

In this paper we explore the design space of ultra low-power operation and high-reliability responsiveness on the basis of a wireless status-monitoring application for safety and security-aware applications to be operated on battery-powered, resource-constrained devices. The scenario considered is similar to the fire alarm example discussed earlier. Design and development of applications for such highly limited devices is not straightforward but requires cross-layer know-how and meticulous analysis of the key operating parameters and characteristics involved. A dual approach that is driven by both the application requirements and the device characteristics is followed to achieve a functional implementation fulfilling the requirements of status monitoring. In particular, while designing the monitoring concept its feasibility is continuously crosschecked based on the implementation platform using crosscheck-specific testprograms. Opposed to a traditional design-flow based on subsequent phases of concept, simulation and implementation this approach reduces the overhead of multiple design iterations or even worse having to adapt key concepts in final phases of the implementation.

In the first part of the paper, we discuss the status monitoring problem and analyze its key characteristics. We then evaluate candidate device architectures and present a detailed device characterization of the radio and processing components as a base for the subsequent presentation of the status monitoring algorithm. We provide an in-depth protocol analysis for the target platform and present peculiarities resulting from the implementation. We conclude with experimental results and performance estimations of the protocol implementation.

### A. Related Work

A wireless failure-detection service has to tackle the challenges of the underlying unreliable and asynchronous network. In [1], Wang et al. analyzed heartbeat-based failure detection for (mobile) wireless ad-hoc networks. A linear and a two phase gossiping scheme was proposed to disseminate the nodes' status in the network. The proposed schemes were simulated based on a unit disk graph (UDG) model using a constant packet error rate. Although quite robust, the information dissemination is very slow and incurs considerable communication overhead. In [2], Tai et al. propose

a clustered approach to organize the dissemination of the heartbeat information. All nodes are organized in a single-hop neighborhood relative to their clusterhead. The clusterheads observe the nodes and forward failure reports across clusters. The probabilistic analysis, also based on the UDG model using constant packet error rate, provides probabilistic guarantees for robustness and efficiency. However, the latency aspect is not investigated, and the nodes operate on a 100 percent duty cycle. Rost et al. [3] propose a distributed failure detection system, requiring nodes to periodically send heartbeats. Each node runs an observer that reports a node failed, if a heartbeat is not received within a given time. The observer aggregates and sends the status information to a front-end where the failure of a node is determined. Whenever possible, the failure detection module reuses the broadcasts of other periodic protocols running on the network. This has been implemented on TinyOS using Mica2 evaluating the performance on a 55-node testbed. Although the message overhead is reduced, latency is also not considered here.

### B. Status Monitoring Characteristics

A status monitoring system needs to reliably and timely report failures, where a failure is characterized by a failure to report within a given window of time $T_{Fail}$. For example, a node failure can occur due to hard- and software errors or a node running out of energy. Furthermore, an operational but unreachable node is also considered to be failed.

Apart from the multi-hop and the single-sink assumption, the status monitoring system discussed in this paper is based on the following observation and assumptions:

a) **Radio Characteristic:** The design of a robust monitoring system has to tackle the underlying characteristics of wireless communication. Recent publications [4], [5] analyzed these characteristics and showed that packet errors occur at unpredictable times and often in bursts. In particular, this means that the immediate retransmission of an unsuccessful packet is not always a promising strategy.
b) **Broadcast Medium:** A wireless transmission can be heard by all neighboring nodes whether they are the intended receiver or not. Hence a wireless transmission is a broadcast to all nodes that are within reception range.
c) **Energy Consumption:** The radio module is the main consumer of energy, regardless whether the radio is actively transceiving packets or just idle listening. Therefore, on each node the radio's duty-cycle, i.e. the fraction of time the radio is not in its lowest power state, should be minimized. This can be achieved by reducing the time necessary for idle listening, by avoiding packet transmission to nodes that are not listening, by avoiding the overhearing of packets and by minimizing the overall traffic.
d) **TDMA:** A coordinated channel access provides a natural way to be energy efficient. With a TDMA scheme, the time is divided into slots, for which each node knows whether the radio has to be turned on, for transceiving packets. Furthermore, a controlled channel access allows for providing an exact timing and schedule.
e) **Simplicity:** Even simple algorithms tend to generate huge amounts of code when implemented on real sensor-network devices. To complicate things even further, algorithms often have to be tuned for performance improvements and adapted to provision for rare but crucial corner cases resulting in unmanageable state machines and bloating code size.

The status-monitoring system must report failed nodes reliably, i.e. a type II error also known as a false negative must not occur. This can only be achieved with an active reporting system, i.e. the nodes send messages on a regular basis, since a failed node cannot report itself by definition. Furthermore, there is a time constraint $T_{Fail}$ for detecting a failed node. On the other hand, the system should also be robust and not report operational nodes failed if a few packets are missed. In particular a node should not be reported failed just because a single link in a mesh topology is temporarily down.

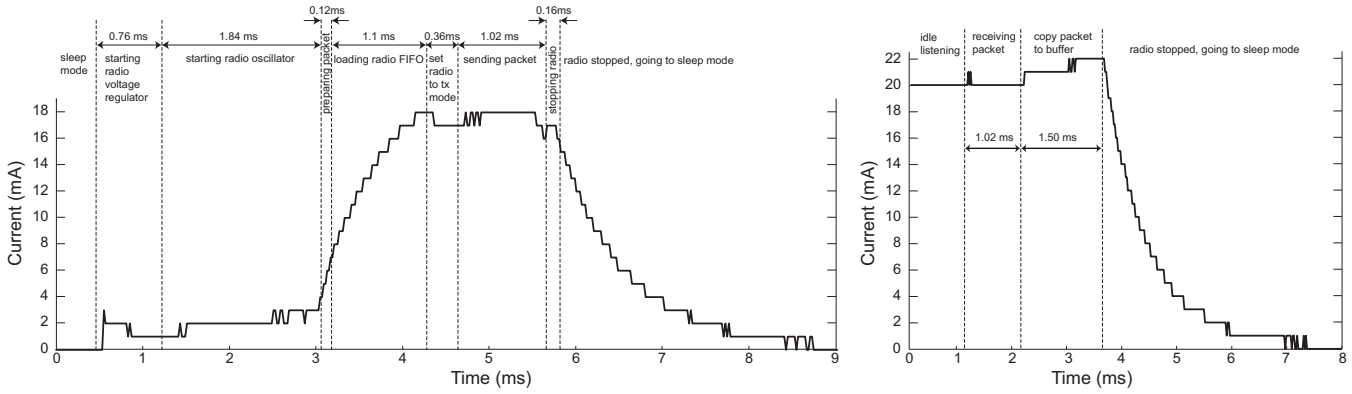
### C. Achieving High Reliability and Low-Power Operation

With the requirement to report the status of all nodes at a single sink in regular intervals and without missing a deadline, the monitoring protocol needs to be independent of other protocol tasks (user traffic, topology control, etc.). A regular window exclusively reserved for the monitoring protocol is the only way to guarantee a non-disruptive status monitoring service. Thus a synchronous approach appears most promising due to its energy efficiency. Initiated at a single sink with an adaptable period and duty-cycle, depending on the number of nodes in a network, the monitoring window can be adapted and the remainder of the "time-budget" can be used for user-specific traffic. In contrast to many "piggy-backed" approaches where status updates are interleaved with regular network traffic this separation of concerns of the status monitoring service and the application-specific user traffic is a guarantee for reliability, robustness and performance.

## II. Device Characterization

In order to develop a functional implementation based on the concept of a low-power protocol suitable for status monitoring, we evaluate candidate device architectures. The goal here is twofold: (i) To select the most suitable hardware architecture for the proposed application and (ii) to maximize efficiency of the implementation by gaining a deep understanding for the details of operation of the selected hardware. For prototyping it is desirable to use a state-of-the art platform that is commonly used and readily available. Such a platform choice results in a timely development time and typically offers a favorable community and support base. At a first glance candidates are among the TinyNode, Tmote Sky, BTnode, MicaZ, Mica2 and EyesIFX platforms.

A brief overview of platform requirements given the problem statement and its characteristics introduced in the previous section are fast wakeup and switching times (from RX to TX and vice versa) for the radio, ample bandwidth to assure quick packet delivery and reasonable balance of the power




(a) Packet transmission

(b) Packet reception

Fig. 1. Trace of the total power consumption for packet transmission and reception on a Tmote Sky (CC2420 radio and MSP430 CPU).

consumption for transmitting and low-power idle listening. Of primary importance is the lowest power level the platform can achieve, as the status monitoring application is separated from application level "user" traffic. This rules out AVR-based sensor network platforms as the BTnode and the Mica family and favors MSP430-based architectures. In the following, we will concentrate on the different options for radio transceivers.

The values shown in Table I are based on datasheet values and measurements where available. This is typical for an early phase in the evaluation process where extensive characterization has not yet been performed based on a test and measurement setup. The CC2420 radio is the best candidate for further analysis due to its superior oscillator startup times (factor 2.5) and quick switching times and high bit rate (factor 4x over the reliable 38kbps typically reported for the CC1000) that overrule the drawback of the higher RX power. As a further asset the CC2420, compared to much simpler radios such as the CC1000, features a buffered packet-based interface that reduces the low-level processing requirements of the host CPU considerably. While a CC2420-based system can perform high-level protocol processing on the CPU and the transmission of packets simultaneously, a CC1000-based system will be busy with low-level baseband and bit-level processing on the host CPU most of the time. This is a clear drawback as it requires much care in the software development of real-time critical low-level drivers, and keeps the system busy for longer periods of time before going back to sleep.

| | CC1000 | CC1021 | CC2420 | XE1205 |
|---|---|---|---|---|
| Bit Rate [kbps] | 76.8 | 153.6 | 250 | 152.3 |
| Osc Startup [us] | 2000 | 630-1550 | 860 | 1000 |
| Stby-to-RX [us] | 250 | 700 | 192 | 700 |
| Stby-to-TX [us] | 250 | 700 | 192 | 250 |
| RX-to-TX [us] | 200 | 14 | 192 | tbd |
| TX-to-RX [us] | 200 | 14 | 192 | 150 |
| Sleep [uA] | 1 | 1.8 | 1 | 0.2 |
| Standby [uA] | na | na | 426 | 850 |
| RX [mA] | 11.8 | 19.9 | 19.7 | 14 |
| TX Min/Max [mA] | 8.6/25.4 | 14.5/25.1 | 8.5/17.4 | 33/62 |

TABLE I

LOW-POWER RADIO TRANSCEIVER CHARACTERISTICS.

In a subsequent step, a more thorough analysis of the behavior and the timing and power characteristics was performed on a live system. The chosen Tmote Sky platform comprises a Chipcon CC2420 radio and an MSP430 microcontroller and is well supported both with documentation and software [6]. A test program written in TinyOS-2.0 that implements a simple synchronized packet transmission and reception was profiled, yielding exact traces of the power consumption and timing characteristics. Figure 1(a) shows the total supply current measured on a Tmote Sky for the transmission of one packet with a payload of 29 bytes, starting and ending in low-power sleep mode. The packet reception is depicted in Figure 1(b). Since the startup phase for both transmission and reception are very similar, the phases shown in Figure 1(b) are from the reception of a startframe deliminiter until the packet received has been copied to the MSP430 buffer for further processing, as well as the subsequent transition to sleep mode.

From this detailed analysis it is straightforward to see where most power is consumed and to create guidelines to be used for the further protocol development: (i) the transmission of a packet should be performed as quickly as possible, e.g. at maximum speed, (ii) RX and TX windows should be synchronized and (iii) the copying of data to and from the radio FIFO is a costly and power-hungry phase.

## III. ALGORITHM OVERVIEW

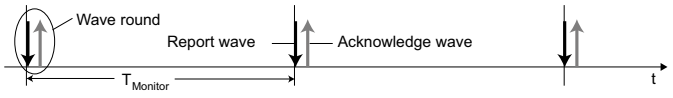
Based on the observations made in section I-B and the device characteristics presented in the previous section, we propose a heartbeat-style failure-detection service based on reporting and acknowledging waves as detailed below:

### A. General Wave Scheme

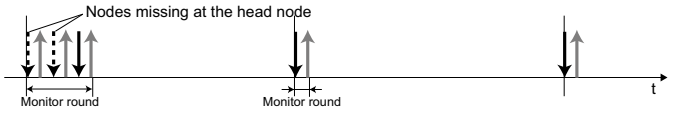
The algorithm is based on so-called waves that propagate from the outside of a multi-hop neighborhood towards a single sink (the *head node* $\mathcal{H}$). A wave can be imagined as the nodes each sending a message, one after the other, whereas the nodes on the outside of the network will send before the nodes that are closer to the head node. For this scheme, the nodes are required to wake up synchronously every *monitor interval* $T_{Monitor} < T_{Fail}$ and start a new *monitor round* consisting

of one or more *wave rounds*. A wave round consists of a *reporting wave* towards $\mathcal{H}$ (building up the nodes' status list along the way) and a subsequent *acknowledgement wave* in the opposite direction as shown in Figure 2(a). If all nodes are reported and acknowledged within the first wave round the nodes can go to sleep or do other tasks (e.g. process user traffic) until the next monitor round is due. In the case of a missing report or acknowledgement of a node, up to $R_s$ independent wave rounds can be initiated as depicted in Figure 2(b). In the case of a failure, either in a transmission or in a node itself, a burst of reporting and acknowledgement waves will be initiated in order to recover the missing nodes if somehow possible. After each successful monitor round, the head node knows whether all nodes are alive or not. This information can then be forwarded to a central control station to make decisions upon further actions.

Contained within this algorithm, there are two concepts to increase the robustness and the reliability: Redundant paths and acknowledgments. Messages are broadcasted to all neighbors and therefore all links are used to propagate the status information towards the head node. Nevertheless, a nodes' status beacon might not be received by nodes that still have to send a beacon. In this case, the acknowledgement wave will initiate subsequent recovery waves. In the following a detailed description of the wave scheme components is presented along with Algorithms 1 and 2 given in the Appendix.




(a) Each monitor round consist of at least one wave round, i.e. a report and an acknowledgement wave.




(b) Subsequent recovery wave rounds are initiated if nodes are missing at the head node resulting in a variable duration of the monitor rounds.

Fig. 2. Wave scheme of the failure-detection service.

### B. Reporting Wave

Every node maintains a status list $S$ of nodes known to be operational, i.e. depicting each node's limited view of the network. This list is cleared at the beginning of each monitor round, and all nodes known to be operational are subsequently entered. During the progression of the reporting wave, every node will broadcast a status beacon containing the node's current status list according to a schedule. Whenever a node receives a status message, the received status list $S_r$ is merged with the local list $S$, i.e. $S := S \cup S_r$. Following this paradigm the status of the nodes is disseminated towards $\mathcal{H}$.

### C. Acknowledgement Wave

After the reporting wave has arrived at the head node, it checks its status list $S_{\mathcal{H}}$ and determines if all known nodes have reported in on this wave. The head node will then initiate the acknowledgement wave by sending an acknowledgement message. The nodes in the network forward the acknowledgement message in the reverse order, i.e. the node that has sent the status message last will forward the head's acknowledgement message first. The acknowledgement message has several purposes: (i) The message contains either a positive or a negative acknowledgement. The latter case will automatically initiate another wave round in order to immediately recover the missing node(s). (ii) The message contains the head node's status list $S_{\mathcal{H}}$. This list is merged with the receiving nodes own list. The acknowledgement wave is used for the recovery of a node missing temporarily. (iii) The acknowledgement message contains the schedule for the next monitor round that is discussed in detail in the next section. (iv) Furthermore, the acknowledgement wave is also used to synchronize the nodes with the head node. After forwarding a positive acknowledgement message or after $R_s$ wave rounds have elapsed, the nodes can go to a sleep state terminating the monitor round.

### D. Transmission Schedule

During a wave, either reporting or acknowledgement, every node is supposed to broadcast one message. This requires the coordination of the channel access in order to avoid collisions. We propose to use a TDMA schedule. This will not only prevent collisions but will also provide a time efficient and therefore also energy efficient access to the channel. The TDMA schedule divides the waves into slots of equal length.

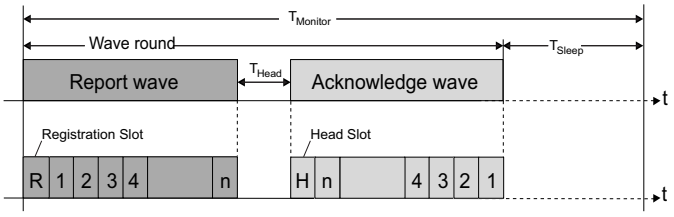



Fig. 3. TDMA schedule for a wave round. The slots 1–n are assigned to the nodes in the network except for the head node. The reporting wave provides an additional slot 'R' that allows new nodes to register to the network. The acknowledgement wave on the other hand provides an additional slot 'H' for the head node to initialize an acknowledgement wave.

The reporting wave starts with a special slot *R* that is used for new nodes to register to the network. The remaining slots 1 to n, are assigned to the nodes in the network. Each node is in control of exactly one slot, during which the node's current status list $S$ is broadcasted. The nodes schedule their transmissions according to the hop distance to the head node, e.g. the node with the highest hop count gets assigned the earliest slot. This slot assignment is performed by the head node that is required to detect and monitor the hop count of all individual nodes. The hop count is determined with the arrival of a node's registration at the head node. In many cases several nodes will have an equal hop count to the head node that are then assigned to slots using a random order. It would also be possible to extend this scheduling scheme

to order the nodes according to their connectivity respective to nodes with a smaller hop count or to schedule nodes that are reported missing more often earlier. However, adding such extensions increases the complexity and jeopardizes the paradigm of creating a simple and lightweight algorithm. After the reporting wave, the head node has to configure the acknowledgement message: (i) check the local status list $S_{\mathcal{H}}$, (ii) check for a new node to be registered and adapt the schedule accordingly and (iii) add a time stamp for network synchronization. As soon as the packet is ready, the elapsed time $T_{\mathcal{H}}$ since the end of the reporting wave is determined and the head node broadcasts the acknowledgement message in the first slot $H$ of the acknowledgement wave. The order of the slot assignments of the acknowledgement wave is the inverted order of the reporting wave.

### E. Node Synchronization

In order to reduce the node's duty cycle, the proposed TDMA slots should be as short as possible which requires good synchronization [7], [8]. Therefore, the nodes synchronize to the head node with every incoming acknowledgement wave. At the beginning of each monitor round, the head node takes a time stamp $T_{Start}$. This time stamp is propagated with the acknowledgement wave to synchronize the nodes. Between synchronizations, the clocks then start drifting within the limits of a platform specific parameter $\theta$, that specifies the clock drift in parts per million (ppm). The maximal drift time is therefore calculated as $T_{drift} = \theta T_{\Delta}$, where $T_{\Delta}$ denotes the elapsed time since the last synchronization.

One important factor for a good synchronization and minimal slot times are the characteristic of the underlying platform. Another important factor for determining the maximal slot length is the guard time balancing the clock drift. This guard time increases linearly with the last synchronization point that is propagated with the acknowledgement wave. This means that the first reporting wave requires a longer guard time than the subsequent wave(s) of a monitor round.

### F. Initialization

The startup of the wave scheme is initialized by the head node, who sends an acknowledgement message every $T_{Monitor}$ containing the schedule for the next reporting wave. The first wave will only contain the *register slot* $R$ and with more and more nodes responding and registering at the head node evolve into a schedule for the whole network on subsequent iterations. An unregistered node has to listen on the channel continuously until it receives a first acknowledgement message providing the schedule, allowing the node to register itself. The node knows being successfully registered upon receiving an acknowledge wave containing its id in the schedule. Upon an unsuccessful registration, the node waits for a random number of rounds (exponential back off) before registering again in order to resolve possible congestion. In a multi-hop environment, not every node is in the head node's one hop neighborhood requiring a node receiving a registration request to forward the request by attaching it to its own status message. A second request within the same round is dropped in order to minimize the computation overhead. In addition to the request, the head node needs to know the new node's hop count. The registration request additionally provides the node's hop count that is initially set to zero and incremented with every hop the request is forwarded within the network. It is assumed that the head node is known in advance. Alternatively, a node could also select itself as the head node based on an initial (random) timer. This initialization scheme allows one node to be registered per monitor round. If insufficient, the head node dynamically schedules a second or third registration slot if a node has registered in the last round, but similar to section III-D this again complicates the algorithm and its implementation.

### G. Special Cases

A problem occurs if all messages of the acknowledgement wave are missed: First, a node does not know whether the acknowledgement was positive and second the newest synchronization time can be missed. In this case a node has to keep listening on the channel for additional wave rounds. Should no acknowledgement message be received for the time required to send the maximal number $R_s$ of wave rounds, the node can turn off its radio. However, in this case the node looses proper synchronization and cannot send a status message in the subsequent reporting wave. Such a monitor round will therefore consist of at least two wave rounds.

### H. Protocol Alternatives

The first reporting wave might have rather long slots. This is due to the long guard time that is directly proportional to the monitor interval $T_{Monitor}$. For a long $T_{Monitor}$ it might be more efficient if the head node initiates a synchronization wave at first providing the newest time stamp $T_{Start}$. Due to the short slot time of the synchronization and the subsequent reporting wave, these two waves take less time than the loosely synchronized single reporting wave. In addition, the synchronization wave can be used to disseminate status information, reducing the amount of required second rounds.

## IV. TDMA Analysis and Implementation

After the introduction of the algorithm in the previous section we will now discuss some peculiarities resulting from the implementation of a TDMA protocol on the Tmote Sky.

### A. Slot Time Minimization

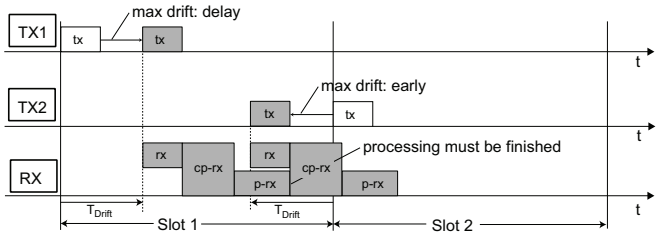
The proposed TDMA schedule divides the waves into slots of the same size. Minimizing the slot time will therefore also optimize the overall efficiency. There are two constraints for the slot time $T_{slot}$: (i) the clock inaccuracy must be accounted for and (ii) there must be enough time available for the nodes to process a received message and be ready in time to send a message in the following slot. The crystals on the different nodes do not oscillate with exactly the same characteristics. As a result, the individual clocks each run with slightly

different speed, exhibiting individual drift characteristics and therefore need to be synchronized regularly. There are several approaches to synchronize nodes [7] in a wireless ad hoc network, such as unidirectional synchronization, round-trip measurements or reference broadcasts [8]. The first approach fits perfectly into the proposed wave scheme, has only a small overhead but has the downside that the transmission delay needs to be estimated. The clocks can run too fast, but also too slow. In the worst case, the sender of a first slot (TX1) has a clock running too slow while the clock of subsequent slot's sender (TX2) is running too fast. The slot time must be chosen in such a way, that a third node (RX) can receive both message, i.e. finish processing the first message before the second is received (see Figure 4(a) with the measured values shown in Table II). This results in a minimum slot time for the clock-drift compensation

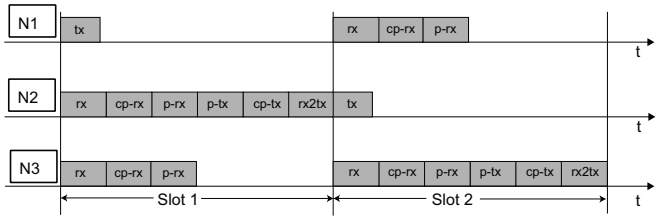
$$T^d_{slot}(T_\Delta) = 2\theta T_\Delta + T_{receive}, \quad \text{where} \qquad (1)$$
$$T_{receive} = T_{rx} + T_{cp-rx} + T_{p-rx}.$$

The second constraint for the slot time deals with the receiving and sending taking place in subsequent slots. After a message is received and copied to the CPU it must be processed before the new message can be prepared for sending. This is illustrated in Figure 4(b) and results in a minimum slot time for the compensation of the message processing of

$$T^p_{slot} = T_{receive} + T_{p-tx} + T_{cp-tx} + T_{rx2tx}. \qquad (2)$$




(a) Clock drift compensation: In a first slot the transmission is maximally delayed (clock running too slow on TX1), while the subsequent slot is too early (clock running too fast on TX2). The slot time needs to be adapted in such way that both messages can be received and processed.




(b) Received data processing: N2 receives the message from N1 in the first slot and needs a certain time to process the received packet and prepare the next transmission to be sent to N3 in the next slot.

Fig. 4. Constraints for the minimal slot time $T_{slot}$.

### B. Wave Timing Optimization

The maximal clock drift increases with time and therefore with every slot. It would be possible to calculate the optimal slot time for every slot individually. However, this is a rather complex task and we fix the slot time for a single wave. With a fixed slot time $T_{slot}$, the wave time is given by

$$T_{wave} = (n+1) \cdot T_{slot} \cdot (1 + 2\theta). \qquad (3)$$

The slot time needs to guard the maximal clock drift that can occur during a wave and is therefore different for the different wave types. For the acknowledge wave, the network is synchronized at the beginning resulting in a slot time of:

$$\begin{aligned} T^{ack}_{slot} &= \max(T^p_{slot}, T^d_{slot}(T_\Delta)) | T_\Delta = nT^d_{slot}(T_\Delta) \\ &= \max(T^p_{slot}, T_{receive}/(1-2n\theta)) \end{aligned} \qquad (4)$$

The synchronization for the first reporting wave has been performed in the previous monitor round, i.e $T_\Delta$ can be assumed to be equal $T_{Monitor}$. The slot time for this first reporting round is therefore:

$$T^{r1st}_{slot} = \max(T^p_{slot}, T^d_{slot}(T_{Monitor})) \qquad (5)$$

The subsequent reporting waves require compensating only the drift of the preceding acknowledgement wave:

$$\begin{aligned} T_\Delta &= T^{ack}_{wave} + nT^d_{slot}(T\Delta) \\ T^{r2nd}_{slot} &= \max(T^p_{slot}, T^d_{slot}(T_\Delta)) \\ &= \max(T^p_{slot}, (2\theta T^{ack}_{wave} + T_{receive})/(1-2n\theta)) \end{aligned} \qquad (6)$$

### C. Implementation Performance and Characteristics

Based on the device characterization presented in Section II the detailed timing characteristics were determined on the Tmote Sky platform using a prototype implementation of the TDMA scheme discussed earlier. The parameters determined through measurement are given in Table II. With the optimizations and the use of the time-stamping support of the CC2420 a synchronization error of $\leq 60\mu s$ was achieved. The time-stamping support allows associating an interrupt generated time-stamp with an event occurring in the radio, e.g. a preamble detection or a packet transmission.

| Abbreviation | Description | Time [ms] |
|---|---|---|
| $T_{rx}$ | Message Reception | 1.02 |
| $T_{tx}$ | Message Transmission | 1.02 |
| $T_{cp-rx}$ | Message copy to CPU | 1.50 |
| $T_{p-rx}$ | Message processing on reception | 0.26 |
| $T_{p-tx}$ | Preparation of a message to be sent | 0.12 |
| $T_{cp-tx}$ | Message copy to radio | 1.10 |
| $T_{rx2tx}$ | Radio switching time from RX to TX | 0.36 |
| $\theta$ | Clock drift | 20 ppm |

TABLE II
DEVICE SPECIFIC TIMINGS USED FOR SLOT AND WAVE TIMINGS

## V. EXPERIMENTAL RESULTS AND EVALUATION

### A. Testbed Setup and Experimental Procedure

The algorithms discussed were implemented on the Tmote Sky platform [6] using TinyOS 2.0. The software structure was designed to keep the code as simple and short as possible. In particular code segments resulting in unpredictable program flow and execution time were avoided.

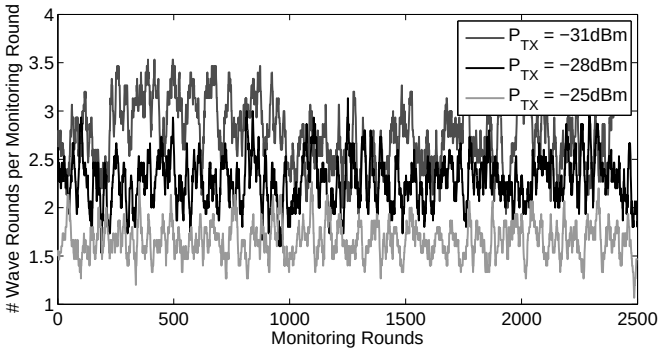



Fig. 5. Over half a day the average number of wave rounds per monitor round indicates the packet loss rate at different transmit power levels.

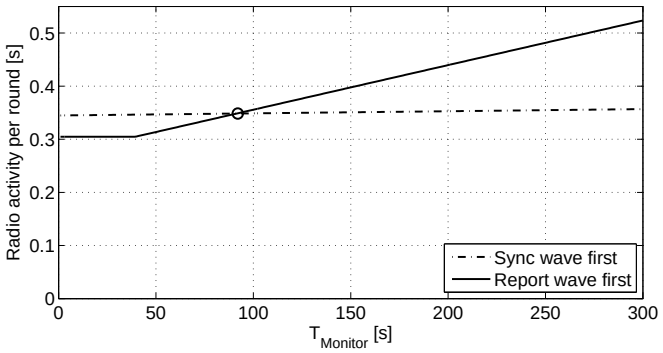



Fig. 6. For long surveillance intervals it is more efficient to start with a synchronization wave first, i.e. with a negative acknowledge wave.

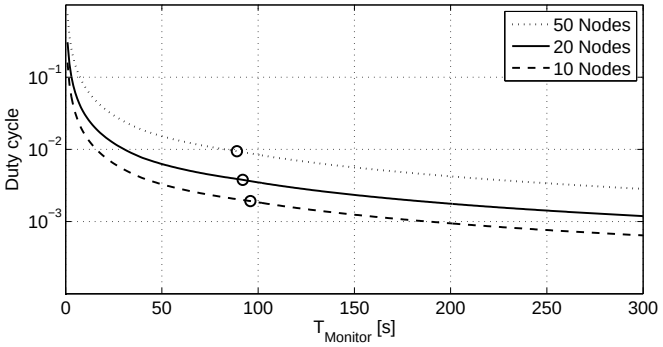



Fig. 7. The radio duty cycle is depending on the network size and the monitor interval. The threshold discussed in Figure 6 is denoted with the three circles.

Experiments were performed in an office floor scenario with a dozen nodes scattered around a room. At first a very relaxed time schedule was used, i.e. rather long slot times. Using this setup the worst-case execution time was measured. With these execution-time measurements and the device analysis in Section II we had all the necessary information to calculate the optimal slot times according the protocol analysis provided in Section IV. Using the optimized parameter set, experiments were continued to assess the performance and robustness of the protocol. To the best of our knowledge there is no commonly used setting or benchmark scenario available for such testing, making results hard to analyze and compare. In order to get a feeling for a worst-case scenario in a multi-hop environment a setup on a table top in combination with very low transmit power levels resulting in a s-hop neighborhood towards the sink node was used. This approach allows executing test runs using a common communication characteristic on many nodes.

Based on the algorithm, it is expected that nodes are being reported as failed after a time of at most $T_{Monitor} + T^{max}_{Monitor-round} < T_{Fail}$ showing a rather uniformly distributed reporting time. Long-term testing that randomly turned off nodes and measured the respective reporting time showed exactly this behavior. In particular, no false negatives occurred with tests being run over multiple hours. Upon decrease of the signal quality an increase in the number of wave rounds necessary to acknowledge all nodes is expected. This was emulated by gradually reducing the transmit power and limiting the maximum number of wave rounds to $R_s := 4$. The result of this test (using three different transmission power levels) is shown in Figure 5. The number of wave rounds executed is plotted using a sliding window of the size 15 allowing to show the average number of rounds required and their variance. The effect of the reduced signal quality for lower transmit power levels is with a lot more nodes being reported as failed for a shorter amount of time.

### *B. Performance Estimation*

Based on the experimental results, the performance is computed based on Equations 1–6 in the following. In Section III-H the possibility was discussed to start the monitor round with a (negative) acknowledge wave in order to first synchronize the network and as a result minimizing the slot time. This trade-off is depicted in Figure 6, showing the required radio activity relative to the monitor interval for a 20-node network. The curve for sending the report wave first starts rather flat and increases linearly starting at $T_{Monitor} = 40s$. This is due to the constraints of the slot time. For short monitor intervals, the processing time is the dominating factor. For long intervals on the other hand, the drift compensation needs to be considered, resulting in a linear increase of the slot time. The plot for the case of sending the acknowledge wave first increases gradually since only the wave requires a guard time but not the individual slots. For $T_{Monitor} > 90s$ the costs for sending an additional synchronization wave start to pay off. With increasing network size the radio's activity is also increased which can be seen in Figure 7, showing the nodes' duty cycle. Doubling the network size almost doubles the nodes radio activity. However, with a network size of 20 nodes and a monitoring interval of five minutes, a duty cycle of about one-tenth of a percent can be achieved.

## VI. Conclusion

The dual approach followed in the design of the monitoring application showed many positive effects. The understanding of the most critical components of the algorithm would have been near-sighted if not impossible without the in-depth characterization and analysis of the used platform prior to the

implementation of the whole system. Moreover, a platform is not just made up of its components but is a functional unit that needs to be analyzed from a system perspective and not only by citing key figures from component datasheets. For reliability and robustness of the monitoring application the separation of the protocol tasks into a monitoring window and a data transfer or sleep window for the remainder of the time as well as the synchronized approach have proven to be a good choice. The results presented in both analysis and experiment yield promising results that are able to guarantee performance for the status monitoring application.

## 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.

## REFERENCES

[1] S.-C. Wang and S.-Y. Kuo, "Communication strategies for heartbeat-style failure detectors in wireless ad hoc networks," in *DSN '03: Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03)*, pp. 361–376, June 2003.

[2] A. T. Tai, K. S. Tso, and W. H. Sanders, "Cluster-based failure detection service for large-scale ad hoc wireless network applications," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, (Washington, DC, USA), p. 805, IEEE Computer Society, 2004.

[3] S. Rost and H. Balakrishnan, "Memento: A health monitoring system for wireless sensor networks," in *IEEE SECON 2006*, (Reston, VA), Sept. 2006.

[4] D. Son, B. Krishnamachari, and J. Heidemann, "Experimental study of concurrent transmission in wireless sensor networks," in *Proc. 4th ACM Conf. Embedded Networked Sensor Systems (SenSys 2006)*, (New York, NY, USA), pp. 237–250, ACM Press, 2006.

[5] A. Willig and R. Mitschke, "Results of bit error measurements with sensor nodes and casuistic consequences for design of energy-efficient error control schemes.," in *Proc. 3rd European Workshop on Sensor Networks (EWSN 2006)*, pp. 310–325, 2006.

[6] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pp. 364–369, IEEE, Piscataway, NJ, Apr. 2005.

[7] K. Römer, P. Blum, and L. Meier, *Sensor Networks*, ch. Time Synchronization and Calibration in Wireless Sensor Networks. John Wiley & Sons, New York, July 2005.

[8] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proc. 5th Symp. Operating Systems Design and Implementation (OSDI 2002)*, pp. 147–163, ACM Press, New York, Dec. 2002.

## APPENDIX

**Algorithm 1** Algorithm for head node – without registration

```
1: var S_H    // status list
2: var wave_count
3: var T_Start    // head node's time stamp
4: timer monitor_round
5:
6: upon startup
7:     start monitor_round timer ← small value
8: end upon
9:
10: upon timeout of monitor_round timer
11:     S_H ← {}
12:     T_Start ← Now
13:     wave_count ← 0
14:     turn radio on
15:     start monitor_round timer ← T_Monitor
16:     start report_finished timer ← Time for reporting wave
17: end upon
18:
19: upon reception of message m
20:     S_H ← m.S ∪ S_H
21: end upon
22:
23: upon timeout of report_finished timer
24:     var m    // new message
25:     m.ack ← S_H == {'All'}
26:     m.S ← S_H
27:     m.time ← T_Start
28:     m.schedule ← determine schedule
29:     broadcast m
30:     if m.ack then
31:         turn radio off
32:     else
33:         if ++wave_count ≤ R_s then
34:             start report_finished timer ← Time for wave round
35:         else
36:             report missing nodes
37:             turn radio off
38:         end if
39:     end if
40: end upon
```

**Algorithm 2** Message exchange for node $u$ – without registration

```
1: var S ← {u}
2: var m_ack ← Null
3: var T_start    // head node's time stamp
4: timer monitor_round    // set at initialization
5:
6: upon timeout of monitor_round timer
7:     S ← {u}
8:     turn radio on
9:     if last round time stamp available then
10:         start report_slot timer ← next Report Slot Time
11:     end if
12:     start turn_off timer ← time for R_s wave rounds
13:     start monitor_round timer ← (T_Monitor − Guard Time)
14: end upon
15:
16: upon reception of reporting message m
17:     S ← m.S ∪ S
18: end upon
19:
20: upon reception of ack message m
21:     m_ack ← m
22:     S ← m.S ∪ S
23:     if T_start is not form current round then
24:         update T_start
25:         update monitor_round timer ← T_Monitor
26:         if u's ack slot has not yet passed then
27:             start ack_slot timer
28:         end if
29:     end if
30:     if positive acknowledgement in m then
31:         update turn_off timer ← after ack slot
32:     else
33:         start report_slot timer ← next Report Slot Time
34:     end if
35: end upon
36:
37: upon timeout of turn_off timer
38:     turn radio off
39: end upon
40:
41: upon timeout of report_slot timer
42:     m_ack ← Null    //reset for next ack wave
43:     broadcast current status list S
44: end upon
45:
46: upon timeout of ack_slot timer
47:     m.S ← m.S ∪ S
48:     broadcast acknowledgement message m
49: end upon
```