pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols

Marco Zimmerling*, Federico Ferrari*, Luca Mottola†, Thiemo Voigt†, Lothar Thiele*
*Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
†Swedish Institute of Computer Science (SICS), Kista, Sweden
{zimmerling, ferrari, thiele}@tik.ee.ethz.ch {luca, thiemo}@sics.se

ABSTRACT

We present pTunes, a framework for runtime adaptation of low-power MAC protocol parameters. The MAC operating parameters bear great influence on the system performance, yet their optimal choice is a function of the current network state. Based on application requirements expressed as network lifetime, end-to-end latency, and end-to-end reliability, pTunes automatically determines optimized parameter values to adapt to link, topology, and traffic dynamics. To this end, we introduce a flexible modeling approach, separating protocol-dependent from protocol-independent aspects, which facilitates using pTunes with different MAC protocols, and design an efficient system support that integrates smoothly with the application. To demonstrate its effectiveness, we apply pTunes to X-MAC and LPP. In a 44-node testbed, pTunes achieves up to three-fold lifetime gains over static MAC parameters optimized for peak traffic, the latter being current—and almost unavoidable—practice in real deployments. pTunes promptly reacts to changes in traffic load and link quality, reducing packet loss by 80 % during periods of controlled wireless interference. Moreover, pTunes helps the routing protocol recover quickly from critical network changes, reducing packet loss by 70 % in a scenario where multiple core routing nodes fail.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*

General Terms

Design, Experimentation, Performance

Keywords

Runtime adaptation, parameter optimization, MAC protocol, multi-objective, centralized, end-to-end, sensor network

IPSN'12, April 16–20, 2012, Beijing, China.

1. INTRODUCTION

Media access control (MAC) protocols play a key role in determining the performance and reliability of low-power wireless networks, but very few of the many proposed solutions have been adopted in real deployments [23, 34].

Challenges. There exists a large conceptual gap between the high-level application requirements and the low-level MAC protocol operation [23]. In particular, it requires expert knowledge to find MAC operating parameters whose performance satisfies given application requirements.

In most deployments today, the choice of MAC parameters is based on experience and rules of thumb involving a coarse-grained analysis of expected network load and topology dynamics. This can yield a performance far off the application requirements [24]. Alternatively, system designers perform several field trials to identify suitable MAC parameters [8]. This time-consuming and deployment-specific practice, however, is hardly sustainable in the long term.

Even if the MAC parameters are appropriate at one time, they are likely to perform poorly when the network state changes. Wireless link quality varies significantly over time, leading to unpredictable packet loss [41]; harsh environmental conditions cause nodes to be temporarily disconnected or to fail [2]; and changes in the routing topology or the sensing activity result in traffic fluctuations. Statically configured MAC protocols cannot cope with these dynamics.

To perform efficiently, MAC protocols must adapt their operating parameters at runtime. One way to approach this problem is to embed adaptivity within the protocol operation [21]. This, however, hard-codes the adaptation decisions and thus limits their applicability. Instead, separating adaptivity from the protocol operation enables higher-layer services to dynamically adjust the operating parameters [32]. Although a few mechanisms utilize such control knobs, they focus either on a single metric—typically energy [9, 22, 28]—or consider only local metrics, such as per-hop latency [5, 31]. Real-world applications, however, often require to balance *multiple* conflicting needs such as reliability, energy, and latency, expressed on a *network-wide* scale [7, 36, 38].

Contributions and road-map. To tackle the issues above, we present pTunes, a framework for runtime adaptation of low-power MAC protocol parameters. In pTunes, users specify application requirements in terms of *network lifetime*, *end-to-end reliability*, and *end-to-end latency*—key performance metrics in real-world applications [7, 8, 36–38]. Using information about the current network state, pTunes automatically determines optimized MAC parameters whose performance meets the requirements specification.

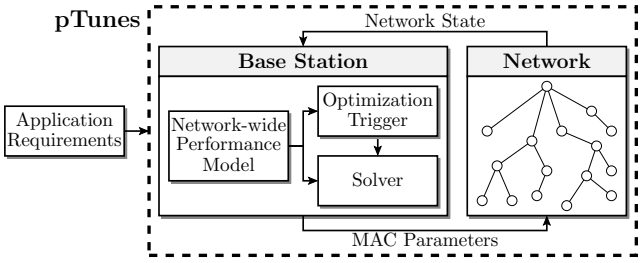


Figure 1: The pTunes framework.

This paper makes the following contributions:

- We introduce the pTunes framework, targeting data collection systems using tree-shaped routing topologies atop low-power MAC protocols. As shown in Fig. 1, the base station collects reports on the *network state*, such as topology and link quality information, required to evaluate the network-wide metrics we target. The *optimization trigger* decides when to carry out the parameter optimization, based on a periodic timer or a mechanism that uses the *network-wide performance model* to check if the *application requirements* are violated under the current network state. The *solver* determines a vector of *optimized MAC parameters*, which is disseminated in the network and installed on all nodes. Sec. 2 further characterizes the *multi-objective* parameter optimization problem in pTunes.
- We design a well-structured modeling framework to solve the parameter optimization problem. Our layered modeling approach, described in Sec. 3, separates application-level, protocol-independent, and protocol-dependent quantities. This increases generality and flexibility, as it cleanly determines what needs to be changed to account for a different MAC protocol. We apply this modeling approach to two state-of-the-art protocols, X-MAC [5] and LPP [29], based on their implementations in Contiki. We leverage these models throughout the rest of the paper, ultimately demonstrating that they are both practical and accurate.
- We present the design and implementation of an efficient system support to meet the system-level challenges arising in pTunes. These include, for instance, the timely collection of accurate network state with little energy overhead and minimum disruption for the application operation. As described in Sec. 4, unlike most approaches in the literature, we meet these requirements with a novel solution for collecting network state and disseminating new MAC parameters *independent* of other protocols running concurrently. Our approach utilizes fast and reliable Glossy network floods [16], allowing pTunes to collect *consistent* network state snapshots, taken with microsecond accuracy at all nodes simultaneously, with very low energy cost.

After illustrating implementation details in Sec. 5, we evaluate pTunes in Sec. 6 using experiments with X-MAC and LPP on a 44-node testbed. For instance, we find that adapting their parameters using pTunes enables up to threefold lifetime gains over static MAC parameters optimized for peak traffic, the latter being current practice in many real deployments [23]. pTunes promptly reacts to changes in traffic load and link quality, meeting application-level requirements through an 80 % reduction in packet loss during periods of controlled wireless interference. Moreover, we find that pTunes helps the routing protocol recover from critical network changes, reducing the total number of parent switches and settling quickly on a stable, high-quality routing topology. This reduces packet loss by 70% in a scenario where multiple core routing nodes fail simultaneously.

We discuss design trade-offs of pTunes in Sec. 7, review related work in Sec. 8, and conclude the paper in Sec. 9.

2. OPTIMIZATION PROBLEM

In pTunes, we simultaneously consider three key performance metrics of real-world applications [7, 8, 36–38]: network lifetime T, end-to-end reliability R, and end-to-end latency L. The MAC parameter optimization problem thus becomes a *multi-objective optimization problem (MOP)*. This involves optimizing the objective functions $T(\mathbf{c})$, $R(\mathbf{c})$, and $L(\mathbf{c})$, where $\mathbf{c}$ is a vector of MAC parameters, or *MAC configuration* for short. There may exist not one unique optimal solution to this MOP, but rather a set of solutions that are optimal in the sense that no other solution is superior in *all* objectives. These are known as *Pareto-optimal* solutions and represent different optimal trade-offs among T, R, and L.

Given the many Pareto-optimal solutions, a natural question is which solution best serves the application demands. pTunes needs to make this decision at runtime in an automated fashion, without involving the user (*e.g.*, to manually select a solution from a set of candidates). With this requirement in mind, we adopt from among the many MOP solving techniques an approach inspired by the epsilon-constraint method [20]. This method treats all but one objective as constraints, and thus provides a natural interface for specifying typical requirements of low-power wireless systems such as "batteries should last for at least 6 months." Using this approach, pTunes solves the MOP by optimizing one objective subject to constraints on the remaining objectives

$$\begin{array}{lllll} \text{Maximize/Minimize} & M_1(\mathbf{c}) & & & \\ \text{Subject to} & M_2(\mathbf{c}) & \geq, \leq & C_1 & \\ & M_3(\mathbf{c}) & \geq, \leq & C_2 & \end{array} \tag{1}$$

where each M_i is one among $\{T, R, L\}$ and $\{C_1, C_2\}$ are *soft* requirements to be satisfied in the long run, corresponding to the best-effort operation of many data collection systems [18]. By varying $\{C_1, C_2\}$, all Pareto-optimal solutions can be generated. Based on concrete values for $\{C_1, C_2\}$ set by the user on some objectives, pTunes translates the application requirements into a solution that optimizes the remaining objective. The resulting solution is Pareto-optimal while representing the trade-off provided by the user.

As an example, in long-term structural monitoring the major concern is typically system lifetime, but domain experts also require a certain reliability in delivering sensed data [7]. Based on (1), maximizing network lifetime subject to a minimum end-to-end reliability is specified as

$$\begin{array}{lll} \text{Maximize} & T(\mathbf{c}) & \\ \text{Subject to} & R(\mathbf{c}) & \geq \quad R_{min} \end{array} \tag{2}$$

In addition, we may impose a constraint on end-to-end latency, $L(\mathbf{c}) \leq L_{max}$, if timely data delivery is relevant.

3. MODELING FRAMEWORK

To facilitate using pTunes with different low-power MAC protocols, we break up the modeling into three distinct layers, as shown in the model frame in Fig. 2. The upper layer defines application-level metrics (R, L, T) as functions of link and node-specific metrics (R_l, L_l, T_n). The middle layer expresses these metrics in a protocol-independent manner,

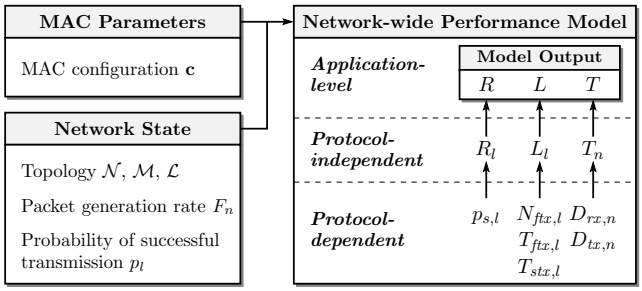


Figure 2: Modeling framework with inputs, output, and mapping between modeling layers. *Only the protocol-dependent layer must be changed to prepare the network-wide performance model for another MAC protocol.*

and provides the entry point for the modeling of a concrete MAC protocol by exposing six terms to the lower protocol-dependent layer. Binding these terms to concrete protocol-specific expressions is sufficient to adapt the network-wide performance model in pTunes to a given MAC protocol.

Model inputs are the MAC parameters and the network state, comprising information about routing topology, traffic volumes, and link qualities. As a measure of the latter, we take the probability of successful transmission p_l over the link to the parent in the routing tree. To keep our models simple and practical, we assume the delivery of individual packets to be independent of their size, of the delivery of any other packet, and of the link direction they travel along. As illustrated in Sec. 4, our runtime evaluation of p_l captures the impact of channel contention on link quality, allowing us not to consider it explicitly in our models. Testbed experiments in Sec. 6.2 show that this approach results in highly accurate models for both X-MAC and LPP.

3.1 Application-level Metrics

In a typical data collection scenario with static nodes, a tree-shaped routing topology provides a unique path from every sensor node to a sink node. These paths are generally time-varying, as the routing protocol adapts them according to link quality estimates among other things [18, 33]. In the following, we use $\mathcal{N}$ to denote the set of *all nodes* in the network excluding the sink, and $\mathcal{M} \subseteq \mathcal{N}$ to denote the set of *source nodes* generating packets. We also indicate with $\mathcal{L}$ the set of *communication links* that form the current routing tree. The *path* $\mathcal{P}_n \subseteq \mathcal{L}$ originating at node $n \in \mathcal{M}$ includes all intermediate links that connect node n to the sink. Table 1 lists these and other modeling terms we use to denote network state and protocol-dependent quantities.

End-to-end reliability and latency. The reliability $R_{\mathcal{P}_n}$ of path $\mathcal{P}_n$ is the expected fraction of packets delivered from node $n \in \mathcal{M}$ to the sink along $\mathcal{P}_n$. Thus, $R_{\mathcal{P}_n}$ is the product of per-hop reliabilities R_l, $l \in \mathcal{P}_n$. We define the *end-to-end reliability* R as the average reliability of all paths $\mathcal{P}_n$.

$$R = \frac{1}{|\mathcal{M}|} \sum_{n \in \mathcal{M}} R_{\mathcal{P}_n} = \frac{1}{|\mathcal{M}|} \sum_{n \in \mathcal{M}} \left(\prod_{l \in \mathcal{P}_n} R_l \right) \tag{3}$$

Likewise, the latency $L_{\mathcal{P}_n}$ of path $\mathcal{P}_n$ is the expected time between the first transmission of a packet at node $n \in \mathcal{M}$ and its reception at the sink. Thus, $L_{\mathcal{P}_n}$ is the sum of per-hop latencies L_l, $l \in \mathcal{P}_n$. Similar to (3), we define the *end-to-end latency* L for *successfully delivered* packets as the average latency of all paths $\mathcal{P}_n$, and omit the formula.

Term	Description
$\mathcal{N}$	Set of all nodes in the network excluding the sink
$\mathcal{M}$	Set of source nodes generating packets
$\mathcal{L}$	Set of all links forming the routing tree
F_n	Packet generation rate of node n
p_l	Probability of successful transmission over link l
$p_{s,l}$	Probability of successful unicast transm. over link l
$N_{ftx,l}$	No. of failed unicast transm. before success over link l
$T_{ftx,l}$	Time for a failed unicast transmission over link l
$T_{stx,l}$	Time for a successful unicast transmission over link l
$D_{rx,n}$	Fraction of time radio is in receive mode at node n
$D_{tx,n}$	Fraction of time radio is in transmit mode at node n

Table 1: Glossary of modeling terms used to denote network state and protocol-dependent quantities.

We define R and L as averages of all source-sink paths since the global, long-term performance is of ultimate interest for most data collection systems [36–38]. Local, short-term deviations from the requirements are usually tolerated, provided they are compensated in the long run. In other scenarios (*e.g.*, industrial settings), it might be more appropriate to define R and L as the minimum reliability and the maximum latency among all source-sink paths, which would only require modifying the two definitions above.

Network lifetime. Similar to prior work [27], we define the *network lifetime* T as the expected shortest node lifetime T_n, $n \in \mathcal{N}$. We assume the sink has infinite energy supply.

$$T = \min_{n \in \mathcal{N}} (T_n) \tag{4}$$

This choice is motivated by the fact that a single node failure can lead to network partition and service interruption. It is also possible to express other notions of network lifetime in pTunes, such as the time until some fraction of nodes fails, again requiring only to modify the definition in (4).

3.2 Protocol-independent Modeling

The section above expressed the application-level metrics R, L, and T as functions of per-hop reliability R_l, per-hop latency L_l, and node lifetime T_n (see Fig. 2). We now define the latter three in a protocol-independent manner, which increases flexibility and generality by isolating protocol-dependent quantities. We omit a few explicit expressions and refer to an extended report [42] where applicable.

Per-hop reliability and latency. Several factors influence these metrics: (*i*) the MAC operation when transmitting packets, (*ii*) packet queuing throughout the network stack due to insufficient bandwidth, and (*iii*) application-level buffering (*e.g.*, to perform in-network processing). The MAC parameters control (*i*) and may avoid the occurrence of (*ii*), provided a MAC configuration exists that provides sufficient bandwidth. Application-specific in-network functionality akin to (*iii*) is out of the scope of this work.

We present next expressions for per-hop reliability and latency due to the MAC operation, corresponding to (*i*). Additionally, pTunes includes models to detect situations akin to (*ii*) [42]. In fact, as we show in Sec. 6.2, pTunes automatically adjusts the MAC parameters to provide higher bandwidth against increased traffic, thus avoiding the occurrence of local packet queuing until the network capacity attainable in our experimental setting is fully exhausted.

We define the *per-hop reliability* R_l of link $l \in \mathcal{L}$, which connects node $n \in \mathcal{N}$ to its parent m in the routing tree, as

the probability that n successfully transmits a packet to m.

$$R_l = 1 - (1 - p_{s,l})^{N+1} \quad (5)$$

Here, $p_{s,l}$ represents the MAC-dependent probability that a single unicast transmission over link l succeeds, and N is the maximum number of retransmissions per packet, modeling automatic repeat request (ARQ) mechanisms used by many low-power MAC protocols to improve reliability.

Furthermore, we define the *per-hop latency* L_l of link l as the time for node n to deliver a message to its parent m.

$$L_l = N_{ftx,l} \cdot T_{ftx,l} + T_{stx,l} \quad (6)$$

$T_{ftx,l}$ and $T_{stx,l}$ are the MAC-dependent times spent for each failed and the final successful transmission. The expected number of failed transmissions $N_{ftx,l}$ depends on $p_{s,l}$ and N, and the retransmission policy of the MAC protocol [42].

Node lifetime. Sensor nodes consume energy by communicating, sensing, processing, and storing data. Adapting the MAC parameters has no significant impact on the latter three, but affects energy expenditures on communication to a large extent, as the radio is typically the major energy consumer. Given a battery capacity Q, we define the *node lifetime* T_n of node $n \in \mathcal{N}$ as

$$T_n = Q/(D_{tx,n} \cdot I_{tx} + D_{rx,n} \cdot I_{rx} + D_{idle,n} \cdot I_{idle}) \quad (7)$$

where I_{tx}, I_{rx}, and I_i are the current draws of the radio in transmit, receive, and idle mode. T_n is thus the expected node lifetime based on the fractions of time in each mode $D_{tx,n}$, $D_{rx,n}$, and $D_{idle,n} = 1 - D_{tx,n} - D_{rx,n}$, which depend on the MAC protocol and the traffic volume at node n.

The *traffic volume* is the rate at which nodes send and receive packets. A node $n \in \mathcal{N}$ generates packets at rate F_n and receives packets from its children $\mathcal{C}_n \subseteq \mathcal{N}$ in the routing tree, if any. The rate of packet reception depends on each child's packet transmission rate $F_{tx,c}$ and the individual per-hop reliabilities R_{l_c} of links l_c, $c \in \mathcal{C}_n$, connecting each child c with n. Thus, node n transmits packets at rate

$$F_{tx,n} = (N_{rtx,l} + 1) \cdot \left(F_n + \sum_{c \in \mathcal{C}_n} F_{tx,c} \cdot R_{l_c} \right) \quad (8)$$

$N_{rtx,l}$ is the expected number of retransmissions per packet over link l, which is a function of N and the MAC-dependent probability that a retransmission occurs [42].

We demonstrate next the modeling of a concrete MAC protocol. This requires to find expressions for six protocol-specific terms, as shown in Fig. 2 and described in Table 1.

3.3 Protocol-specific Modeling

We use two state-of-the-art MAC protocols to exemplify the protocol-specific modeling. X-MAC [5] is representative of many sender-initiated MAC protocols based on low-power listening (LPL) [32] that proved viable in real-world deployments [23]. Recent work focuses on receiver-initiated MAC protocols such as low-power probing (LPP) [29]. In the following, we refer to implementations of X-MAC and LPP in Contiki 2.3, which we also use in our experiments in Sec. 6.

3.3.1 Sender-initiated: X-MAC

Fig. 3 shows a successful unicast transmission in X-MAC. Nodes wake up periodically for T_{on} to poll the channel $\langle 1 \rangle$, where T_{off} is the time between two channel polls. To send a packet, a node transmits a sequence of *strobes* $\langle 2 \rangle$, short

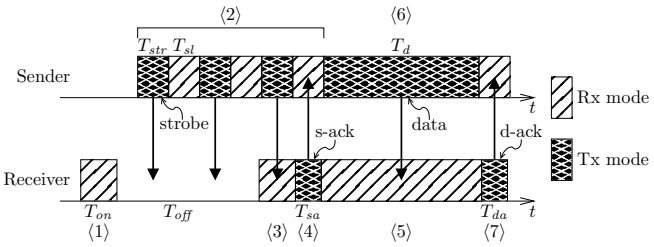


Figure 3: Unicast transmission in X-MAC.

packets containing the identifier of the receiver. Strobing continues for a period sufficient to make at least one strobe overlap with a receiver wake-up $\langle 3 \rangle$. The receiver replies with a *strobe acknowledgment (s-ack)* $\langle 4 \rangle$ and keeps the radio on awaiting the transmission of the *data packet* $\langle 5 \rangle$. The sender transmits the data packet upon receiving the s-ack $\langle 6 \rangle$ and waits for the *data acknowledgment (d-ack)* $\langle 7 \rangle$ from the receiver. Afterward, both nodes turn off their radios.

Failed s-ack, d-ack, and data packet transmissions are handled by timeouts. When a timeout occurs, the sender backs off for a random period and retries beginning with the strobing phase, for at most N times. Broadcasts proceed similarly to unicast transmissions, but the strobing phase lasts for $T_m = 2 \cdot T_{on} + T_{off}$ to make a strobe overlap with the wake-up of all neighboring nodes. Nodes receiving a broadcast strobe keep their radio on until they receive the data packet at the end of the sender's strobing phase.

Several variables are adjustable in the X-MAC implementation we consider. However, three specific parameters affect its performance to a major extent.

$$\mathbf{c} = [T_{on}, T_{off}, N] \quad (9)$$

We let pTunes adapt these parameters at runtime, using the X-MAC-specific models presented next.

Per-hop reliability. We determine $p_{s,l}$ in (5), the probability that a single unicast from node n to its parent m succeeds. This is the case if m hears a strobe (with probability $p_{str,l}$), the s-ack reaches n, and m receives the data packet. Each of the latter two succeeds with probability p_l, collected at runtime as part of the network state (see Sec. 4).

$$p_{s,l} = p_{str,l} \cdot p_l^2 \quad (10)$$

The probability of receiving at least one strobe is

$$p_{str,l} = 1 - (1 - p_l)^{(T_{on} - T_{str})/T_{it}} \quad (11)$$

where $T_{it} = T_{str} + T_{sl}$ is the duration of a strobe iteration at the sender, which includes the length of a strobe transmission T_{str} and listening T_{sl} for an s-ack.

Per-hop latency. We determine $T_{ftx,l}$ and $T_{stx,l}$ in (6), the times spent for failed and successful transmissions. $T_{ftx,l}$ depends on whether node n receives an s-ack. If so, n stops strobing, sends the data packet, and times out after T_{out}. Otherwise, n sends strobes for T_m. In either case, node n backs off for T_b before retransmitting.

$$T_{ftx,l} = (N_{it} T_{it} + T_d + T_{out}) p_{str,l} + T_m (1 - p_{str,l}) + T_b \quad (12)$$

Here, $N_{it} = (T_{on} + T_{off})/(2 \cdot T_{it})$ is the average number of strobe iterations before m possibly replies with an s-ack.

The time for a successful transmission $T_{stx,l}$ includes the time to wait for the s-ack and to send the data packet.

$$T_{stx,l} = N_{it} \cdot T_{it} + T_d \quad (13)$$

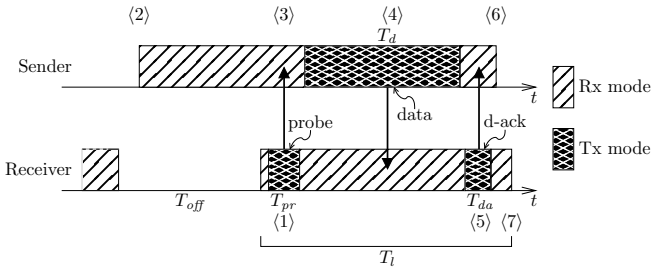


Figure 4: Unicast transmission in LPP.

Node lifetime. We determine $D_{tx,n}$ and $D_{rx,n}$ in (7), the fractions of time spent by the radio in transmit and receive mode. Both quantities depend on the rate F_{arx,l_c} at which node n attempts to receive a packet from child c over link l_c

$$F_{arx,l_c} = (N_{rtx,l_c} + 1) \cdot F_{tx,c} \cdot p_{str,l_c} \tag{14}$$

where $F_{tx,c}$ and p_{str,l_c} are given by (8) and (11).

We first consider $D_{tx,n}$. Node n transmits during packet receptions from child c (to send s-ack and d-ack) and during packet transmissions to its parent m (to send strobes and data packet). We define T_{rxt,l_c} and $T_{txt,l}$ as the average times spent by the radio in transmission mode during receptions over link l_c and transmissions over link l [42].

$$D_{tx,n} = F_{tx,n} \cdot T_{txt,l} + \sum_{c \in \mathcal{C}_n} F_{arx,l_c} \cdot T_{rxt,l_c} \tag{15}$$

Next we consider $D_{rx,n}$. Node n is in receive mode during packet transmissions to its parent m (to receive s-ack and d-ack) and packet receptions from child c (to receive strobe and data packet). Let $T_{txr,l}$ and T_{rxr,l_c} be the average times spent by the radio in reception mode during transmissions over link l and receptions over link l_c [42]. The fraction of time in receive mode for actual communication is

$$D_{rxc,n} = F_{tx,n} \cdot T_{txr,l} + \sum_{c \in \mathcal{C}_n} F_{arx,l_c} \cdot T_{rxr,l_c} \tag{16}$$

In addition, n is in receive mode for $F_{cc} = T_{on}/(T_{on} + T_{off})$ during channel checks, which leads to

$$D_{rx,n} = D_{rxc,n} + (1 - D_{rxc,n}) \cdot F_{cc} \tag{17}$$

3.3.2 Receiver-initiated: LPP

Fig. 4 shows a successful unicast transmission in LPP. Nodes periodically turn on their radio for T_l and transmit a short *probe* ⟨1⟩ containing their own identifier. To send a packet, a node turns on its radio ⟨2⟩ and listens for a probe from the intended receiver ⟨3⟩, for at most T_{on}. Then the sender transmits the data packet ⟨4⟩, waits for the d-ack from the receiver ⟨5⟩, and goes back to sleep ⟨6⟩. After sending the d-ack, the receiver keeps the radio on until a timeout signals the end of the active phase ⟨7⟩. Between two active phases nodes sleep for T_{off}. To send a broadcast, the sender keeps its radio on for $T_m = 2 \cdot T_l + T_{off}$ to receive a probe from every neighbor, immediately replying to each received probe with the data packet. We let pTunes adapt the same set of LPP parameters **c** in (9) as for X-MAC (note that T_{on} has now a different meaning as explained above).

Per-hop reliability. A single LPP unicast from node n to its parent m succeeds if n receives a probe from m (with probability $p_{pr,l}$) and then successfully transmits the data packet (with probability p_l).

$$p_{s,l} = p_{pr,l} \cdot p_l \tag{18}$$

The probability that n receives a probe is given by

$$p_{pr,l} = 1 - (1 - p_l)^k \tag{19}$$

where $k = (T_{on} - T_{pr})/T$ is the number of possible probe receptions while node n listens for at most T_{on}. The term $T = T_l + T_{off} + T_{rm}/2$ denotes the LPP duty cycle period, which is the sum of radio on-time, radio off-time, and a small random quantity with uniform distribution $\{0, \ldots, T_{rm}\}$ to scatter probe transmissions.

Per-hop latency. We determine the time for a failed transmission. If node n receives a probe after waiting for $T_{pw,l}$, it sends the data packet and times out after T_{out}. Otherwise, n listens for T_{on}. Node n retransmits after backing off for T_b.

$$T_{ftx,l} = (T_{pw,l} + T_d + T_{out}) p_{pr,l} + T_{on}\,(1 - p_{pr,l}) + T_b \tag{20}$$

On average, node n receives a probe from its parent m after

$$T_{pw,l} = T_{pr} + \sum_{i=1}^{\lfloor k \rfloor + 1} p_i \cdot T_i \tag{21}$$

where p_i is the probability that n receives the i-th probe, and T_i is the expected time to await the i-th probe [42].

The time for a successful transmission includes the time to wait for a probe and to send the data packet.

$$T_{stx,l} = T_{pw,l} + T_d \tag{22}$$

Node lifetime. We determine the fractions of time in transmit and receive mode. Both depend on the rate F_{arx,l_c} at which node n receives packets from child c over link l_c

$$F_{arx,l_c} = (N_{rtx,l_c} + 1) \cdot F_{tx,c} \cdot p_{s,l_c} \tag{23}$$

where $F_{tx,c}$ and p_{s,l_c} are given by (8) and (18).

Node n transmits a probe every duty cycle period T and sends d-acks to child c with frequency F_{arx,l_c}. Further, n is in transmit mode for $T_{txt,l}$ to send packets to m [42].

$$D_{tx,n} = T_{pr}/T + T_{da} \sum_{c \in \mathcal{C}_n} F_{arx,l_c} + F_{tx,n} \cdot T_{txt,l} \tag{24}$$

Node n is in receive mode when the radio is turned on but does not transmit probes or d-acks. Additionally, node n is in receive mode for $T_{txr,l}$ during packet transmissions [42].

$$D_{rx,n} = (T_l - T_{pr})/T - T_{da} \sum_{c \in \mathcal{C}_n} F_{arx,l_c} + F_{tx,n} \cdot T_{txr,l} \tag{25}$$

4. SYSTEM SUPPORT

pTunes must tackle several system-level challenges to obtain an efficient runtime operation. This section highlights these challenges and presents the system support we design to meet them. This includes a novel approach for collecting network state information and disseminating new MAC parameters, and the techniques and tools we use to solve the parameter optimization problem efficiently.

4.1 Challenges

Minimum disruption. pTunes must reduce the amount of disruption perceived by the application, particularly with respect to application data traffic, to avoid influencing its behavior beyond the adaptation of MAC parameters. This is in itself a major challenge in low-power wireless networks [10].

Timeliness. Timely collection of accurate network state, computation of optimized MAC parameters, and their reliable and rapid dissemination are fundamental to pTunes.

Only this way pTunes can provide MAC operating parameters that do match the current network state. However, it is difficult to perform the above operations in a timely manner, especially when involving resource-constrained devices.

Consistency. pTunes requires consistent snapshots of network state, possibly captured by all nodes at the same time. Otherwise, optimizing MAC parameters based on information different from the actual network conditions may even negatively affect the system performance. Coordinating distributed sensor nodes to achieve consistency is challenging, given their bandwidth and energy limitations.

Energy efficiency. pTunes must meet all the previous challenges while introducing only a limited, possibly predictable, energy overhead at the sensor nodes. To be viable, the overhead of pTunes must not outweigh the gains obtained from adapting the MAC parameters.

4.2 Collection and Dissemination

pTunes uses Glossy network floods [16] to collect network state information and disseminate MAC parameters. In particular, pTunes exploits Glossy's time synchronization service to schedule and execute both operations within short time frames, repeated every *collection period* T_c. Every frame starts with a Glossy flood initiated by the sink, which serves to time-synchronize the nodes and disseminate new MAC parameters. Following the initial flood by the sink, each of the other nodes initiates a flood in turn within exclusive slots, reporting network state for the subsequent trigger decision and parameter optimization.

The collection period T_c can range from a few tens of seconds to several minutes depending on network dynamics and application needs, and represents a trade-off between the energy overhead of pTunes and its responsiveness to changes in the network: a shorter T_c permits more frequent parameter updates but increases the energy consumption of the nodes. The efficiency of Glossy allows us to limit the length of the periodic collection and dissemination frames, thus keeping the energy overhead to a minimum. For instance, we measure on a 44-node testbed an average duration of 5.2 ms for a single flood, and an average radio duty cycle of 0.35 % due to pTunes collection and dissemination for $T_c = 1$ min, which reduces to about 0.07 % for $T_c = 5$ min. Given that state-of-the-art low-power MAC protocols exhibit duty cycles of 3–7 % in testbed settings comparable to ours [13, 18], the energy overhead of pTunes is marginal.

An alternative to our approach may be to piggyback network state on application packets and to use a variant of Trickle [25] to disseminate MAC parameters. We employed this approach at an early stage of this work, but found it inadequate for our purposes. For instance, running Trickle concurrently with data collection increases contention, especially during parameter updates, which degrades application data yield [10]. Moreover, piggybacking on data packets induces a dependency on the rate and reliability of application traffic. In low-rate applications, it may take a very long time until network state from all nodes becomes available for optimization. Packets may also be generated at different times and experience varying end-to-end delays (*e.g.*, due to contention or routing loops), so the collected network state is likely to be out-of-date and inconsistent. Our approach avoids these problems by temporally decoupling collection and dissemination from application tasks, and by leveraging consistent network state snapshots taken with microsecond accuracy at all nodes independently of application traffic.

In particular, pTunes collects three pieces of network state from each node: (*i*) the node id and the id of the routing parent, to allow pTunes to learn about the current routing tree ($\mathcal{N}$, $\mathcal{M}$, $\mathcal{L}$); (*ii*) the number of packets generated per second F_n, allowing pTunes to determine the traffic volumes; and (*iii*) the ratio $H_{s,l}/H_{t,l}$ of successful and total number of link-layer handshakes over link l to the routing parent. There are two handshakes in X-MAC, strobe/s-ack and data/d-ack; LPP features only the latter (see Figs. 3 and 4). To account for parent switches and link dynamics, a node maintains counters $H_{s,l}$ and $H_{t,l}$ in a way similar to an exponentially weighted moving average (EWMA). Based on their ratio received from each node and by taking the square root, pTunes obtains estimates of the probability of successful transmission p_l of all links in the current routing tree. The collected information totals 6 bytes per node.

4.3 Optimization Tools

Applying the optimization problem in (1) to our X-MAC and LPP models in Sec. 3 leads to a mixed-integer nonlinear program (MINLP) with non-convex objective and constraint functions. To solve it efficiently, we use the ECL^{i}PS^e constraint programming system [1]. Its high-level programming paradigm allows for a succinct modeling of our optimization problem. We use modules to separate protocol-independent from protocol-dependent code; the latter amounts to about 100 lines for each X-MAC and LPP.

We use the branch-and-bound algorithm coupled with a complete search routine, both provided by the interval constraint (IC) solver of ECL^{i}PS^e. The running time of the optimization depends to a large extent on the size of the search space. To reduce it, we exploit the fact that MAC protocols are commonly implemented using hardware timers. The resolution of these timers determines the maximum required granularity of the MAC timing parameters. We therefore discretize the domains of T_{on} and T_{off} considered for adaptation, letting ECL^{i}PS^e determine values with millisecond granularity. Based on the literature and our own experience, we set the upper bounds of N and T_{off} to 10 retransmissions and 1 s; T_{on} is chosen such that a node listens long enough to overlap with exactly one receiver wake-up in LPP, and with at least one but not more than three strobe transmissions in X-MAC. For these settings and in the scenarios we tested, representative of a large fraction of deployed sensor networks, ECL^{i}PS^e finds optimized MAC parameters within a few tens of seconds on a standard laptop computer. Compared with our current approach, which leverages general-purpose algorithms and off-the-shelf implementations, dedicated solution techniques and implementations are likely to improve significantly on this figure.

5. IMPLEMENTATION DETAILS

On the sensor nodes, we use Contiki 2.3. We extended the existing X-MAC implementation with link-layer retransmissions and an interface to adjust the parameters in (9) at runtime. Since the existing LPP implementation suffered from performance problems that could bias our results, we re-implemented LPP within the Contiki stack and extended it in the same way as X-MAC. For data collection we use Contiki Collect, which maintains a tree-based routing topology using expected transmissions (ETX) as cost metric.

The pTunes control application running on the base station is implemented in Java. It retrieves collected network state from the sink, starts the optimization process depending on the trigger decision, and transfers new MAC parameters back to the sink for dissemination.

An important decision for pTunes is when to trigger the parameter optimization. In general, we want to optimize as often as possible to make the MAC parameters closely match the network state. At the same time, we want to minimize the energy overhead for collection and dissemination, and need to consider that running the solver takes time. Therefore, pTunes provides three basic optimization triggers to decide when to start the solver. Nevertheless, pTunes users can implement their own application-specific triggers using a set of basic interfaces we provide.

Among the triggers we provide, *TimedTrigger* optimizes periodically, where the period is typically a multiple of the collection period T_c. In this way, a TimedTrigger may launch the solver immediately after the collection of network state, and pTunes floods the new MAC parameters at the next dissemination. Nevertheless, depending on application-specific requirements and performance goals, users may also combine a TimedTrigger with one of the following two triggers.

A *ConstraintTrigger* uses the model to estimate the current network performance based on the collected network state, and launches the solver only if any of the constraints in (1) is violated. A ConstraintTrigger may be implemented to tolerate short-term violations of a constraint, or a violation within some threshold around the constraint. Alternatively, a *NetworkStateTrigger* can infer directly from the network state if the MAC parameters should be updated. For example, a NetworkStateTrigger may fire if it detects a significant increase in traffic volume, thus starting the solver to find MAC parameters that provide higher bandwidth.

6. EXPERIMENTAL RESULTS

This section uses measurements from a 44-node testbed to study both the effectiveness of pTunes and the interactions of MAC parameter adaptation with the routing protocol. Our experiments reveal the following key findings:

- Validation against measurements shows that our X-MAC and LPP models are highly accurate.
- pTunes automatically determines MAC parameters that provide higher bandwidth when the traffic load increases. This avoids the occurrence of queuing until the network capacity attainable in our setting is fully exhausted.
- In the scenarios we tested, pTunes achieves up to threefold lifetime gains over static MAC parameters optimized for peak traffic volumes.
- In a scenario where the packet rates vary across nodes and fluctuate over time, pTunes satisfies given end-to-end latency and reliability requirements at peak traffic while extending the network lifetime at relaxed traffic.
- During phases of controlled wireless interference, pTunes reduces packet loss by 80 % compared to static MAC parameters optimized for the applied traffic without interference, satisfying given end-to-end reliability requirements.
- pTunes helps the routing protocol recover from critical network changes, reducing the number of parent switches and settling quickly on a stable routing topology. This reduces packet loss by 70% in a scenario where multiple core routing nodes fail simultaneously.

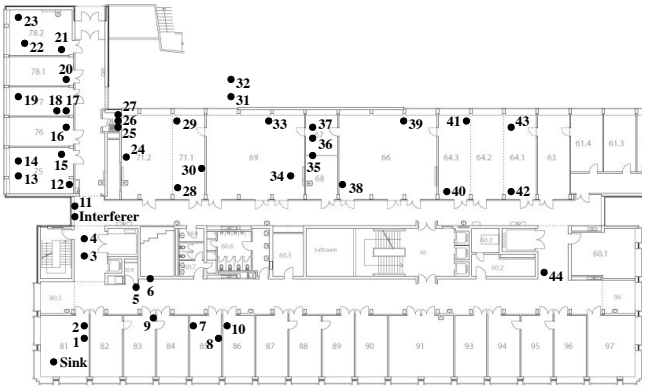


Figure 5: Testbed layout. *Nodes 31 and 32 are located outside on the rooftop; the interferer is only used in Sec. 6.6.*

6.1 Setting and Metrics

Testbed. Our testbed spans one floor in an ETH building [3, 14]. Fig. 5 shows the positions of the 44 Tmote Sky nodes distributed in several offices, passages, and storerooms; two nodes are located outside on the rooftop. The sink is connected to a laptop computer that acts as the base station. Paths between nodes and sink are between 1 to 5 hops in length. Nodes transmit at the highest power setting, using channel 26 to limit the interference with co-located WiFi.

Metrics. Our evaluation uses the metrics defined in Sec. 3.1. To measure network lifetime, we use Contiki's energy profiler to obtain the fractions of time the radio is in receive, transmit, and idle mode. Then, we compute *projected* node lifetimes using (7) and current draws from the CC2420 data sheet, assuming batteries constantly supply 2000 mAh at 3 V. When pTunes is enabled, the measured network lifetime includes the energy overhead of pTunes collection and dissemination, performed every $T_c = 1$ min in all experiments. We measure end-to-end reliability based on sequence numbers of data packets received at the sink. To measure end-to-end latency, we exploit Glossy's time synchronization service and timestamp data packets at the source.

Requirements. We consider typical requirements of real-world data collection applications: maximize network lifetime while providing a certain end-to-end reliability [7, 37]. We also enforce a constraint on end-to-end latency, accounting for applications that require timely delivery [8].

$$\begin{array}{ll} \text{Maximize} & T(\mathbf{c}) \\ \text{Subject to} & R(\mathbf{c}) \geq 95\,\% \;\text{ and }\; L(\mathbf{c}) \leq 1\,\text{s} \end{array} \tag{26}$$

pTunes solves (26) at runtime to determine optimized MAC parameters. If there exists no solution because either constraint in (26) is unsatisfiable (*e.g.*, due to extremely low link qualities), pTunes maximizes R without constraints. This policy serves to exemplify the capabilities of pTunes; other application-specific policies can be implemented within the pTunes optimization triggers.

Methodology. We compare pTunes with several static MAC configurations optimized for a variety of different workloads and application requirements, as listed in Table 2. We found these MAC configurations using pTunes and extensive experiments on our testbed. Existing MAC adaptation approaches, on the other hand, consider only per-link and per-node metrics [5, 31] or focus solely on energy [9, 22, 28], rendering the comparison against pTunes purposeless.

	Name	Configuration $[T_{on}, T_{off}, N]$	Performance Trade-Off (R, L, T)
X-MAC	S1	[16 ms, 100 ms, 8]	(high, low, low)
	S2	[11 ms, 250 ms, 5]	(medium, medium, medium)
	S3	[6 ms, 500 ms, 2]	(low, high, high)
	S4	[6 ms, 100 ms, 3]	optimized for IPI = 30 s
	S5	[11 ms, 350 ms, 2]	optimized for IPI = 300 s
	S6	[16 ms, 20 ms, 10]	(very high, very low, very low)
LPP	S7	[116 ms, 100 ms, 8]	(high, low, low)
	S8	[266 ms, 250 ms, 5]	(medium, medium, medium)
	S9	[516 ms, 500 ms, 2]	(low, high, high)

Table 2: Static MAC configurations optimized for different performance trade-offs and workloads.

6.2 Model Validation

Before evaluating pTunes under traffic fluctuations, wireless interference, and node failures, we validate our models and assumptions from Sec. 3 on real nodes.

Scenario. We run experiments in which we let pTunes periodically estimate the application-level metrics based on the collected network state, and compare the model estimation $e(M_i)$ against the actual measurement $m(M_i)$ by computing the absolute *model error* $\delta(M_i) = m(M_i) - e(M_i)$ for each metric $M_i \in \{R, L, T\}$. Using δ we assess the model accuracy depending on MAC configuration and network state.

To evaluate the dependency on the former, we use three static MAC configurations for each protocol (S1–S3 and S7–S9 in Table 2). We also perform one run with pTunes enabled, using a TimedTrigger to adapt the MAC parameters every 10 min. To evaluate the dependency on network state, in each run we progressively decrease the inter-packet interval (IPI) at all nodes, from 300 s to 180, 60, 30, 20, 10, 5, and 2 s. In this way, we also validate our models against different probabilities of successful transmission p_l: a shorter IPI increases contention and thus lowers the link success rates. We conduct repeatable experiments by enforcing the same static routing topology across all runs.

Results. Table 3 lists average model errors in R, L, and T for X-MAC and LPP. We see that both models are highly accurate in all metrics. For example, with pTunes enabled, our LPP models estimate R, L, and T with average absolute errors of 0.41 %, 0.08 s, and -0.73 days. Note that node dwell times, which are included in the measurements but ignored in the model of L, introduce only a negligible error since pTunes aims at avoiding packet queuing, as explained next.

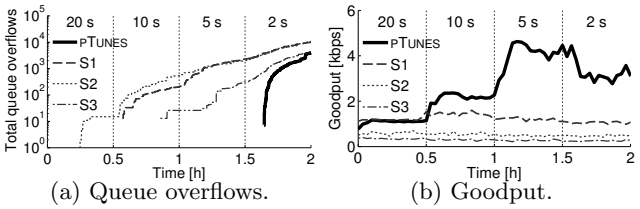
6.3 Impact on Bandwidth and Queuing

Based on the experiments above, we study also the impact of the MAC configuration on bandwidth and local packet queuing. To this end, we analyze queuing statistics collected from the nodes and the goodput measured at the sink (application packets carry 69 bytes of data).

Results. Fig. 6 plots total queue overflows and goodput for X-MAC as the IPI decreases. We can see from Fig. 6(a) that pTunes avoids queue overflows up to IPI = 2 s, whereas S1–S3 fail to prevent overflows already at longer IPIs. The increasing traffic requires more and more bandwidth, leading to local packet queuing and ultimately to queue overflows when the bandwidth becomes insufficient. Unlike S1–S3, pTunes tolerates such increasing bandwidth demands by automatically adjusting the MAC parameters to provide higher bandwidth. By doing so, pTunes avoids the occurrence of queuing until even the MAC parameters providing the highest bandwidth (S6 in Table 2), based on the settings and X-MAC implementation we use, are insufficient.

	X-MAC				LPP			
	S1	S2	S3	pTunes	S7	S8	S9	pTunes
$\delta(R)$ [%]	-0.68	-0.18	0.09	0.24	4.77	-0.22	0.49	0.41
$\delta(L)$ [s]	0.37	0.04	0.18	0.05	-0.12	0.07	0.04	0.08
$\delta(T)$ [d]	0.25	0.64	0.65	-0.50	0.37	-0.91	0.96	-0.73

Table 3: Average absolute errors of the network-wide performance model in testbed experiments, with pTunes and six static MAC configurations. *Our X-MAC and LPP models are highly accurate in all metrics.*



(a) Queue overflows. (b) Goodput.

Figure 6: Queue overflows across all nodes and goodput at the sink with X-MAC as the traffic increases, using pTunes and three static MAC configurations. pTunes *triples the goodput and avoids the occurrence of local packet queuing until the network capacity is fully exhausted.*

This is also confirmed by looking at the goodput, shown in Fig. 6(b). First, we note that pTunes achieves a more than three-fold increase in goodput over S1–S3 at IPI = 5 s. When queuing occurs also with pTunes at IPI = 2 s, goodput drops from 4.6 kbps to 3.1 kbps because increased contention leads to more transmission failures and queue overflows. This confirms that the network capacity is fully exhausted at this point. To keep satisfying the requirements in such situations, an application needs to employ higher-layer mechanisms, such as a rate-controlled transport layer that reduces the transmission rate in response to congestion [30].

6.4 Lifetime Gain

In real deployments, it is common practice to overprovision the MAC parameters based on the highest expected traffic load [23]. The goal is to provide sufficient bandwidth during periods of peak traffic, for example, when an important event causes nodes to temporarily generate more sensor data. However, because such traffic peaks are usually rare and short compared to the total system lifetime, overprovisioning results in a significant waste of resources [24]. We now analyze how pTunes helps alleviate this problem.

Scenario. We conduct two experiments in which nodes gradually increase the IPI from 10 s to 20 s, 30 s, 60 s, 3 min, 5 min, and 20 min. In the first experiment, we use pTunes exactly once at the very beginning to determine MAC parameters optimized for the initial IPI of 10 s, and then keep this overprovisioned MAC configuration until the end of the experiment. In the second experiment, we let pTunes adapt the MAC parameters, using a TimedTrigger with a period of 10 min; pTunes maximizes T subject to $R \geq 95\,\%$ and no constraint on L. We enforce the same static routing topology in both experiments to factor out effects related to routing topology changes, an aspect we consider in Secs. 6.5 and 6.7. We then compute the *lifetime gain* as the ratio between the measured network lifetime with and without pTunes.

Results. Table 4 lists lifetime gains for X-MAC and LPP, *including* the energy overhead of pTunes collection and dis-

Fraction of time at peak traffic (IPI = 10 s)	X-MAC Baseline IPI [min]				LPP Baseline IPI [min]			
	1	3	5	20	1	3	5	20
75%	1.05	1.17	1.24	1.43	1.14	1.27	1.35	1.57
50%	1.14	1.36	1.50	1.88	1.24	1.50	1.65	2.08
25%	1.21	1.55	1.75	2.33	1.33	1.72	1.95	2.60
0%	1.29	1.74	2.01	2.77	1.42	1.94	2.24	3.11

Table 4: Lifetime gains of pTunes over static MAC parameters optimized for peak traffic depending on baseline traffic and fraction of time at peak traffic. pTunes *achieves up to three-fold lifetime gains in settings with extremely rare traffic peaks and low baseline traffic.*

semination phases. We see that the lifetime gain achieved by pTunes increases as (*i*) the system spends less time at peak traffic (75–0 % from top to bottom), and (*ii*) the difference between the shortest, overprovisioned IPI of 10 s and the longest, baseline IPI increases (1–20 min from left to right). For instance, for a baseline traffic at IPI = 20 min and extremely rare traffic peaks at IPI = 10 s, the lifetime gain is close to 2.77 for X-MAC and close to 3.11 for LPP compared to static MAC parameters overprovisioned for peak traffic.

The above experimental results reveal that pTunes enables significant lifetime gains, not least due to its energy-efficient system support (see Sec. 4). The following sections examine how pTunes trades possible gains in network lifetime for satisfying end-to-end reliability and latency requirements under varying network conditions.

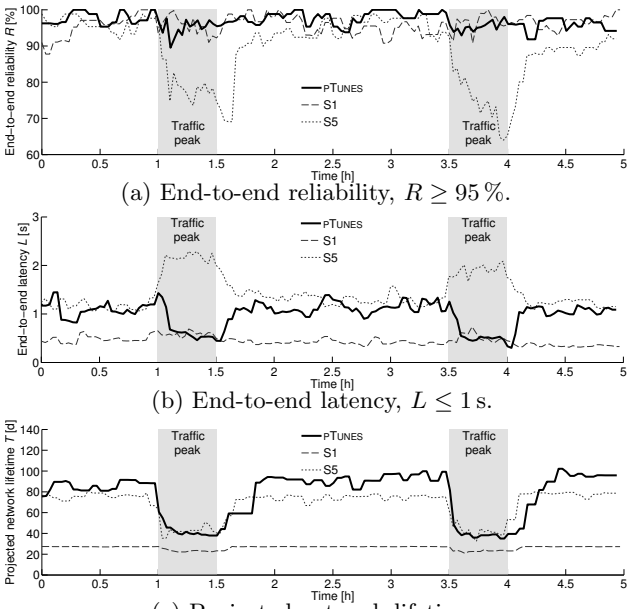
6.5 Adaptation to Traffic Fluctuations

Traffic fluctuations are characteristic of many sensor network applications, where the data rate often depends on time-varying external stimuli. The following experiments investigate the benefits pTunes brings to these applications.

Scenario. All nodes send packets with IPI = 5 min for 5 h. However, during two periods of 30 min each, two clusters of 10 and 5 spatially close nodes (14–23 and 40–44 in Fig. 5) send packets with IPI = 10 s, emulating the detection of an important event that deserves reporting more sensor data.

We run three experiments with X-MAC and dynamic routing using Contiki Collect. In the first two experiments, we use static MAC configurations S1 and S5: S1 provides high bandwidth when nodes send more packets, and S5 extends network lifetime at normal traffic (see Table 2). In the third experiment, we let pTunes adapt the MAC parameters according to (26). We couple a TimedTrigger with a NetworkStateTrigger as follows. When nodes transmit at low rate, the TimedTrigger starts the solver every 10 min. As soon as the NetworkStateTrigger detects the beginning of a traffic peak, it starts the solver immediately and adapts the period of the TimedTrigger to 5 min, setting it back to 10 min at the end of a peak. In this way, pTunes reacts promptly to traffic changes, and adapts more frequently during traffic peaks when nodes report important sensor data.

Results. Fig. 7 plots performance over time in the three experiments. We see that S5 approximately satisfies the reliability and latency requirements when nodes send at low rate, achieving also a high projected network lifetime. However, as soon as the two node clusters start transmitting at high rate, reliability drops significantly below 75 %. This is because S5 does not provide sufficient bandwidth, leading to high contention and ultimately to packet loss. Similarly, S5 violates the latency requirement during traffic peaks, making L exceed 2 s due to queuing and retransmission delays.



(a) End-to-end reliability, $R \geq 95\,\%$.

(b) End-to-end latency, $L \leq 1\,\text{s}$.

(c) Projected network lifetime.

Figure 7: Performance of pTunes against two static MAC configurations as the traffic volume changes. pTunes *satisfies the end-to-end requirements at high traffic while extending network lifetime at low traffic. Static MAC parameters optimized for a specific traffic load fail to meet the application requirements as the traffic conditions change.*

S1, instead, provides sufficient bandwidth and satisfies the end-to-end requirements. However, network lifetime is always below 30 days: the higher bandwidth comes at a huge energy cost, paid also when a lower bandwidth would suffice.

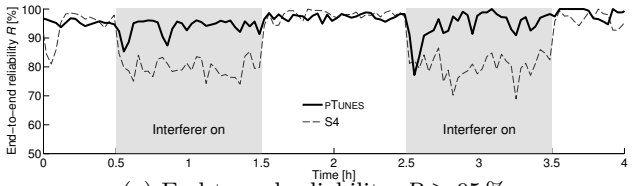
By contrast, pTunes satisfies the end-to-end requirements under high and low rate. Moreover, when nodes transmit at low rate, the projected network lifetime increases up to 90 days. By adapting the MAC parameters, pTunes always provides a bandwidth sufficient to satisfy the end-to-end requirements without sacrificing lifetime unnecessarily: at the beginning of a traffic peak, pTunes reduces T_{off} from about 300 ms to 120 ms (and slightly adapts T_{on} and N), which explains why reliability stays up and latency is halved. Static MAC configurations lack this flexibility; they can only be optimized for a specific workload and thus fail to trade the performance metrics as the traffic conditions change.

6.6 Adaptation to Changes in Link Quality

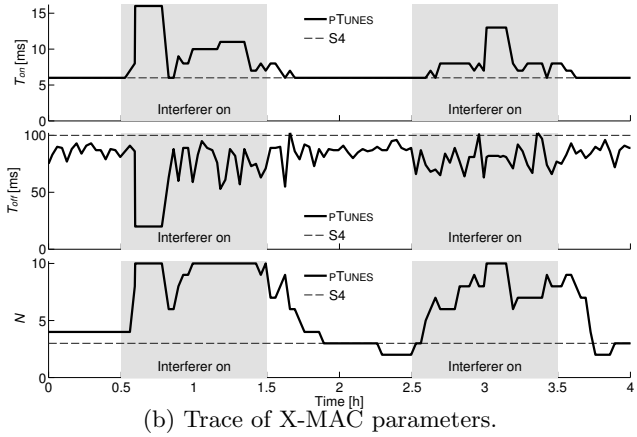
Unpredictable changes in link quality are characteristic of low-power wireless [41]. Adapting the MAC parameters to these changes is important but non-trivial, as we show next.

Scenario. We use the technique by Boano et al. to generate controllable interference patterns [4], making the link quality fluctuate in a repeatable manner. To this end, we deploy an additional interferer node in a position where it affects the communication links of at least one fourth of the nodes in our testbed, as shown in Fig. 5. When active, the interferer transmits a modulated carrier on channel 26 for 1 ms at the highest power setting. Then, it sets the radio to idle mode for 10 ms before transmitting the next carrier.

All nodes generate packets with IPI = 30 s for 4 h. The interferer is active during two periods of 1 h each. In a first experiment, we use static MAC configuration S4, optimized for IPI = 30 s (see Table 2). We enable pTunes in a second

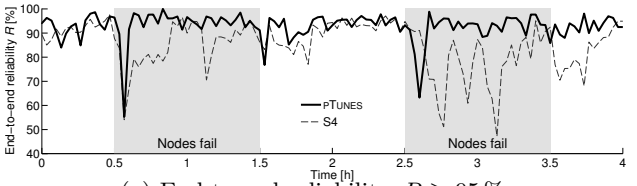


(a) End-to-end reliability, $R \geq 95\,\%$.

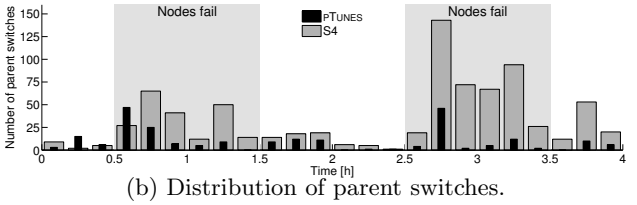


(b) Trace of X-MAC parameters.

Figure 8: End-to-end reliability and trace of X-MAC parameters as the link quality changes. pTunes *reduces packet loss by 80 % during periods of controlled wireless interference in comparison with static MAC parameters optimized for the applied traffic load without interference.*



(a) End-to-end reliability, $R \geq 95\,\%$.



(b) Distribution of parent switches.

Figure 9: End-to-end reliability and distribution of parent switches when eight core routing nodes fail simultaneously. pTunes *helps the routing protocol recover from node failures by settling quickly on a stable routing topology, thus reducing packet loss by 70 % compared with static MAC parameters optimized for the applied traffic load.*

experiment, using a TimedTrigger with a period of 1 min to adapt the MAC parameters according to (26). We deliberately enforce a static routing tree to separate effects related to link quality changes from those related to topology changes. We investigate the latter in detail in Sec. 6.7.

Results. Fig. 8 shows end-to-end reliability and the trace of X-MAC parameters. Looking at Fig. 8(a), we see that S4 and pTunes satisfy the reliability requirement when the interferer is off. When the interferer is on, reliability starts to drop below 95 %. However, as soon as pTunes collects network state, it detects a decrease in link quality and adapts the X-MAC parameters accordingly. In particular, as shown in Fig. 8(b), pTunes increases N from 3 or 4 to values between 6 and 10. T_{on} is also increased (from 6 ms to 10–16 ms) to further help satisfy the reliability requirement. Moreover, pTunes decreases T_{off} (from 100 ms to 20–90 ms) to provide more bandwidth and combat increased channel contention, which is a consequence of numerous retransmission attempts over low-quality links. Indeed, these low-quality links make (26) temporarily unsatisfiable (while $T_{on} = 16$ ms in the first interference phase), triggering pTunes to instead maximize R as explained in Sec. 6.1. As a result of these decisions, pTunes achieves an average end-to-end reliability of 95.4 % also in presence of interference.

S4, instead, fails to satisfy the reliability requirement when the interferer is active: reliability ranges between 70 % and 80 %, and never recovers while the interferer is on. In total, 2252 packets are lost with S4 during interference. pTunes reduces this number to 418—a reduction of more than 80 %.

6.7 Interaction with Routing

Several studies emphasize the significance of cross-layer interactions to the overall system performance [12]. We study this aspect between best-effort tree routing and parameter adaptation of an underlying low-power MAC protocol with pTunes. To do so, during each of the following experiments, we temporarily remove multiple core routing nodes important for forwarding packets. In this way, we emulate node failures, which are common in deployed systems [2], and force the routing protocol to find new routes.

Scenario. We run two 4-hour experiments with Contiki Collect and X-MAC. After 30 min, we turn off eight nodes within the sink's neighborhood that forward most packets in the network (1–8 in Fig. 5). We turn them on again after 60 min, and repeat the on-off pattern after 1 h. Nodes generate packets with IPI = 30 s. In the first experiment, we use static MAC configuration S4, optimized for this traffic load (see Table 2). In the second experiment, we enable pTunes and use a TimedTrigger to solve (26) every minute.

Results. Fig. 9(a) shows end-to-end reliability over time, accounting for packets from nodes that are currently turned on. During the first 30 min, both S4 and pTunes satisfy the reliability requirement. However, when nodes are removed, reliability starts to drop below 70 %. Many packets are indeed lost since children of removed nodes fail to transmit packets: the routing protocol needs to find new routes.

We see from Fig. 9(a) that end-to-end reliability recovers much faster when pTunes is enabled. During the two periods when eight nodes are removed, S4 fails to deliver in total 2673 packets from the remaining 35 nodes. pTunes reduces this number to 813—a reduction of 70 %.

To further investigate this behavior, we plot in Fig. 9(b) the distribution of parent switches. pTunes reduces the total number of parent switches compared to S4 (from 631 to 165), and shifts them to the beginning of the periods in which nodes are removed. At this point, pTunes quickly realizes a significant drop in link quality, reported by nodes whose parent disappeared. pTunes thus increases T_{on} and N to improve reliability, and decreases T_{off} to provide more bandwidth for retransmissions and route discovery.

As a result of increasing the maximum number of retransmissions per packet N, transmission attempts of nodes with a dead parent fail with a higher number of retries. This causes the corresponding ETX values to drop more severely than with S4 (which has a lower N), and so nodes switch

much faster to a new parent. Moreover, the MAC parameters provided by pTunes help deliver packets over the remaining links. Delivering more packets also enables the routing protocol to quickly detect route inconsistencies and eventually settle on a stable topology. As the topology stabilizes, pTunes gradually relaxes the MAC parameters (reduce T_{on} and N, increase T_{off}) to extend network lifetime.

These results demonstrate that, by adapting the MAC parameters, pTunes helps the routing protocol recover faster from critical network changes. Protocols like CTP [18] and Arbutus [33] also utilize feedback from unicast transmissions to compute the ETX. In addition, CTP uses data path validation to detect possible loops based on ETX values embedded in data packets [18]. Our findings with Contiki Collect, which uses similar techniques, suggest that these protocols could also benefit from pTunes.

Additionally, the results demonstrate the advantage of decoupling network state collection from application packet routing, as we argue in Sec. 4.2. As long as the network remains connected, Glossy provides up-to-date network state to the base station with very high reliability [16]. Changes in the routing tree do no affect network flooding: information about faulty links is collected even when the routing protocol fails to deliver packets from nodes whose parent died, allowing pTunes to react promptly and thus effectively.

7. DISCUSSION

Designing a MAC adaptation framework involves striking a balance between goals typically at odds with each other. We discuss in this section some of the trade-offs we make in pTunes and the implications of our particular choices.

Feasibility *vs.* scalability. We adopt a centralized approach rather than a likely more scalable distributed solution; in return for this, pTunes allows users to express their requirements in terms of network-wide metrics, which better reflect the way domain experts are used to state performance objectives compared to per-node or per-link metrics. In fact, distributing the tasks of collecting global state information, computing MAC parameters optimized for network-wide objectives, and coordinating the consistent installation of new parameters would hardly be feasible, if at all, on resource-constrained devices. Instead, pTunes exploits the better resources of a central base station, which is already present in many sensor network deployments [34], and achieves simplicity of in-network functionality by moving most of its intelligence out of the nodes and into the base station.

Flexibility *vs.* optimality. We focus on existing MAC protocols rather than on the design or adaptation of cross-layer solutions (*e.g.*, coupling link and network layer) which may, in principle, achieve better performance; in return for this, pTunes allows system designers to choose the MAC and routing protocol independently from existing code bases. In comparison, cross-layer solutions tend to enjoy little generality and flexibility, as they are often designed for very specific scenarios (*e.g.*, periodic, low-rate data collection [6]).

Robustness *vs.* optimality. We determine network-wide parameters rather than per-node parameters, which may better match the current role of a node in the routing tree (*e.g.*, with respect to traffic load); in return for this, the parameters pTunes provides are much more robust to changes in the routing topology. It is not unlikely that, even in the most benign environment, slight variations in the link qualities trigger drastic changes in the routing topology. For instance, Ceriotti et al. observe that nodes serving many children suddenly become leaves in the routing tree [7]. In such a case, per-node MAC parameters become inappropriate and must be quickly updated. Similar situations can happen frequently, even several times per minute [19], which would render per-node parameter adaptation impractical.

As a consequence of the design decisions above, pTunes represents one particular point in a multi-dimensional design space. Corresponding to this point is a large fraction of deployed low-power wireless networks comprising tens of nodes, leveraging protocols such as X-MAC and LPP, and yet failing to meet the application requirements often due to communication issues ultimately related to inadequate MAC parameter choices and lack of adaptiveness [23,34]. pTunes is directly and immediately applicable in these settings.

8. RELATED WORK

pTunes uses a model to predict how changes in the MAC parameters affect the network-wide performance given the current network state. Based on iterative runtime optimization, it selects MAC parameters such that the predicted performance satisfies the application requirements. This approach is similar to the concept of model predictive control (MPC) [17], with the differences that pTunes computes only the next step of the control law and uses no information about past control steps or measured system responses.

Several recent systems incorporate centralized control in their design, much like pTunes does. For example, Koala implements a network-wide routing control plane, where the base station computes end-to-end paths used for packet forwarding [29]. RACNet uses centralized token passing to sequence data downloads [26]. In RCRT, the sink detects congestion and adapts the rates of individual sources [30]. PIP determines schedule and channel assignment for each flow centrally at the base station [35]. Like pTunes, these systems exploit global knowledge and ample resources of the base station to achieve high performance and manageability.

Looking at the large body of prior work on adaptive low-power MAC protocols, we find solutions embedding adaptivity or separating adaptivity from the protocol operation.

In the former category, for instance, Woo and Culler propose an adaptive rate control mechanism, where nodes inject more packets if previous attempts were successful and fewer packets if they failed [39]. Van Dam and Langendoen introduce an adaptive listen period in T-MAC [11] to overcome the drawbacks of the fixed duty cycle of S-MAC [40]. El-Hoiydi and Decotignie adapt radio wake-ups in WiseMAC to shorten the LPL preamble [15]. More recently, Hurni and Braun propose MaxMAC, which schedules additional X-MAC wake-ups at medium traffic and switches to pure CSMA at high traffic [21]. Such hard-coded adaptivity mechanisms can be highly effective in specific scenarios, but lack general applicability and bear no direct connection to the high-level application demands. pTunes is more general by adding parameter adaptation atop existing MAC protocols, thus leveraging available implementations, and by explicitly incorporating user-provided application requirements.

Polastre et al. instead separate adaptivity from the protocol operation and present a model of node lifetime for B-MAC [32]. Jurdak et al. use this model to dynamically recompute check interval and preamble length, showing substantial energy savings [22]. Buettner et al. demonstrate en-

ergy savings in X-MAC by adapting the wake-up interval to traffic load for one sender-receiver pair [5]. Meier et al. [28] and Challen et al. [9] extend network lifetime by adjusting the wake-up interval to traffic load in a static routing tree. Park et al. present numerical results that indicate the potential of adaptation policies for IEEE 802.15.4 MAC protocols, based on per-link and per-node metrics [31]. pTunes builds on these foundations but extends them in several ways. First, pTunes considers multiple network-wide metrics and adapts multiple MAC parameters. Second, our modeling is more realistic by accounting for packet loss and ARQ mechanisms, and more flexible by isolating protocol-dependent from protocol-independent functionality. Third, we evaluate pTunes in real-world scenarios, including dynamic routing trees, wireless interference, and node failures.

9. CONCLUSIONS

pTunes provides runtime parameter adaptation for low-power MAC protocols, automatically translating application-level requirements into MAC parameters that meet these requirements and achieve very good performance across a variety of scenarios, ranging from low traffic to high traffic, from good links to bad links, and wireless interference to node failures. pTunes thus greatly aids in meeting the requirements of real-world sensor network applications by eliminating the need for time-consuming, and yet error-prone, manual MAC configuration when the network conditions change.

Acknowledgments. The authors thank Renato lo Cigno, Kay Römer, Olga Saukh, and the anonymous reviewers for their insightful comments. This work was supported by Nano-Tera, the National Competence Center in Research on Mobile Information and Communication Systems under SNSF grant number 5005-67322, the Swedish Foundation for Strategic Research, and the Cooperating Objects Network of Excellence under contract number EU-FP7-2-224053.

10. REFERENCES

[1] K. R. Apt and M. G. Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, 2007.
[2] J. Beutel et al. PermaDAQ: A scientific instrument for precision sensing and data recovery under extreme conditions. In *ACM/IEEE IPSN*, 2009.
[3] J. Beutel et al. Poster abstract: The FlockLab testbed architecture. In *ACM SenSys*, 2009.
[4] C. A. Boano, T. Voigt, N. Tsiftes, L. Mottola, K. Römer, and M. Zuniga. Making sensornet MAC protocols robust against interference. In *EWSN*, 2010.
[5] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks. In *ACM SenSys*, 2006.
[6] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low power data gathering in sensor networks. In *ACM/IEEE IPSN*, 2007.
[7] M. Ceriotti et al. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *ACM/IEEE IPSN*, 2009.
[8] M. Ceriotti et al. Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels. In *ACM/IEEE IPSN*, 2011.
[9] G. W. Challen, J. Waterman, and M. Welsh. IDEA: Integrated distributed energy awareness for wireless sensor networks. In *ACM MobiSys*, 2010.
[10] J. Choi, M. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In *ACM SenSys*, 2009.
[11] T. Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *ACM SenSys*, 2003.
[12] S. R. Das, C. E. Perkins, and E. M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *IEEE INFOCOM*, 2000.
[13] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *ACM SenSys*, 2010.
[14] M. Dyer et al. Deployment support network: A toolkit for the development of WSNs. In *EWSN*, 2007.
[15] A. El-Hoiydi and J.-D. Decotignie. WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks. In *ALGOSENSORS*, 2004.
[16] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *ACM/IEEE IPSN*, 2011.
[17] C. E. Garciá, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3), 1989.
[18] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *ACM SenSys*, 2009.
[19] O. Gnawali, L. Guibas, and P. Levis. A case for evaluating sensor network protocols concurrently. In *ACM WiNTECH*, 2010.
[20] Y. Y. Haimes, L. S. Lasdon, and D. A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Trans. Syst., Man, Cybern.*, 1(3), 1971.
[21] P. Hurni and T. Braun. MaxMAC: A maximally traffic-adaptive MAC protocol for wireless sensor networks. In *EWSN*, 2010.
[22] R. Jurdak, P. Baldi, and C. V. Lopes. Adaptive low power listening for wireless sensor networks. *IEEE Trans. Mobile Comput.*, 6, 2007.
[23] R. Kuntz, A. Gallais, and T. Noel. Medium access control facing the reality of WSN deployments. *ACM SIGCOMM Comp. Comm. Rev.*, 39(3), 2009.
[24] K. Langendoen and A. Meier. Analyzing MAC protocols for low data-rate applications. *ACM Trans. on Sens. Netw.*, 7(2), 2010.
[25] P. Levis et al. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *USENIX NSDI*, 2004.
[26] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. RACNet: A high-fidelity data center sensing network. In *ACM SenSys*, 2009.
[27] R. Madan and S. Lall. Distributed algorithms for maximum lifetime routing in wireless sensor networks. *IEEE Trans. Wireless Commun.*, 5(8), 2006.
[28] A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele. ZeroCal: Automatic MAC protocol calibration. In *IEEE DCOSS*, 2010.
[29] R. Musaloiu-E., C.-J. M. Liang, and A. Terzis. Koala: Ultra-low power data retrieval in wireless sensor networks. In *ACM/IEEE IPSN*, 2008.
[30] J. Paek and R. Govindan. RCRT: Rate-controlled reliable transport for wireless sensor networks. In *ACM SenSys*, 2007.
[31] P. Park, C. Fischione, and K. Johansson. Adaptive IEEE 802.15.4 protocol for energy efficient, reliable and timely communications. In *ACM/IEEE IPSN*, 2010.
[32] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *ACM SenSys*, 2004.
[33] D. Puccinelli and M. Haenggi. Reliable data delivery in large-scale low-power sensor networks. *ACM Trans. on Sens. Netw.*, 6(4), 2010.
[34] B. Raman and K. Chebrolu. Censor networks: A critique of "sensor networks" from a systems perspective. *ACM SIGCOMM Comp. Comm. Rev.*, 38(3), 2008.
[35] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale. PIP: A connection-oriented, multi-hop, multi-channel TDMA-based MAC for high throughput bulk transfer. In *ACM SenSys*, 2010.
[36] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *ACM SenSys*, 2004.
[37] G. Tolle et al. A macroscope in the redwoods. In *ACM SenSys*, 2005.
[38] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *USENIX OSDI*, 2006.
[39] A. Woo and D. Culler. A transmission control scheme for media access in sensor networks. In *ACM MobiCom*, 2001.
[40] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *IEEE INFOCOM*, 2002.
[41] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *ACM SenSys*, 2003.
[42] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. pTunes: Runtime parameter adaptation for low-power MAC protocols. Technical Report 325, ETH Zurich, 2012.