Duty Cycle Control for Low-Power-Listening MAC Protocols

Christophe J. Merlin and Wendi B. Heinzelman
Department of Electrical and Computer Engineering,
University of Rochester, Rochester NY
{merlin, wheinzel}@ece.rochester.edu

***Abstract*—Energy efficiency is of the utmost importance in wireless sensor networks. The family of *Low-Power-Listening* MAC protocols was proposed to reduce one form of energy dissipation—idle listening, a radio state for which the energy consumption cannot be neglected. Low-Power-Listening MAC protocols are characterized by a duty cycle: a node probes the channel every t_i s of sleep. A low duty cycle favors receiving nodes because they may sleep for longer periods of time, but at the same time, contention may increase locally, thereby reducing the number of packets that can be sent. We propose two new approaches to control the duty cycle so that the target rate of transmitted packets is reached, while the consumed energy is minimized. The first approach, called asymmetric additive duty cycle control (*AADCC*), employs a linear increase / linear decrease in the t_i value based on the number of successfully received packets. This approach is easy to implement, but it cannot provide an ideal solution. The second approach, called dynamic duty cycle control (*DDCC*) utilizes control theory to strike a near-optimal balance between energy consumption and packet delivery successes. We generalize both approaches to multi-hop networks. Results show that both approaches can appropriately adjust t_i to the current network conditions, although the dynamic controller (DDCC) yields results closer to the ideal solution. Thus, the network can use an energy saving low duty cycle, while delivering up to four times more packets in a timely manner when the offered load increases.**

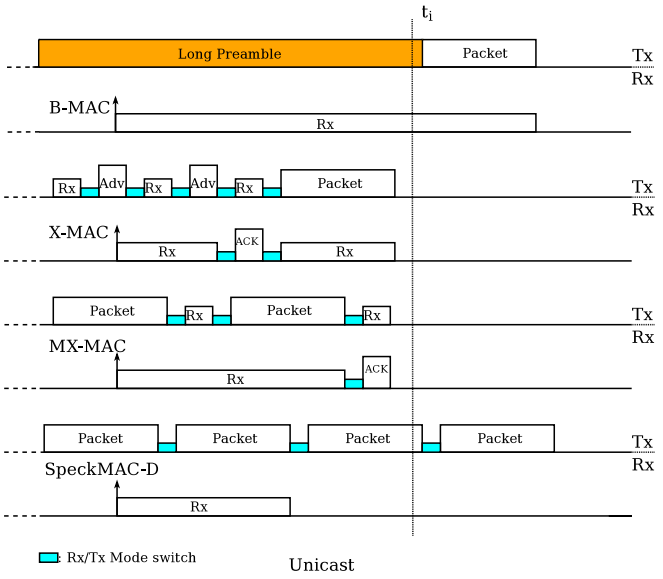


Fig. 1. Schedule for B-MAC, X-MAC, MX-MAC and SpeckMAC-D.

I. Introduction

Today more than ever, sensor network applications require individual nodes to lower their energy consumption in order to support an application for longer periods of time. Every layer in the protocol stack must reduce its own energy dissipation. *Low-Power-Listening* (*LPL*) protocols form a family of MAC protocols that drastically reduce idle listening, a state of the node when its radio is turned on and in receive mode, but not receiving any packets.

In a LPL protocol, nodes probe the channel every t_i s, and if they do not receive any data during this probe, they return to sleep for another t_i s. Aloha with preamble sampling (PS) [1], WiseMAC [2], and B-MAC [3] were among the first random access MAC protocols to be proposed[1]. All these protocols send data packets with very long preambles so as to ensure that the intended receiver will stay on upon probing the medium. However, the protocols are not adapted to recent radios like the IEEE 802.15.4 [5] compliant Chipcon CC2420 [6] radio. Consequently, researchers introduced new compatible LPL protocols such as X-MAC [7], SpeckMac-D [8], and MX-MAC [9]. These protocols are based on repeating either the data packet itself or an advertisement packet, in place of long preambles. The transmission schedules (hereafter "MAC schedule") of some LPL MAC protocols are given in Figure 1.

Previous work [9] has shown that, along a one-hop link, longer t_i values favor receiving nodes, because longer t_i values lower a node's duty cycle while switching to *Receive* mode for the same period of time within the duty cycle. On the other hand, nodes that are mostly sending can greatly reduce their energy consumption if the t_i value is low: they can stay in *Sending* mode for shorter periods of time. Consequently, there is a trade-off between the nodes at the two ends of a unidirectional wireless link. In addition, lower duty cycles often cause contention in areas of the network experiencing higher rates of packet transmissions. As Figure 1 shows, only one data packet can be transmitted per cycle, which can cause a node to miss the target rate m^* of packet transmissions.

In [10], Jurdak et al. convincingly argue that a fixed t_i value does not fit WSN deployments where the node locations and

[1]In his taxonomy of MAC protocols [4], Langendöen identifies Low-Power-Listening and Preamble Sampling protocols as two branches of random access MAC protocols, with the only difference that LPL MAC protocols need not know anything about their neighbors and their wake-up schedules.

traffic patterns are not uniform over the network. Because a fixed t_i value is decided *a-priori*, it would have to be set conservatively to accommodate areas in the network where traffic is expected to be heavy, thus forcing idle subregions to waste energy.

In this paper, we propose two adaptive solutions to adjust the duty cycle. The first one is an intuitive linear increase / linear decrease scheme (AADCC). The second one (DDCC) borrows from control theory to dynamically adjust the duty cycle of the nodes based on a small set of parameters. We begin with one-hop networks. The goal of our methods is to minimize the energy consumed by the node with the lowest remaining energy (or the node which the application deems most important), referred to as node $\mathcal{N}$, while exchanging a target number of packets. If $\mathcal{N}$ is mostly sending, lowering t_i (increasing the duty cycle) will have no adverse effect on the target rate m^* of successfully sent packets, and it will reduce the energy dissipation for $\mathcal{N}$, so there is no need for t_i control. However, when $\mathcal{N}$ is mostly receiving, lowering the duty cycle (increasing t_i), while reducing the energy dissipation for $\mathcal{N}$, will cause packets to be dropped. This is the conflict that we propose to arbitrate. DDCC can also be extended to control the energy consumed by both the sending and receiving nodes on a wireless link. More generally, we provide a methodological framework that can be applied to control other aspects of the network as well.

We generalize both methods to multi-hop networks, starting with only one data source, as is often the case when source selection is performed. For the dynamic controller (DDCC), we must utilize a path synchronization scheme that, among other many benefits, reestablishes linearity in the system. We then lift the last restriction (only one source) through the study of t_i control for multi-hop networks with several sources. To successfully control the duty cycle with several sources, improvements to an existing path synchronization technique are introduced to support multiple branches.

The remainder of this paper is organized as follows. Section II presents related work. Section III introduces the AADCC protocol and the theoretical foundations for DDCC and expands on these to adapt to our specific problem of t_i control for channel-probing MAC protocols in a one-hop network. Section IV presents simulation results using both schemes AADCC and DDCC. Section V expands these results to single-branch multi-hop networks. Section VI lifts this last constraint and presents results showing successful t_i control of a multi-hop network with several sources. Section VII concludes this paper and discusses the results.

II. Related Work

The low-power-listening family is composed of many MAC protocols [3] [7] [8] [9] [11]. All these protocols are characterized by a trade-off in terms of energy savings for the sending and receiving nodes, although to varying degrees that depend on each individual MAC schedule. WiseMAC [2], a Preamble Sampling MAC, is a related channel probing protocol, but it is not part of the narrower LPL family. Nodes running WiseMAC must exchange scheduling information so that a node with packets to send can start transmitting a short time before its intended receiver wakes up. In this work, we consider only LPL MAC protocols for their capacity to synchronize along slowly changing paths.

While the idea of dynamic duty cycling for MAC protocols was explored by Lin et al. [12], Jurdak et al. [10] introduced the idea of adaptive duty cycles in LPL protocols. Because a protocol designer must account for busy regions of the network, a fixed t_i value would have to be set conservatively. Consequently, many parts of the network would waste energy by running at an unnecessarily high duty cycle. Adaptive Low-Power-Listening, or ALPL, allows areas of the network to run at a lower duty cycle. After forming their routing tree, each individual node can evaluate the number of packets they will transmit per second based on the expected number of packets they and their descendant nodes will originate. These values are periodically announced by the nodes. The further away from the data sink, the fewer children a node has, and consequently, the smaller the packet rate it is expected to carry. Its duty cycle can thus be lowered to a smaller value than that of nodes closer to the data sink. Contrary to ALPL, our approaches do not use a heuristic and DDCC adapts the duty cycle to meet the target rate of packets.

The idea of using control theory in sensor networks is not a new one, especially because wireless sensor and actuator networks require such solutions. In our unique approach, DDCC optimizes the duty cycle for both energy use and packet transmissions, which cannot be easily modeled. Examples of existing methods that use results of control theory to adapt parameters in a WSN include [13] and [14].

In [13], Vigorito et al. use control theory to adapt the duty cycle of nodes capable of harvesting energy. Maintaining a sufficient power supply level is a non-trivial problem because of changing environmental patterns such as the weather. The authors introduce a model-free approach to adapt the duty cycle in dynamic conditions. Although they set out to control only one parameter in the system (the energy supply level), which constitutes a marked difference from our goals, much of their underlying theoretical foundations are similar to those in the first part of our work.

In [14], Le et al. propose to optimize channel assignment to increase the throughput in multi-channel WSNs using a control theory approach. The throughput on individual channels can be easily modeled with the nodes' individual load, which includes that of its descendant nodes. When the total load M_i on channel i is above its optimal value M_r (one that guarantees little contention for instance), nodes transmitting on this channel may change to another channel $j \neq i$ with a probability proportional to the difference (error) between M_r and M_i. Le et al. also account for delay, which can cause overshooting and undershooting—instability of the system response.

III. Estimation and Control for Multi-variable Systems

Because low duty cycle schemes tend to create contention and delays, a node wishing to send m^* packets may not be able

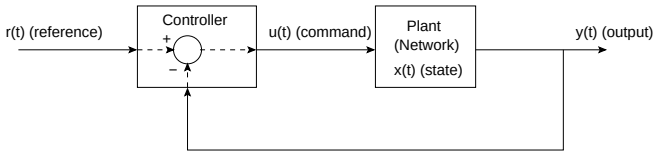


Fig. 2. Representation of the system with input / output and its controller.

to do so in a timely manner. Let us consider a one-hop network with various flows among neighbors. Node A wants to send m^* packets to node B in a certain time period $\mathcal{T}$, where node B is designated as node $\mathcal{N}$, a critical node for the application, or one with very low remaining energy. Unfortunately, the medium is sometimes occupied by other transmissions. If node A only gets to send $m < m^*$ packets, it may elect to increase its duty cycle. When the duty cycle is larger than its optimal value, node $\mathcal{N}$ wastes precious energy, and may wish to scale back its duty cycle (t_i increased). The control of the duty cycle to send m^* packets is the subject of this section. We use $t_i(t)$ to designate the time-varying nature of t_i.

A. Asymmetric Additive Duty Cycle Control

The first proposed scheme is called asymmetric additive duty cycle control (AADCC). Protocol designers could easily find inspiration in the adaptive back-off scheme of the 802.11 MAC protocol. We chose to design our adaptive duty cycle control based on the number of consecutive packet transmissions. While 802.11 employs a multiplicative increase / linear decrease back-off, multiplicative increase turned out to be too disruptive in duty cycle control tests that we ran over the full range of t_i values. Thus, we made our additive controller an asymmetric linear increase / linear decrease scheme. Whenever five consecutive packets are successfully sent to the destination, t_i is increased by $100\ ms$, or $t_i((k+1)\mathcal{T}) = t_i(k\mathcal{T})+0.1$. Each failed packet is followed by a decrease of $250\ ms$ in t_i, or $t_i((k+1)\mathcal{T}) = t_i(k\mathcal{T}) - 0.25$. While this simple additive controller can produce better results than a static t_i value, it cannot provide the optimal solution, as it does not consider energy dissipation or even try to approach a target m^*. Therefore, our second scheme, DDCC, is based on control theory to optimize t_i such that the target number of packets is sent while reducing energy dissipation.

B. Background for Dynamic Control

Here we provide the mathematical background for our dynamic controller.

1) Generalities: We start by assuming that the system we wish to represent and control is mostly linear. For instance, the relationship between energy consumption and t_i is linear, as energy consumption grows linearly with the number of probes done per second. Likewise, the number of packets received is mostly linearly related to energy consumption.

Figure 2 illustrates the system at hand. The network is represented by a "plant" that reacts to an input $u(t)$ by producing an output $y(t)$, which it tries to match to a reference $r(t)$. A controller modifies $u(t)$ so as to obtain the desired output $y^*(t) = r(t)$. In order to do so, the process under control can be defined by its state $x(t)$. A deterministic noisy linear process can be represented in its discrete form as follows:

$$x(t+1) = Ax(t) + Bu(t) + Cw(t) + w(t+1) \quad (1)$$

where $x(t+1)$ designates the value of the system state at time $(k+1)\mathcal{T}$ and w is the noise. $\mathcal{T}$ represents the period between re-evaluations of the control $u(t)$.

For controlling $t_i(t)$, we can set $y(t)$ to $m(t)$ (the number of packets that are successfully sent at time t) and $u(t)$ to the $t_i(t)$ value at time t. The objective value $y^*(t+1)$ becomes $m^*(t+1)$, the desired number of packets to be transmitted at time $t+1$.

Because the fundamental characteristics of the system (A, B and C) and its state $x(t)$ cannot be *a-priori* known, the system's output must be estimated using an internal parameter θ and a history of p values of $\{x(t)\}$ (or $\{y(t)\}$) and $\{u(t)\}$ values stored in ϕ.

C. The Dynamic Regulator

In this first part of our work, we would like to control t_i to send the target number of packets m^*. We introduce a SISO (single variable) estimator and controller.

1) Stochastic SISO Estimator and Controller: We begin with the formulation of our goal, *i.e.* the minimization of the expected error between the desired output at time $t+1$, $y^*(t+1)$ and the actual output at time $t+1$, $y(t+1)$, which is mathematically represented by the following:

$$J = E[(y(t+1) - y^*(t+1))^2] \quad (2)$$

This control problem is referred to as linear-quadratic: the system dynamics are linear (Equation 1), but the cost function to be minimized (Equation 2) is quadratic. Because the system response contains a random component (the exact wake-up timing between two neighbors), we study a system estimator and controller for the stochastic case.

First, and as suggested in [15], we introduce the following notation for time delay:

$$x(t-1) = q^{-1}x(t)$$

We can write the system as:

$$y(t) = ay(t-1) + bu(t-1) + cw(t-1) + w(t) \quad (3)$$

$$\Leftrightarrow (1 - aq^{-1})y(t) = bq^{-1}u(t) + (1 + cq^{-1})w(t) \quad (4)$$

From [15], Equation 3 can be put in the form:

$$C(q^{-1})y^0(t+1|t) = \alpha(q^{-1})y(t) + \beta(q^{-1})u(t) \quad (5)$$

where

$$\begin{cases} C(q^{-1}) = 1 - aq^{-1} + q^{-1}g_0 = 1 + (g_0 - a)q^{-1} \\ \alpha(q^{-1}) = g_0 \\ \beta(q^{-1}) = b \end{cases}$$

and y^0 represents the next value taken by y and g_0 is a constant.

The control law is thus shown to be:

$$u(t) = \frac{y^*(t+1) + (g_0 - a)y^*(t) - g_0 y(t)}{b \neq 0}$$

Let $g_0 - a = c$,

$$u(t) = \frac{y^*(t+1) + cy^*(t) - (a+c)y(t)}{b} \quad (6)$$

which is also the control law used in [13]. It follows easily that Equation 6 minimizes the mean-square error function J.

Next, we define the ϕ and θ vectors as:

$$\phi(t)^T \theta(t) = \hat{y}(t+1)$$

where $\hat{y}(t+1)$ is the estimated system output at time $t+1$. As a starting point, we chose to keep only the previous values of the input and output, or $p = 1$. From Equation 6, we use the two vectors:

$$\phi(t) = \begin{bmatrix} y(t) \\ u(t) \\ y^*(t) \end{bmatrix} \quad \theta(t) = \begin{bmatrix} a+c \\ b \\ -c \end{bmatrix}$$

The estimator can be computed using the Normalized Least-Mean-Square Algorithm (NLMS) [15] [16]:

$$\theta(t+1) = \theta(t) + \frac{\mu(t)\phi(t)}{\phi(t)^T\phi(t) + \omega}[y(t+1) - \phi(t)^T\theta(t)] \quad (7)$$

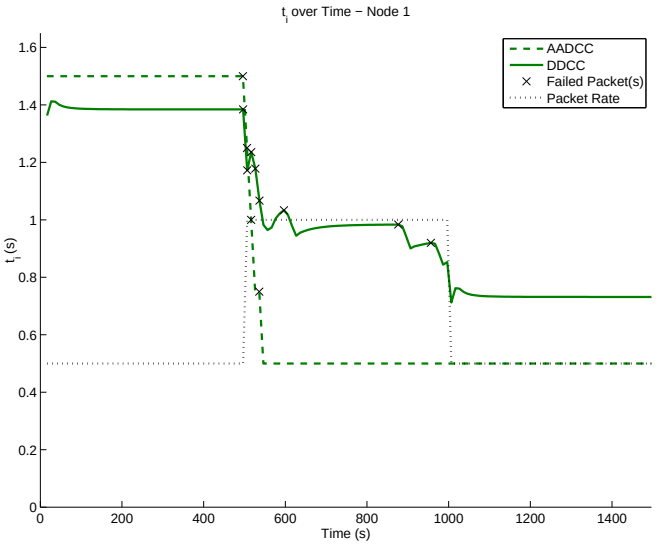
where $\mu(t)$ is a scalar, and ω should be chosen to avoid a division by zero when $\phi(t)^T\phi(t)$ is null. With our notations, $\phi(t)$ is thus the values of the output $y(t) = m(t)$, the command $u(t) = t_i(t)$ and the target $y^*(t) = m^*(t)$. The tuple $\{a, b, c\}$ is estimated using Equation 7.

New t_i values are computed periodically. During each round (of duration $\mathcal{T}$), the number of packets successfully transmitted since the last t_i update (i.e., $m(t)$) is recorded.

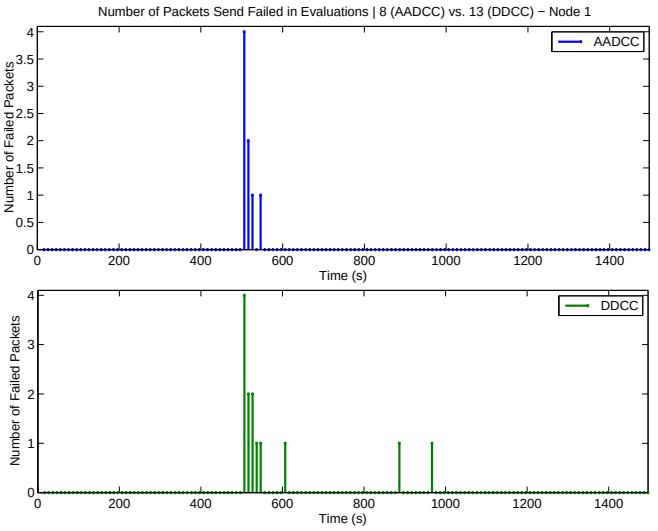
2) Application to Our Estimator: The system control can be approached by estimating the system first, and using the system model to find the input value that minimizes the predicted output.

Preliminary results show that, while the estimator is able to correctly predict the system output, the control law tends to decrease the value of $t_i(t)$ when $m < m^*$. This behavior is in fact to be expected as the system should decrease its duty cycle to increase the number of packet transmissions. Unfortunately, since J carries no consideration for energy use, t_i never increases, even after the number of packets to be sent has reached the target ($m = m^*$). The reason is that the error between $\hat{m}$ and m^* is zero, which does not modify the value of the controlled input $u(t) = t_i(t)$. Figure 3(a) illustrates this problem. At $t = 500\ s$, the packet rate increases to one packet per second, causing t_i to decrease due to packet losses. However, a few seconds later, the packet rate decreases to its original value of 0.5 packet per second, yet t_i does not increase again.

Figure 3(b) shows the packet loss in this scenario, where the number of dropped packets is 13 for the dynamic controller (DDCC), and 8 for the additive controller (AADCC). The number of dropped packets is higher for the dynamic controller because it did not increase its duty cycle as aggressively. However, it preserved more of the nodes' energy (around 10%, not shown). Compared to a case without any duty cycle control (not shown), the number of dropped packets is reduced by over 94% by the duty cycle controller.



(a)



(b)

Fig. 3. (a) Evolution of $t_i(t)$ as the packet rate increases and then decreases when only packet loss is considered. (b) Packet loss in the same scenario.

Consideration must be given to the energy consumed ϵ, which is an incentive to lower the duty cycle. We note $\epsilon^*(t+1)$ as a target energy consumption at $t+1$.

In this now multi-variable case, we decided to estimate both the number of packets sent and the consumed energy separately. For m and ϵ, the ϕ and θ vectors are:

$$\phi_k^m = \begin{bmatrix} m_k & \dots & m_{k-p} & t_{ik} & \dots & t_{ik-p} & m_k^* & \dots & m_{k-p}^* \end{bmatrix}^T$$
$$\theta_k^m = \begin{bmatrix} a_0^m & \dots & a_{p-1}^m & b_0^m & \dots & b_{p-1}^m & c_0^m & \dots & c_{p-1}^m \end{bmatrix}^T$$
$$\phi_k^\epsilon = \begin{bmatrix} \epsilon_k & \dots & \epsilon_{k-p} & t_{ik} & \dots & t_{ik-p} & m_k^* & \dots & m_{k-p}^* \end{bmatrix}^T$$
$$\theta_k^\epsilon = \begin{bmatrix} a_0^\epsilon & \dots & a_{p-1}^\epsilon & b_0^\epsilon & \dots & b_{p-1}^\epsilon & c_0^\epsilon & \dots & c_{p-1}^\epsilon \end{bmatrix}^T$$

where $a, b, c \in \mathbb{R}$ are the estimator coefficients. We chose $p \gtrapprox 3$, a value that allows the estimate for ϵ and m to be accurate, while being still manageable in limited memory space.

3) Cost Minimization: As per Section III-C2, the controller should minimize a cost function with a packet loss and an energy component. We tried to combine the two costs in various ways, including taking the maximum, the sum, and the

weighted sum of the costs. The latter offered the swiftest and most stable response from the network. Thus, the controller attempts to minimize the following cost function J:

$$J = (m^{*+} - \hat{m}_{k+1})^2 + K_\epsilon(\epsilon^{*+} - \hat{\epsilon}_{k+1})^2 \quad (8)$$

where $\begin{cases} \hat{m}_{k+1} = \phi_k^{mT}\theta_k^m \\ \hat{\epsilon}_{k+1} = \phi_k^{\epsilon T}\theta_k^\epsilon \end{cases}$, m^{*+} and ϵ^{*+} designate the target values of m and ϵ at time $(k+1)\mathcal{T}$. K_ϵ is a weight given to the energy component of the cost function in order to indicate a preference to save energy (large K_ϵ) or to strictly meet the number of packets to be sent (small K_ϵ); for instance, K_ϵ can be chosen in $[2; 20]$. The control law finds the value of t_i that minimizes J.

Taking the derivative of J at time $k\mathcal{T}$ (we omit the k index notation for clarity), we obtain Equation 6 for our application:

$$t_i = \frac{\theta_p^m(m^{*+} - \sum_{i\neq p}^{u}\phi_i^m\theta_i^m) + K_\epsilon\theta_p^\epsilon(\epsilon^{*+} - \sum_{i\neq p}^{v}\phi_i^\epsilon\theta_i^\epsilon)}{(\theta_p^m)^2 + K_\epsilon(\theta_p^\epsilon)^2}$$

where the i-index value on ϕ_i and θ_i are the i^{th} value of these vectors, and u and v are the number of elements in ϕ_k^m and ϕ_k^ϵ ($u = 2p$ and $v = 3p$).

In order to smooth the response of the system, we adopt a conservative update policy $\bar{u}$ for the duty cycle with the following set of rules:

$$\begin{cases} \bar{t}_{ik+1} = \bar{t}_{ik} + \alpha(t_i - \bar{t}_{ik}) \\ \bar{u}_{k+1} = f_\delta^\Delta[\bar{t}_{ik+1}] \end{cases} \quad (9)$$

where $\bar{t_i}$ is the smoothed t_i and

$$f_\delta^\Delta[x] = \begin{cases} \delta & \text{if } x < \delta \\ \Delta & \text{if } x > \Delta \\ x & \text{otherwise} \end{cases}$$

δ and Δ are the minimum and maximum values that t_i can ever take, and can be set to $0.1\ s$ and $5\ s$ as reasonable values.

$\alpha \in \mathbb{R}$ is the slope of the update of t_i and helps stabilize the system response, which would otherwise be unstable because of steep variations of the reference $r(t)$ (the desired number of packets for instance) and delays in the feedback. A large α (*i.e.*, close to 1) aggressively updates t_i and incurs oscillations before reaching a determined value. On the other hand, if α is close to 0, no oscillations can be discerned but t_i is slow to reach its eventual value. Poor choices of α may cause energy waste or packet loss. The command used to control the duty cycle is in fact $\bar{u}$ as a smoothed output is critical to a physical network.

D. Evaluating the Target Energy

1) The Evaluation of ϵ and ϵ^*: In some cases, the system designer may want to minimize the consumed energy and choose $\epsilon^* = 0$. The risk incurred by this approach is that the duty cycle will tend to be lowered, even below a reasonable value—one that strikes a balance between the number of lost packets and energy consumption. This could be desirable when designing a system that needs to respond faster to lower energy consumption, and that can tolerate repeated packet losses.

In other systems, an acceptable energy consumption value has to be evaluated so that $t_i(t)$ does not consistently increase past a reasonable value. This target energy has critical importance as the system will have a tendency to stabilize around the value of t_i that yields this energy consumption, provided all packets are correctly sent. The control problem thus becomes a linear quadratic tracking ("LQ tracking") problem where the output of the network must match the energy (and packet delivery) reference.

We chose to evaluate the target energy as the sum of several basic operations (channel probe, packet reception, etc.) for which we precisely measured the energy consumption via a data acquisition board on the Tmote Sky platform. We evaluate the target energy as the minimal energy that can be expended during a round of $\mathcal{T}$ seconds:

$$\epsilon^* = max(0, m^* E[E_{Rx}] + E_{PD}(t_\mathcal{T} - m^* E[t_{Rx}])) \quad (10)$$

where $E[E_{Rx}]$ is the expected energy spent to receive a packet, E_{PD} is the energy consumed by the radio for one second of power down mode, and $t_\mathcal{T}$ is the duration of a feedback round $\mathcal{T}$. The target energy ϵ^* assumes that each packet is sent every $t_i\ s$, and that no energy is wasted on probing a clear channel. It contains no information about other transmissions in the neighborhood as packet loss is taken into account in the first element of J.

2) An Alternative Solution to Evaluating the Consumed Energy: Because it may be impractical to evaluate the energy consumption components ϵ and ϵ^*, an alternative solution is to use the linear increase of AADCC. This reduces the complexity of the system to only one component m.

The relative simplicity of linear increase is offset by the slower nature of the response to increase t_i when the data load diminishes. For DDCC in general, we prefer evaluating the energy, but we show in our later results that such a method where energy is not evaluated does provide satisfactory results.

E. Dynamic Duty Cycle Control Algorithm

The controller described in the previous theoretical foundations is called Dynamic Duty Cycle Control (DDCC). Algorithm 3.1 presents the pseudo-code of DDCC. The initialization of the algorithm variables includes assigning a starting value to the ϕ and θ vectors. ϕ can take the initial values of m^*, t_i and ϵ^*, while θ is initialized with values between -1 and 1. For instance, an increase in t_i translates into a decrease in m and ϵ of node $\mathcal{N}$, and thus the corresponding weights in θ are negative.

In our implementation, we chose an initial $\alpha = 0.01$ and then adjust α to be 0.2 after three iterations of the controller to prevent large oscillations during the first rounds of the estimators. Our network consists of 10 nodes, all in range of one another (the medium can be occupied by only one node at a time). We evaluate the new command $\bar{u}$ every $\mathcal{T} = \frac{5}{\text{packetRate}}$ seconds: for instance, if a node sends packets at a rate of 2 packets per second, the controller will run every 2.5 seconds. The feedback period $\mathcal{T}$ can be increased to reduce overhead, although a large value could cause the network adaptation to be sluggish—or worse, instable.

Variable initialization:
$\phi^m = \begin{bmatrix} m^* & 0 & 0 & t_i & 0 & 0 & m^* & 0 & 0 \end{bmatrix}^T$
$\phi^\epsilon = \begin{bmatrix} \epsilon^* & 0 & 0 & t_i & 0 & 0 & m^* & 0 & 0 \end{bmatrix}^T$
$\theta^m = \begin{bmatrix} 0.95 & 0.1 & 0.1 & -0.5 & -0.1 & -0.1 & 0.3 & 0.1 & 0.1 \end{bmatrix}$
$\theta^\epsilon = \begin{bmatrix} 0.95 & 0.1 & 0.1 & -0.5 & -0.1 & -0.1 & 0.3 & 0.1 & 0.1 \end{bmatrix}$
for ever **do**
  $m^* = f(\text{packetRate}, \mathcal{T})$
  $\epsilon^* = f(\text{radio}, m^*)$
  $\theta^m \mathrel{+}= \frac{\mu^m}{r^m}\phi^m(m - \phi^{mT}\theta^m)$
  $\theta^\epsilon \mathrel{+}= \frac{\mu^\epsilon}{r^\epsilon}\phi^\epsilon(\epsilon - \phi^{\epsilon T}\theta^\epsilon)$
  $u = \frac{\theta_p^m(m^{*+} - \sum_{i \neq p}^{u} \phi_i^m \theta_i^m) + \theta_p^\epsilon(\epsilon^{*+} - \sum_{i \neq p}^{v} \phi_i^\epsilon \theta_i^\epsilon)}{(\theta_p^m)^2 + (\theta_p^\epsilon)^2}$
  $\bar{u} = f_{0.1}^5[\bar{u} + \alpha(u - \bar{u})]$
  $\phi^m = \overrightarrow{\phi^m} + \begin{bmatrix} m & \bar{u} & m^* \end{bmatrix}$
  $\phi^\epsilon = \overrightarrow{\phi^\epsilon} + \begin{bmatrix} \epsilon & \bar{u} & m^* \end{bmatrix}$
  Where $\rightarrow$ is a matrix shift operator
  $r^m \mathrel{+}= \phi^{mT}\phi^m$
  $r^\epsilon \mathrel{+}= \phi^{\epsilon T}\phi^\epsilon$
end for

algorithm 3.1: DDCC pseudo-code for $p = 3$.

F. Observations for an Implementation of the One-Hop Case

For an implementation on real platforms in one-hop scenarios and for both controllers AADCC and DDCC, nodes need to periodically exchange information about their new t_i values. For DDCC, nodes additionally need to exchange information about their remaining energy, using broadcast packets for instance, in order to determine the node $\mathcal{N}$ whose energy should be spared. If nodes A and B are possible choices for node $\mathcal{N}$, node A can elect to minimize the consumed energy at both nodes. DDCC works equally well by estimating the energy consumption at both A and B, although the t_i value tends to be noisier.

If a node is the receiving end of multiple links, it should adopt the smallest t_i value t_{iL} calculated by its descendants in order to receive all packets successfully. The amounts of energy wasted on the links using a lower duty cycle are negligible because the sending nodes will stop their packet transmissions after half t_{iL} s on average—protocols like X-MAC and MX-MAC can interrupt their sending streams after receiving an ACK frame.

Finally, if a node has multiple unicast destinations, a rare case in WSNs, which tend to have only one data sink, node A calculates the appropriate t_i values for each link and sends them to the intended receivers individually. Support for multi-hop networks is introduced in Section V.

IV. Simulation Results For One-Hop Networks

First, we observe the case when two nodes compete for the medium to send packets and only one node can modify its duty cycle. Then, we validate the duty cycle control when more than one node concurrently adjusts their t_i values.

A. Method

The radio behavior was modeled not only after the CC2420 data sheet, but more importantly after the energy use of the whole Tmote Sky platform running a TinyOS implementation. Although we present simulation results, our model closely resembles a real-life deployment, typically within 3% of the measured energy consumption [9]. Here, the term simulation designates an accurate *reconstruction* of the reality. The choice of running simulations rather than an actual implementation was prompted by the difficulty in measuring energy consumption in real implementations: for LPL schemes, energy use does not degrade rapidly enough to collect usable and accurate data. An additional reason came with the objective of this work, which was to set the theoretical background for, and prove the feasibility of, t_i control. In our discussion of the results, we often refer to a "fixed duty cycle" case, which is the scenario when the t_i value is set at the beginning of the simulation and never changes (no duty cycle control). We ran such scenario simulations, but did not include their results on our graphs for clarity and space considerations.

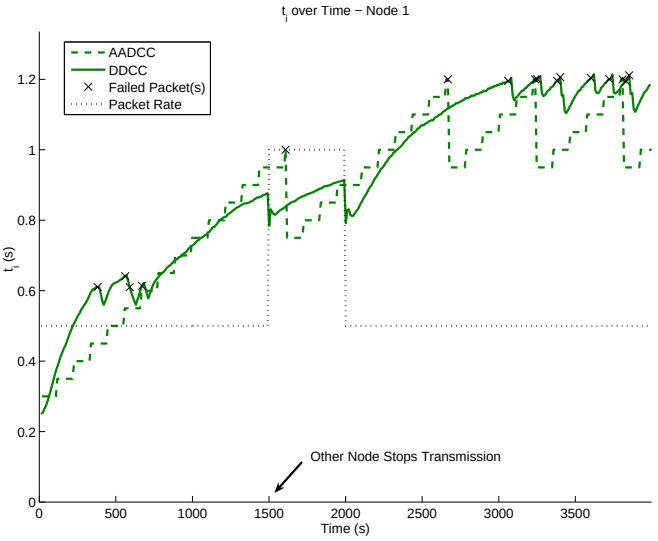


Fig. 4. Evolution of $t_i(t)$ as the packet rate varies.

B. Lowering the Duty Cycle to Save Energy: Demonstration of Principle

Without the ability to adapt t_i, nodes running a LPL MAC protocol would force designers to select a high duty cycle at deployment to ease contention in busy areas of the network. Consequently, we start with a t_i value of $300\ ms$, with two nodes sending packets at an initial rate of 0.5 packet per second.

Figure 4 presents the evolution of t_i as well as the scenario of the simulation. Because a lower duty cycle can comfortably accommodate concurrent packet rates of $0.5\ pkt.s^{-1}$, the value of t_i increases from $300\ ms$ to around $900\ ms$ in under $1,500\ s$ ($25\ min$) for both AADCC and DDCC controllers. At this point, the other packet source is turned off, and the packet rate of the remaining node is increased to $1\ pkt.s^{-1}$. The t_i value remains around $950\ ms$, as this t_i value translates into an energy consumption within close range of the target energy. After $2,000\ s$, the packet rate is halved to $0.5\ pkt.s^{-1}$. This allows the duty cycle to decrease further, as t_i goes

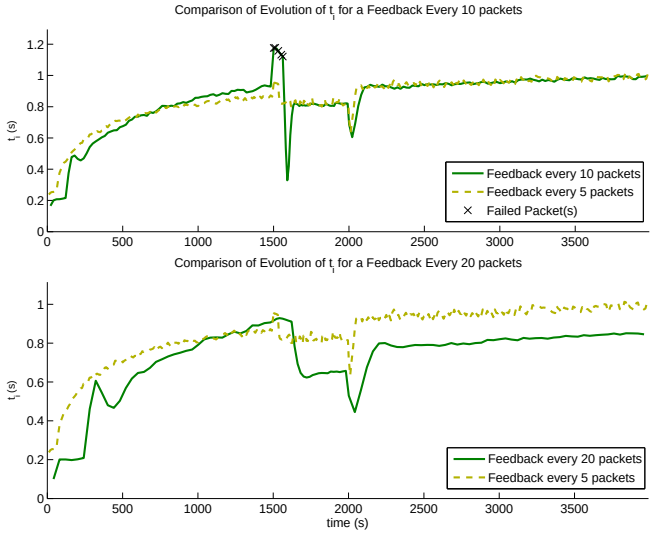


Fig. 5. Evolution of $t_i(t)$ for various values of $\mathcal{T}$.

from 950 ms to 1.2 s. Only 10 packets were lost during this scenario, in spite of the vigorous increase in t_i.

When compared to AADCC, DDCC helped reduce energy consumption by close to 3% (not shown) at the end of the simulation (close to a 19% reduction when compared to the fixed duty cycle case). As the later part of this scenario continues (after $2,000$ s), this number will increase.

C. Effect of the Feedback Period $\mathcal{T}$

We ran the same scenario as before for DDCC only, and changed the number of packets that are scheduled to be sent between evaluations of t_i to test their impact. We doubled and quadrupled the value of $\mathcal{T}$, or evaluated t_i every 10 and 20 packets. Figure 5 shows that there is little difference between periods of 5 and 10 packets. For 10 packets, the evolution of t_i appears to be smoother because of fewer updates. There is a greater difference between the t_i values calculated between transmissions of 5 and 20 packets: the error on the estimation of the target energy is multiplied fourfold, and causes the duty cycle to often be higher than needed.

D. Packet Loss Minimization

We now study the other variable of interest by observing the number of lost packets. This example differs from the previous one in the initial value of t_i (now 1.5 s) and in the number of neighbors transmitting over time. While it is unlikely that a protocol designer would choose such a high value for t_i in a "fixed" case (no available control at all), this part of our work shows the behavior of our control schemes when packet loss occurs.

Figure 6(a) shows a decreasing t_i as packets are dropped in both control cases. Repeated packet losses cause the duty cycle using DDCC to be increased by a greater amount. The t_i value can be observed to increase slightly between two feedback periods $\mathcal{T}$ in the first 25 min of the runtime as the energy component (approximately equal to the packet loss component) pushes the energy consumption down and

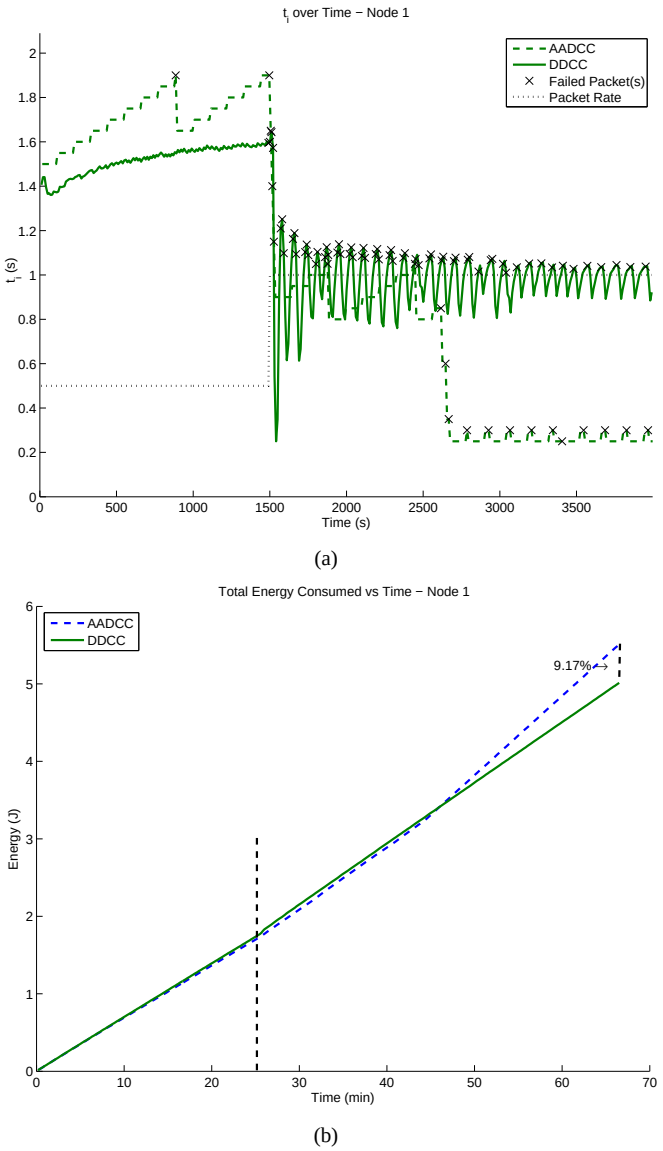


Fig. 6. Evolution of (a) t_i and (b) energy over time under a changing scenario: from a low duty cycle to a higher one.

the t_i value up. In the second part of the runtime, packet losses become more frequent as the t_i value is unrealistically high compared to the packet rate, until it reaches less than 1 s. Design choices could allow for a more aggressive t_i descent, which would prevent the "spikes" on t_i, but this would slightly compromise the rate of the t_i increase once the packet rate declines again. This illustrates once again the trade-off between energy consumption and packet loss that the dynamic scheme balances.

DDCC was able to limit packet loss by 88% over the "fixed" (no-control) scenario. As the duty cycle is iteratively modified, the frequency of dropped packets diminishes. Compared to AADCC, DDCC dropped 44 more packets (or 146% more) because it increased the duty cycle very aggressively. This is transposed on the energy side, shown in Figure 6(b), where DDCC reduced energy consumption by 9.2% compared to AADCC. Over the non-controlled case, the energy increased by 7% (not shown). The reason is that, in general, the

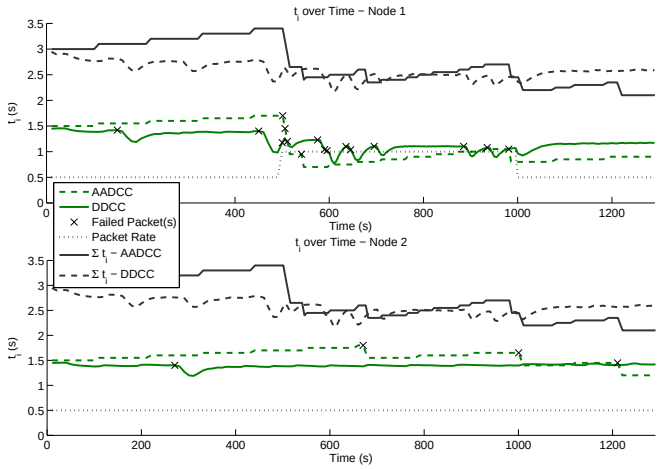


Fig. 7. Evolution of t_i over time under a changing scenario for the two nodes with duty cycle control.

increase in packet deliveries is compensated by an increase in energy consumption. The fixed scheme consumes less energy because of two reasons: its duty cycle remains at a low value, and contention around the nodes forces both the sender and the receiver to sleep for longer periods of time instead of transmitting packets—a behavior that results in lower energy consumption.

E. Multiple Controllers

A legitimate concern of t_i control deals with the implementation of several nodes adapting their duty cycles at the same time, particularly for a dynamic controller: the modifications of one should not destabilize the others.

Figure 7 shows that this is not the case as two sending nodes (nodes 1 and 2) correctly adapt their duty cycles to conditions in the local area. The "cumulative t_i" is the sum of the t_i values of the sending nodes and can be seen as a measure of the busyness of a local area.

V. t_i Control For Multi-Hop Networks

The previous section validates the principle behind t_i control for one-hop networks. In this section, we expand this work to single-branch multi-hop networks: only one data source sends packets to one data sink several hops away.

In this part of the work, the source (node 0) intends to send m^* packets to the destination (node n). Each packet travels along the same slowly changing path (*i.e.*, constant for a long period of time, corresponding to our simulation time for instance) over $h = n$ hops. Each node keeps a queue of a maximum of 100 packets.

A. Challenges Introduced By Multi-Hop Control

Although it is fairly inconsequential for the additive controller AADCC, the introduction of several hops along a source-destination path complicates key aspects of dynamic t_i control (DDCC): the delay between the beginning of a transmission at the source and its reception at the destination greatly increases. This delay is exacerbated by the nature of LPL MAC protocols because they rely on duty cycling. One consequence for dynamic t_i control is that instability increases, although it can be compensated by a smaller updating slope α of the command $u(t)$—we lower it to 0.1 or less.

Most importantly, the larger number of hops on the path induces non-linearities in the system. Before a packet can be transmitted, a node must wait for the packet's next-hop to wake up. At every link along the path, the packet is held for a varying amount of time (although on average equal to $t_i/_2$ s). Since the duty cycle is usually reevaluated every 5 to 10 packets, the packet delay (and its corollary, the number of transmitted packets m) show wide variations from one feedback to the next, with little correlation to the t_i value.

In addition to this problem, two approaches to control the duty cycle can be considered: a per-link strategy and a per-path strategy. The former strategy offered the appeal of simply replicating the work done in Section III for every link along the path, and we tried it first. Investigative work rapidly showed that this approach could not be successful because queueing would happen at one point in the path, deceiving other nodes into increasing their t_i because they correctly transmit m^* packets. In general, this solution offered many untractable problems such as keeping a set of two values of t_i at every node (one for the node itself, one for its next-hop so that the first one could send to the second), coordinating together to avoid queuing, etc.

Instead, the simple observation was made that since only one packet may be transmitted by a node k every t_i s, the nodes farther along the path ($> k$) would witness the same packet rate. Conversely, nodes placed before k would need to send at the same rate as k in order to maintain a constant queue at k. Therefore, we opted for a common duty cycle among all the nodes of a path, avoiding queuing whenever possible.

B. Node Synchronization Along a Path

While per-path t_i control eased many of the challenges we faced, non-linearities remained the main obstacle to multi-hop duty cycle control. We solved this problem through node synchronization along a path.

Certain LPL MAC protocols have the unique ability to synchronize without explicit notification (*i.e.*, without overhead) along a slowly-changing one-branch path. These protocols are X-MAC [7], C-MAC [11] and MX-MAC [9]; a node k following the schedule of either one of these protocols learns of its next-hop neighbor $k + 1$'s wake-up time at the end of every unicast transmission, that is, when it receives an acknowledgement frame. It follows that a node k can decide to back-off by a small t_S time so that it may wake-up right before node $k + 1$ during the next cycle. Done at every node along a path of h hops, nodes are automatically synchronized after the h^{th} packet has been successfully received. The details of node synchronization along a path are presented in [17].

Among other features, path synchronization allows urgent packets to be received and forwarded immediately (within the same t_i period) without loss of synchrony. Broadcast packets do not break node synchronization either.

More importantly, this technique reintroduces linearity in the system since nodes' wake-up times are separated by a

constant amount of time t_S s. Packet delays are equal to $t_{Rx}+t_S+(h-1)(t_i+t_S)$ for regular packets, and ht_S+t_{Rx} for urgent ones, where t_{Rx} is the time to receive a packet, approximately 14 ms on average for our packet size.

Furthermore, path synchronization significantly reduces congestion by staggering node wake-up schedules. Because nodes wake-up sequentially along the path, a packet transmission interferes with next-hop nodes only, and not previous-hop nodes. This greatly reduces the chance for collisions and back-off, which increases the accuracy of energy estimation at remote nodes.

C. Impact on the Energy Component of J

Because the wake-up schedules of nodes are staggered, the time to transmit a packet is predictable and almost constant (t_S+t_{Rx}), whatever the duration of t_i. In addition, since there is only one data source per path, each relay node must receive and send the same number of packets. Therefore, the expected energy consumption is the same at every relay node along the path since both the energies to send and receive a packet are equal at every hop. This reinforces the decision to utilize path-long duty-cycles.

In the case of only one data source on the network, saving the energy of one particular node on a multi-hop path no longer applies since all relays are expected to consume the same energy. Consequently, to lower the energy consumption of every relay node, the number of probes must be lowered such that a node may only wake-up to send or receive a packet. The data source or the data sink are notable exceptions, since the originator of the data does not receive packets. The data sink, which does not send packets, is generally a node with larger resources and is less likely to request its energy be spared. Unless specified otherwise, we discuss the more general results of the relay nodes, although similar techniques can be applied for the nodes at the extremities of the path, as is done in Section III.

Along synchronized paths, the energy consumption is thus the lowest when t_i is the highest but still allows the target number of packets m^* to be received. Hence, the controller arbitrates the trade-off between lower energy consumption and the objective to send m^* packets.

D. Observations for an Implementation of the Multi-hop Case

The control of the duty cycle using DDCC requires information now located more than one-hop away. In this section, we discuss possible practical solutions for implementing multi-hop duty cycle control.

1) Target Number of Packets: In all cases, we set packets to the same high priority. This meant that for path synchronization, they were all treated as urgent, and could thus be delivered within the same t_i period ($ht_S + t_{Rx}$ s later). In this section, we discuss where to close the feedback loop, *i.e.*, which node should be the t_i controller.

a) Calculation at Node 0*:* The target number of packets m^* now depends on the packet rate of the data source, located at the beginning of a multi-hop path. To calculate the new t_i value every feedback period $\mathcal{T}$, a node must know m^*, as well as the actual number of packets m received by the destination n. Because all nodes are sharing the same t_i value and because they are synchronized along the path, the number of packets sent by node 0 is equal to m, provided none of the packets are dropped for unforseen reasons (a bad radio state, localized noise spike, etc.).

b) Calculation at Node $n-1$*:* The previous technique does not guarantee proper delivery of m^* packets at the destination if some of the links along the path are faulty. Because node $n-1$ receives an ACK frame every time the destination receives a packet, it can easily calculate m. For this reason, the next-to-last node can be chosen to perform t_i control.

This method may be preferred by programmers who suspect that nodes may fail and that detection of such failures will be slow. However, there is an inherent trade-off between packet overhead to spread the new t_i value and delivery reliability.

2) New t_i *Value Dissemination:* After $t_i(k+1)$ has been calculated, the duty cycle controlling node should communicate this new t_i value to nodes on the transmission path using $t_i(k)$ and by piggy-backing the new value onto broadcast packets. For the family of LPL MAC protocols, bigger packets incur no extra energy consumption since the radio remains in *sending* mode for the same period of time (t_i s) regardless of the packet length. Nodes can then start using the new calculated duty cycle. The cost of this operation is at most that of transmitting one broadcast packet every $\mathcal{T}$ s on the active data path. In order to be the most energy efficient, this dissemination on the path should coincide with other network maintenance events. In WSNs, the directions of data packets is usually fixed over a small period of time (*e.g.*, $\sim 10\mathcal{T}$) and centripetal: packets tend to travel from peripheral nodes (with valuable information to report) toward the base station (with compute power). Conversely, broadcast packets, generally used for network maintenance such as route repair or service discovery, tend to flow in the opposite direction. Cross-layer optimizations could join other maintenance packets with our t_i updates. It should be noted that the problem of t_i dissemination is not unique to our proposed schemes, but common to the whole family of adaptive duty cycle protocols. In fact, this work benefits from path synchronization, a very energy-efficient technique for multi-hop unicast packet transmissions [18].

3) Energy Estimation: The node running dynamic duty cycle control needs to estimate the energy consumed by other nodes *without* requiring them to report it. The reasons to proceed in this way are threefold:

- Good modeling: we were able to closely measure and model LPL MAC protocol energy consumption. Our model was found to be typically within 3% of test-bed measurements.
- Energy evaluation in isolation: nodes would *also* have to evaluate the energy consumed only by their own LPL MAC protocol because they may be running other processes (packet processing, sensing activity, packet aggregation) that draw energy but are not relevant to the MAC links.
- Poor measuring tools: platforms like the Tmote Sky can

only measure their battery voltage, which can be mapped to remaining energy but does not yield a sufficient precision when energy consumption is small (as is typically the case in LPL MAC protocols).

Errors made in the evaluation of the energy consumption are modeled in the noise component of the system.

E. Simulation Results

We used Matlab to simulate these different strategies for a four-hop network with only one source. The control of the duty cycle was strikingly similar, whether t_i was calculated at node 0 or node $n-1$. However, our simulation did not model unforseen congestion at nodes > 1 (caused by other transmissions in the vicinity of a node for instance), thus allowing the controller at node 0 to perform equally well.

In this section, we present results obtained when node $n-1$ is the controller, as in Section VI.

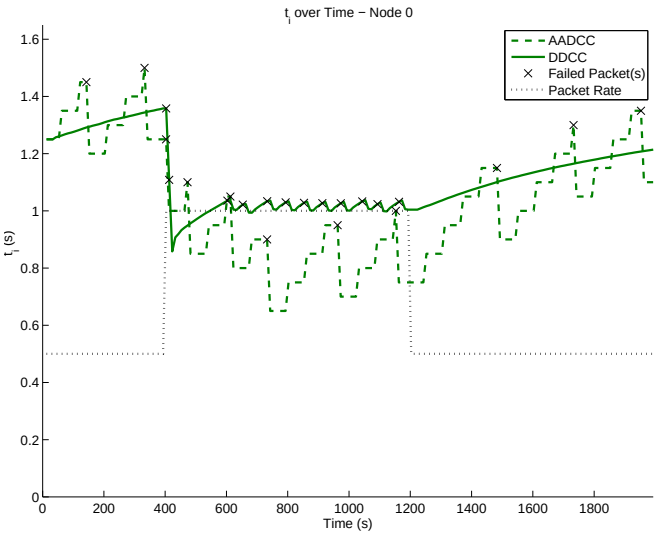
We first present results that were obtained through the direct evaluation of the consumed energy. Figure 8(a) shows the evolution of the duty cycle of the nodes when the packet rate of the source changes over time. When the packet rate of source node 0 doubles after $400\ s$, it fails to send m^* packets per $\mathcal{T}$ period. Both controllers are successful in bringing the duty cycle to a value that allows the target number of packets to be reached ($0.9\ s$ for AADCC and $1\ s$ for DDCC). Because DDCC has an energy component in its command computation, it is sometimes too eager to increase t_i; we see an example of this here. When the packet rate returns to $0.5\ pkt.s^{-1}$, the duty cycle decreases again, providing a t_i around $1.2\ s$. Because we opted for a small update rate $\alpha = 0.1$, the dynamically controlled t_i value increases slowly.

Figure 8(b) shows that t_i control reduces the number of packets failing to be delivered. Compared to the case without a controller, this reduction reaches a factor of four (not shown). The packets transmitted by node 0 to node 1 are delivered to the destination within the same cycle. For the fixed duty cycle case however, packets must be queued between $400\ s$ and $1,250\ s$. While queued packets can be eventually sent to the destination after the packet rate decreases, stale information is of little use to the application. Compared to AADCC, the number of dropped packets is similar for DDCC (13 vs. 17, or 40%) with no queuing happening in either case.

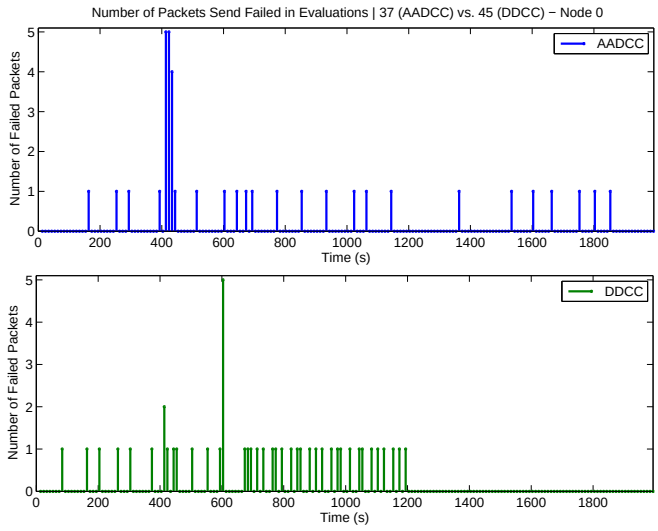
The energy consumption of DDCC is comparable, although lower than that of AADCC by 2% (not shown). Compared to the non-controlled case, the improvement in packet delivery is obtained by a relative increase in energy consumption of 10% at relay nodes after $2,000\ s$, although the eventual energy consumption at the relay node can be eased when the duty cycle returns to a low value.

The second set of results is presented for the case when the energy consumption is not evaluated at the dynamic controller, and the command $u(t)$ is automatically increased by $0.1\ s$ when $5\ \mathcal{T}$ have passed without packet loss (like the linear increase of AADCC).

Figure 9(a) shows the correct reduction of t_i to accommodate sending more packets with either controller. When the network is favorable to a t_i increase, the response of DDCC



(a)



(b)

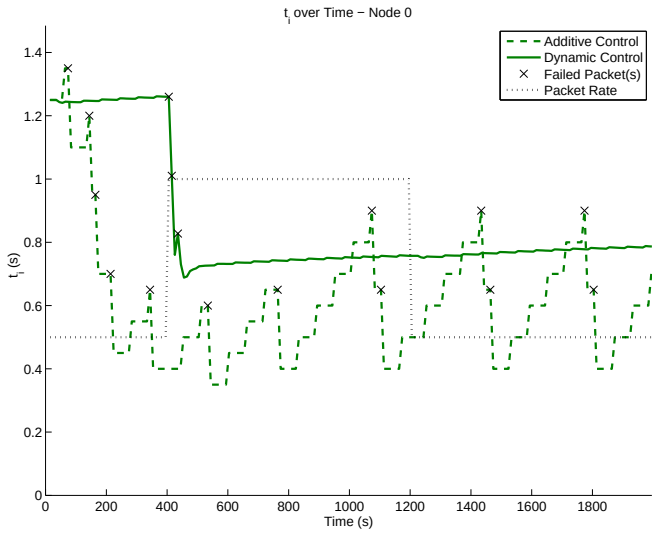
Fig. 8. Comparison of (a) the evolution of t_i and (b) the dropped packets for the dynamic controller and additive controller when the energy is evaluated.

is much more sluggish than in the regular and the additive controller cases (in fact, the dynamic controller opposes artificial t_i increases). However, this translates in fewer dropped packets (reduced by a factor of three in Figure 9(b), or a factor of nine compared to the fixed duty cycle case), and in a lower energy consumption of 3% (higher by 8% over the fixed duty cycle case).

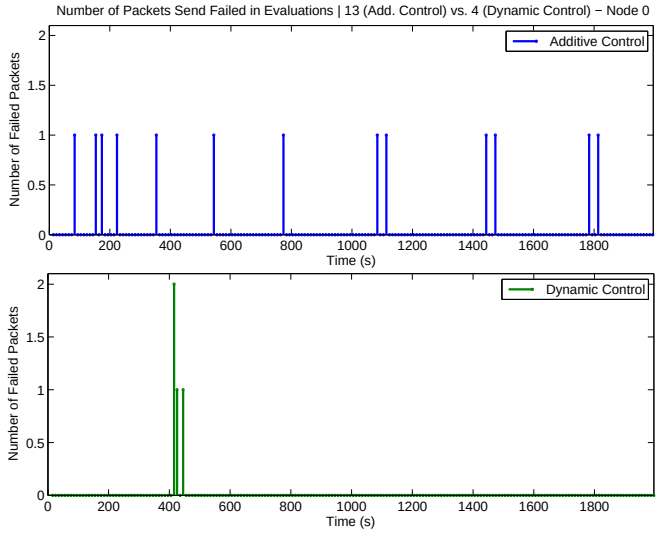
These results illustrate the trade-off existing between the two techniques for utilizing energy information in DDCC, described in Section III-D2: the speed of the response translates into different energy consumption and packet delivery ratios. The decision to implement one technique or the other depends on the application needs and constraints.

VI. t_i Control for Multi-Hop Networks With Multiple Sources

While adaptation of our controllers to multi-hop networks has greatly expanded the applications of t_i control, the



(a)



(b)

Fig. 9. Comparison of (a) the evolution of t_i and (b) the dropped packets for the dynamic controller and additive controller cases when the energy is not evaluated as per Section III-D2.

limitation imposed by only one data source limits its use to networks performing source selection—target tracking or building monitoring networks are instances of these networks.

In this section, we modify both t_i control strategies to support multi-hop networks with multiple data sources: while the control engine remains the same as in the previous section, our synchronization technique was upgraded to support multiple sources converging into one branch.

A. Path Synchronization With Multiple Sources

The greatest challenge posed by the use of multiple sources does not directly fall onto the theory behind t_i control, but rather concerns how path synchronization can be maintained when several sources are converging at one node.

Let Figure 10 represent a three-hop network with two sources (marked by $*$) sending packets to a common destination $h = 3$. First, we define several terms used throughout

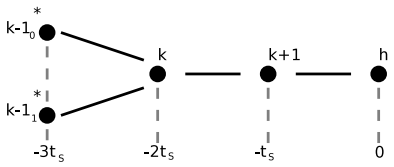


Fig. 10. A multi-hop network with two sources $k-1_0$ and $k-1_1$ ($*$ denotes a source).

in this section: a *branch node* refers to node k, as the location where several flows meet. Nodes from a packet source to the branch node form a *branch*. The nodes placed after the branch node are part of the *root* of the path.

The current synchronization technique staggers node transmission schedules such that node $k+1$ would wake-up $t_S\ s$ before the destination h, node k $2t_S\ s$ before h, etc. This forces both sources (more generally, all nodes $k-1$ forming a link $k-1 \rightarrow k$) to wake-up at almost the same time. While this may be acceptable for low offered loads, it cannot accommodate high packet rates from both sources.

1) Strategy: The key idea to support the convergence of l flows to one node k consists in increasing the duty cycle of root nodes ($\geq k$). The t_i value must be divided by l to accommodate fair access to node k by all sources $k-1_j$[2].

Upon receiving a new unicast packet, node k checks the ID of the previous-hop and adds it to a neighbor table if not present already. If a new source is detected, node k modifies the received packet to include a MAC header containing the value l, the ID $k+1$ of its next-hop, and the ID of the new previous-hop. The new t_i value is calculated as:

$$t_i^{new} = t_{i,k}^{new} = t_{i,k}\frac{l}{l-1}$$

The packet is then broadcast to all immediate neighbors. Nodes k and $k+1$ adopt the new duty cycle after forwarding the packet to their next-hop neighbor.

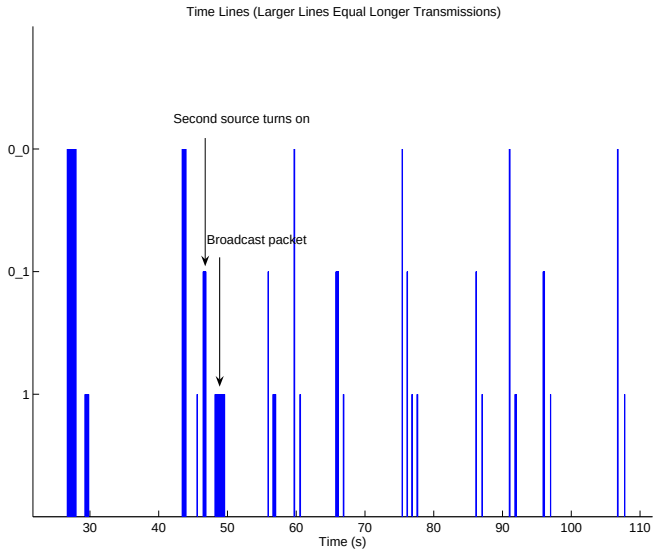
The broadcast packet sent by node k need not specify who the new source node is: upon attempting to send a packet and finding a busy network, a node $k-1_j$ will back-off by $t_i^{new}/_l\ s$ until all schedules are staggered.

2) Implementation in TinyOS: We tested and implemented this synchronization technique in TinyOS [19] using the Tmote Sky platform [20]. In order to gather results, we let Matlab collect information through a TinyOS gateway. One of the difficulties in showing path synchronization is the fact that probes are "silent": nodes simply turn their radios in *receive* mode, and receive a packet or go back to sleep. Only packet transmissions can be reported and plotted. We show path synchronization through the timing of packet transmissions.

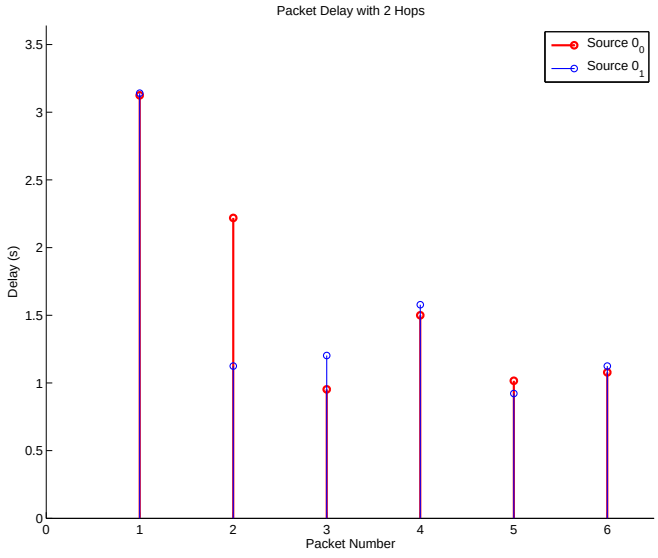
Our test network consisted of four nodes, with two sources (0_0 and 0_1) sending to node 1. The initial t_i value was $1.5\ s$. The TinyOS code was successfully tested for more sources, but since they are harder to read, these results are not shown here.

[2]The notation $k-1_j$, where $0 \leq j < l$, designates the previous hop of node k on branch j.

Figure 11 illustrates the process of path synchronization with two sources sending data packets to a common destination every $20\ s$ and $10\ s$. The Y-axis indicates the ID of the transmitting node. During the first half-minute, the path is established through a simple route discovery protocol and only one source is turned on. It takes two packets to synchronize the first source, which can be observed by the narrowing of the transmission bars. After $45\ s$, the second source turns on and sends its packet. It is immediately followed by a broadcast packet (sent over the full duration of the t_i interval). Immediately after, we can see that the schedules are staggered. After $75\ s$, node 0_0 sends a packet to node 1, immediately followed ($^{t_i}/_2\ s$ later) by node 0_1. Node 1 then forwards both packets successively.



(a)



(b)

Fig. 11. (a) Successful path synchronization for two nodes sending data packets to a common destination. The packet rates are $^1/_{20}$ and $^1/_{10}\ pkt.s^{-1}$. (b) The reduction in packet delay as the synchronization takes place.

Figure 11(b) shows the packet delivery delay for both sources. Once the nodes are synchronized, the packet delay hovers around $1\ s$—except for each sources' fourth packet, because 0_1 sends its data before 1 can forward the packet from 0_0. The first packets for both sources experience almost the same long delay (greater than $3\ s$), although for different reasons: when source 0_0 sends its first packet, nodes are not yet synchronized, and packet delivery is delayed by long transmission times. We see this delay being reduced in the following packet because node 1 synchronizes with its next-hop neighbor. The first packet of the second source, however, is delayed by the transmission of the broadcast packet indicating a new t_i value. Since synchronization is already in place on the existing path, source 0_1 is synchronized with node 1 with the first packet, which explains why the following packet (with ID 2) from node 0_1 experiences low delay.

B. t_i Control For Synchronized Paths With Multiple Sources

Path synchronization with multiple sources causes the root of the path to use different t_i values. This technique prevents DDCC from excessively increasing the duty cycle to support both sources with the same t_i. Compared to that approach, path synchronization results in energy savings for the branches of the network since their t_i value can be l times that of the root.

Since the node controlling the duty cycle must learn about the target number of packets m^*, it should be placed after the branch node. Thus, node $n-1$ (the node immediately before the destination) is a good candidate to be the controller for a path with multiple branches. Upon starting and stopping its flow of packets, a source must notify the controller of the number of packets it needs to transmit every second. This value is piggy-backed onto the unicast data packet and is read by the controller. After the new t_i value has been computed, it is broadcast and flooded onto the path.

While the synchronization technique has evolved to support multiple branches, the t_i control engine has remained the same as that of Section V. This shows the robustness of the both control techniques designed for multi-hop cases.

C. Simulation Results

The tested network consists of two sources, as shown in Figure 12 sending packets over a four-hop path with initial t_i of $1.25\ s$. The controller is the node placed before the destination, and we compare compared both the AADCC and DDCC controllers. Both schemes benefit from path synchronization to guarantee fairness in the comparison.

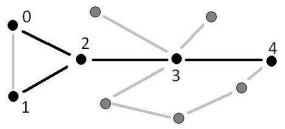


Fig. 12. Topology of the tested network with two sources.

Figure 13(a) shows the evolution of the duty cycle for AADCC and DDCC. During the initial phase of the simulation, the network experiences difficulties delivering all its packets, and thus decreases t_i. After $400\ s$, the second source is turned on, and the total packet rate is tripled. The branch

synchronization process activates, which can be visualized by a division of t_i by two. The duty cycle of the network with AADCC is still doubled because of the synchronization process taking place after a second source has been detected. The response from the DDCC is a controlled increase in the duty cycle to accommodate the new load. During this time, DDCC is a little too eager to increase the t_i value, and occasionally exceeds a safe value (presumably around $500\ ms$) that allows delivery of m^* packets (the same is true of AADCC). At $1,250\ s$, the second source is turned back off, and the duty cycle is reduced to save energy.

Using DDCC, the network dropped the same number of packets as when using AADCC (it dropped fewer packets by a factor of six compared to the fixed duty cycle network): Figure 13(b) shows that packets are mostly lost in the moments after the second source is turned on. Immediately following its activation (after $400\ s$), a loss of five, then three packets pushes the t_i value lower.

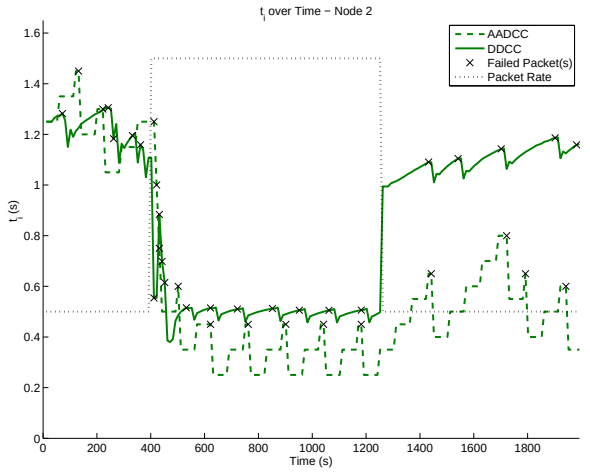
Figure 13(c) shows the extra energy consumed at node 2 (the node where the branches converge) when the duty cycle is increased. DDCC was able to balance dropped packets and energy consumption better than AADCC, as the energy consumption of the former is 8.5% lower (it is 3% higher compared to the fixed duty cycle case). There is a clear trade-off in using any sort of adaptive duty cycle controller between improvement in quality of service (through the increase in immediate delivery of packets) and energy consumption. However, it can be argued that although the energy expanded in the case of the fixed duty cycle scheme is lower (because of a lower duty cycle and because contention forces nodes to sleep longer), it is done in vain since many of the packets fail to be delivered. When using a duty cycle controller, the proposed DDCC strikes a better balance between packet delivery and energy consumption than AADCC for this tested scenario.

VII. Conclusions, Discussion and Future Work

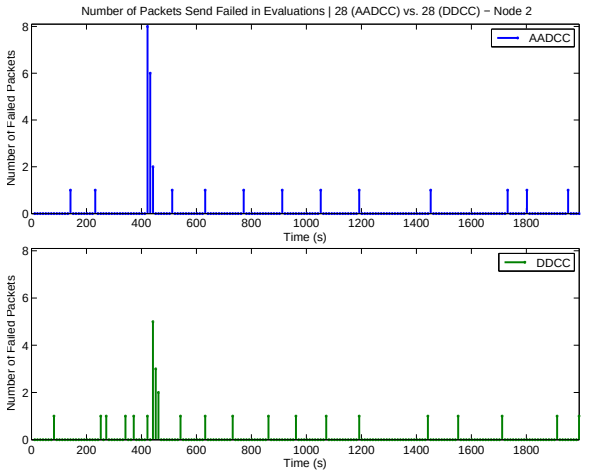
A. Summary of Work

Low-Power-Listening MAC protocols show great promise to increase WSN lifetime by reducing idle listening. However, such MAC protocols were typically reserved for networks with low packet rates so as to allow low duty cycles (and greater energy savings).

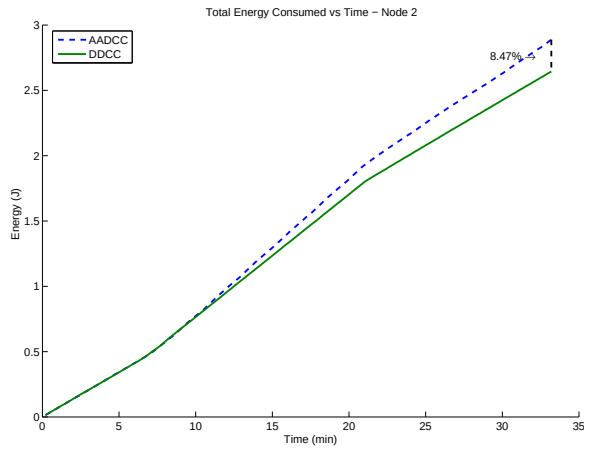
In this paper, we introduce two adaptive duty cycle control schemes. The Asymmetric Additive Duty Cycle (AADCC) Control and the Dynamic Duty Cycle Control (DDCC) have differing performance, but both provide adaptive control to the duty cycle of LPL MAC protocols. Both schemes are capable of increasing the duty cycle when the number of packets to be transmitted cannot be accommodated, and they can both decrease the duty cycle to conserve energy. DDCC jointly optimizes the energy consumed at vulnerable nodes and the number of packets to be transmitted. This results in energy savings of 2% to 10% compared to AADCC, and 20% compared to the fixed duty cycle case for single-hop networks. When they cannot lower energy (because they have to accommodate the transmission of more packets), both AADCC and DDCC succeed in drastically reducing the number of dropped packets compared to the fixed duty cycle case.



(a)



(b)



(c)

Fig. 13. Comparison of (a) the evolution of t_i, (b) the dropped packets and (c) the energy consumed for the dynamic controller and additive controller cases for two sources in a multi-hop network.

We generalized these results to multi-hop networks with multiple sources. The key to successful t_i control was using and improving a path synchronization technique that allowed linearity to be maintained in the system. In our experiment, we saw a reduction of dropped packets by a factor of six

(compared to the fixed duty cycle case) when the offered load increases. The higher duty cycle caused only a limited increase in energy consumption.

This work showed that t_i control allows networks to respond to sudden bursts of packets as caused by the occurrence of an event in a monitoring network, making LPL MAC protocols suitable for a greater number of WSN applications. t_i control allows the network to choose a very low duty cycle, thus saving considerable amounts of energy when the network load is low, while accommodating higher loads whenever needed.

More importantly, the proposed DDCC method, which does not require knowledge of a system's physical model, can also be applied to the control of many other parameters in a network.

B. Discussion

Although the balance of sent packets and energy consumption is closer to optimal when using DDCC (minimization of the error function), protocol designers may consider various aspects when deciding whether to implement additive control or dynamic control. While DDCC generally provides t_i values closer to the optimal values, its implementation requires selecting an update coefficient α, as well as determining if energy conservation is more important than packet delivery. These aspects guide the responsiveness of DDCC to packet loss or when the packet rate decreases. In the results presented in this work, DDCC proved to correctly handle packet losses and to respond swiftly by increasing the duty cycle (Figures 6, 7, 8 and 13). When it comes to increasing t_i when the packet rate decreases, DDCC is relatively sluggish. This is due to a small α to prevent oscillations. On the other hand, AADCC brings t_i down very quickly when a packet loss occurs, and it is in incapable of adapting the amount of t_i decrease or increase as it nears its target (unlike DDCC). However, AADCC is a simpler scheme that provides satisfactory results. The implementation of AADCC promises to be quicker, and to utilize much smaller computational resources. All these considerations may make one controller more attractive over the other.

C. Future Work

For our future work, we plan to adapt DDCC to real-life deployments. Other estimators may be considered for our work: we could replace the NLMS algorithm by the Newton's method of gradient descent. We plan to adapt our method of path synchronization and t_i control to more particular networks such as those made of branches of branches. Finally, we plan to investigate the possible cross-layer interactions made possible by this work. In particular, we would like to explore the impact of duty cycle on route selection: the routing protocol may find alternate routes to nodes with very high duty cycles and little remaining energy.

REFERENCES

[1] A. El-Hoiydi, "Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks," in *Proceedings of the IEEE International Conference on Communications (ICC'02)*, Apr. 2002.

[2] A. El-Hoiydi and J. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," in *Proceedings Computers and Communications (ISCC'04)*, Jun. 2004.

[3] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2^{nd} ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Nov. 2004, pp. 95–107.

[4] K. Langendoen, "Medium access control in wireless sensor networks," in *Medium Access Control in Wireless Networks, Volume II: Practice and Standards*. Nova Science Publishers, 2007.

[5] IEEE Computer Society LAN MAN Standards Committee, "Wireless medium access control (MAC) and physical layer (PHY) specifications for low rate wireless personal area networks (LR-WPANs)," in *IEEE Std. 802.15*, 2004.

[6] Chipcon Products from Texas Instruments, "CC2420 data sheet, 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver." [Online]. Available: http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf

[7] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4^{th} Embedded Networked Sensor Systems (SenSys'06)*, Nov. 2006, pp. 307–320.

[8] K.-J. Wong and D. Arvind, "Speckmac: Low-power decentralised mac protocol low data rate transmissions in specknets," in *Proceedings 2^{nd} IEEE International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality (REALMAN'06)*, May 2006.

[9] C. J. Merlin and W. B. Heinzelman, "Network-aware adaptation of mac scheduling for wireless sensor networks," in *Proceedings 3^{rd} Conference on Distributed Computing in Sensor Systems (DCOSS'07 Poster Session)*, Jun. 2007.

[10] R. Jurdak, P. Baldi, and C. V. Lopes, "Adaptive low power listening for wireless sensor networks," in *IEEE Transactions on Mobile Computing*, vol. 6, no. 8, Aug. 2007.

[11] S. Liu, K.-W. Fan, and P. Sinha, "CMAC: An energy efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks," in *Proc. Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'07)*, Jun. 2007.

[12] P. Lin, C. Qiao, and X. Wang, "Medium access control with a dynamic duty cycle for sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'04)*, vol. 3, Mar. 2004, pp. 1534–1539.

[13] C. M. Vigorito, D. Ganesan, and A. G. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *Proceedings of The Fourth IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'07)*, Jun. 2007.

[14] H. K. Le, D. Henriksson, and T. Abdelzaher, "A control theory approach to throughput optimization in multi-channel collection sensor networks," in *Proceedings of the Sixth International Symposium on Information Processing in Sensor Networks (IPSN'07)*, Apr. 2007.

[15] G. C. Goodwin and K. S. Sin, in *Adaptive Filtering Prediction and Control*. Prentice-Hall, 1984.

[16] P. Kumar and P. Varaiya, in *Stochastic Systems*. Prentice-Hall, 1986.

[17] C. J. Merlin and W. B. Heinzelman, "Node synchronization for minimizing delay and energy consumption in low-power-listening mac protocols," in *Proceedings 5^{th} IEEE Conference on Mobile Ad-hoc and Sensor Systems (MASS'08)*, Sep. 2008.

[18] ——, "Node synchronization for minimizing delay and energy consumption in low-power-listening MAC protocols," in *IEEE Transactions on Mobile Computing (to appear) and available at http://www.ece.rochester.edu/∼merlin/NodeSynchronization/NodeSync URTR.pdf*, 2010.

[19] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for network sensors," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[20] MoteIV Tmote sky, http://www.moteiv.com/tmote.