1-2017

Enabling sustainable bulk transfer in environmentally-powered wireless sensor networks

Alvin Cerdena VALERA
Singapore Management University, alvinvalera@smu.edu.sg

Wee-Seng SOH

Hwee-Pink TAN
Singapore Management University, hptan@smu.edu.sg

DOI: https://doi.org/10.1016/j.adhoc.2016.10.008

Published in Ad Hoc Networks, January 2017, Volume 54, Pages 85–98.
http://doi.org/10.1016/j.adhoc.2016.10.008

Enabling Sustainable Bulk Transfer in Environmentally-Powered Wireless Sensor Networks

Alvin C. Valera[a,b], Wee-Seng Soh[b], Hwee-Pink Tan[a]

[a] *School of Information Systems, Singapore Management University, Singapore 178902*
[b] *Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117583*

Abstract

We address the problem of transferring bulk data in environmentally-powered wireless sensor networks where duty cycle compliance is critical for their uninterrupted operation. We propose Pump-and-Nap, a packet train forwarding technique that maximizes throughput while simultaneously enforcing compliance to dynamic duty cycle limitations. A node using Pump-and-Nap operates by *pumping* a train of packets followed by a *napping* period where the node forgoes any transmission. Pump-and-Nap employs an *adaptive controller* to periodically compute the *optimal capacity*, that is, the maximum number of packets a node can receive and transmit in a train, given its duty cycle constraint. The controller uses prior input-output observations (capacity allocations and their corresponding duty cycle usage) to continuously tune its performance and adapt to wireless link quality variations. Its use of local information makes the controller easily deployable in a distributed fashion. We implemented Pump-and-Nap in TinyOS and evaluated its performance through experiments and testbed simulations. Results show that Pump-and-Nap provides high transfer throughput while it simultaneously tracks the target duty cycle. More importantly, Pump-and-Nap enables sustainable bulk transfer compared to state-of-the-art techniques that greedily maximize throughput at the expense of downtime due to energy depletion.

Keywords: Bulk transfer; energy-harvesting; adaptive control; dynamic duty cycling; sensor network

Email addresses: alvinvalera@smu.edu.sg (Alvin C. Valera), elesohws@nus.edu.sg (Wee-Seng Soh), hptan@smu.edu.sg (Hwee-Pink Tan)

1. Introduction

Wireless sensor networks are becoming ubiquitous because of their diverse applications in areas such as agriculture, environmental monitoring, industrial and home automation, military, and structural health monitoring, to name a few [1]. A critical issue that plagues many deployments, however, is the *limited lifetime problem* due to the finite battery capacity of sensor nodes [2]. Fortunately, advances in energy harvesting and storage technologies are enabling the deployment of *environmentally-powered* wireless sensor networks (EPWSN), wherein the sensor nodes harvest energy from the environment to recharge their batteries or energy stores. Recently, the use of supercapacitors as primary energy store is becoming popular because of their significantly higher number of recharge cycles compared to batteries [3]. Some example nodes that solely rely on supercapacitors are Everlast, Solar-Biscuit, and Sunflower [2].

In many applications (*e.g.*, [4, 5]), sensor nodes are tasked to record time-series data at high sampling rates, resulting in large or bulk sensor data. These bulk data, typically in the order of tens to hundreds of kilobytes, need to be transferred in real-time to a gateway for eventual transmission to the backend, where further processing and analysis can be undertaken. Due to storage limitations in sensor nodes, bulk data must be immediately transferred to avoid overflow and data loss. Bulk transfer in EPWSNs is challenging because the nodes perform adaptive duty cycling to ensure uninterrupted operation [6, 7, 8]. This is especially imperative in deployments wherein the sensor nodes rely solely on low capacity energy stores such as supercapacitors [2]. Such nodes must strictly operate according to a specified duty cycle, or risk downtime due to short-term energy shortage.

In this work, we tackle the problem of bulk data transfer in EPWSNs where adherence to duty cycle constraints is a primary concern. While several bulk transfer schemes have been proposed [9, 10, 11, 12, 13, 14], they focus mainly on maximizing the throughput, neglecting the duty cycle constraints of sensor nodes. The use of existing schemes may therefore cause uncontrolled and rapid draining of the energy reserves, leading to the temporary unavailability of nodes along the transfer path. Ultimately, this will result in transfer disruptions which render the transfer of arbitrarily-sized sensor data difficult, if not infeasible.

Recently, the use of *packet bursting* or *packet trains* in conjunction with radio duty cycling [12] have been proposed to attain low power, high transfer

throughput. While the technique yields low energy consumption, the outcome is *incidental* rather than *intentional*, *i.e.*, the use of packet trains does not actively control the energy usage to be within specified bounds. We therefore introduce PUMP-AND-NAP, a forwarding technique that uses *controlled packet trains* to simultaneously maximize throughput and enforce compliance to dynamic duty cycle limitations. At the heart of PUMP-AND-NAP is an *adaptive controller* that determines a node's *optimal capacity*, defined as the maximum number of packets the node can receive and transmit in a train within its duty cycle constraints. The controller uses prior input-output observations (capacity allocations and their corresponding duty cycle usage) to continuously tune its performance and adapt to wireless link quality variations.

We implement PUMP-AND-NAP in TinyOS [15] and perform experiments in the Indriya testbed [16], a 139-node indoor testbed, to evaluate its performance. Experimental results show that PUMP-AND-NAP can adaptively track duty cycles and provide high bulk transfer throughput at the same time. More importantly, we demonstrate in energy harvesting experiments and testbed simulations that PUMP-AND-NAP can truly enable sustainable bulk transfer compared to state-of-the-art techniques [9, 12] that greedily maximize throughput at the expense of downtime due to energy depletion.

The rest of the paper is organized as follows. In Section 2, we review the state-of-the-art duty cycling and bulk transfer in sensor networks and identify the challenges in the context of EPWSN. In Section 3, we describe PUMP-AND-NAP in detail while in Section 4, we present its implementation, along with the experiments designed to evaluate and compare its performance. In Section 5, we present and discuss the experimental results. We conclude the paper in Section 6 and state several possible future work.

2. State-of-the-art and Challenges

To understand how bulk transfer protocols will perform in EPWSN, we survey the state-of-the-art in duty cycling and bulk transfer. The ultimate aim of this section is to expose the shortcomings of existing bulk transfer schemes when duty cycle compliance is of paramount importance.

2.1. Duty Cycling MAC Protocols

A duty-cycling node may employ any of the state-of-the-art duty cycling medium access control (MAC) protocols to control the sleep and wakeup of

its radio. Duty cycling MAC protocols can be either *synchronous* [17, 18, 19] or *asynchronous* [20, 21, 22, 23]. In the former, the nodes sleep and wakeup at the same time while in the latter, the nodes may sleep and wakeup at different times. In this work, we motivate our design using asynchronous schemes because they offer two distinct advantages over synchronous schemes: (*i*) they do not require periodic re-synchronization which can entail significant energy consumption [24]; and (*ii*) they do not require the storage and exchange of wakeup schedules which can entail significant memory and communication overhead [25]. Nevertheless, our resulting scheme can also be used on top of synchronous MAC protocols after slight modifications.

In asynchronous schemes, a packet transmission is preceded either by a *beacon listening phase* or a *preamble(s) transmission phase*[1]. The former is employed in *receiver-initiated* schemes (*e.g.*, [22]) while the latter is used in *transmitter-initiated* schemes (*e.g.*, [20, 21, 23]). Regardless, the transmitting node always incurs this overhead before it can have the opportunity to transmit its packets. For simplicity, we introduce a common term to refer to either overhead:

Definition 1 (Pre-transmission Overhead). *The duration from the moment a transmitting node v has a packet ready for transmission until the time the receiving node w wakes up. During this time, v's radio is active, either awaiting for a beacon (receiver-initiated) or transmitting preamble(s) (transmitter-initiated).*

2.2. Bulk Transfer

Bulk transfer refers to the transmission of large amount of sensor data from a source node to a destination node, typically a gateway or a base station. Bulk transfer can actually be performed using *generic* transport protocols (Wang *et al.* [26] provides a good survey on this subject) but specific application requirements and tight resource constraints in terms of memory, channel capacity and energy have led to the development of *specialized* protocols for bulk transfers.

Koala [27] is one of the earliest schemes for bulk transfer. It uses round-trip time (RTT) to control the sending rate from the source to the sink. Specifically, Koala sends packets at a rate of RTT/2, relying on its underlying flexible control protocol to provide reliability. Koala supports duty

[1] In [23], preambles are replaced by actual data packets.

cycling and uses low-power probing, a technique akin to beacon transmission in receiver-initiated MAC protocols.

Unfortunately, RTT-based rate control performs poorly over long paths. As such, newer schemes such as Flush [9] and PIP [10] introduced the idea of "pipelining" packets to improve throughput. Flush [9] proposed a method to probe the interference range of a path and uses two simple rules to maximize the sending rate of a node: (*i*) transmit when the successor node is free from interference, and (*ii*) transmit at rate below the successor node's sending rate. PIP [10] took the idea of packet pipelining further through the use of a MAC protocol that is TDMA-based, centralized, connection-oriented and uses multiple channels. PIP essentially aims to tightly coordinate the packet pipelining from the source to the sink and further reduce intra-flow and inter-flow interference. The former occurs when transmissions of different nodes from the same flow interfere with each other, while the latter occurs in the case of transmissions from different nodes belonging to different concurrent flows.

Flush and PIP are designed to maximize throughput without regard to the energy consumption of the sensor nodes. To achieve the desired packet pipelining effect, they need the radio to be turned on for the entire transfer duration. To remedy this problem, Duquennoy *et al.* [12] proposed the use of *packet bursting*, *i.e.*, rapid transmission of successive packets after a single wakeup, in conjunction with duty cycling. Results show that packet bursting in conjunction with duty cycling in ContikiMAC [23] can provide low power and high throughput performance. We note however that although [12] can provide low power consumption, it is *incidental* rather than *intentional*, *i.e.*, it does not actively control consumption to be within specified bounds.

2.3. Bulk Transfer in EPWSN

We can group the bulk transfer schemes that we have presented previously into two categories, namely, *single packet-based* and *packet train-based*. Koala, Flush and PIP fall under the first category while the scheme by Duquennoy *et al.* falls under the latter. In what follows, we identify the issues of using either scheme in the context of EPWSN.

Consider a multi-hop bulk transfer from node s to t. Supposing that we can modify the single packet-based schemes Flush and PIP to operate on top of an asynchronous MAC, the fastest sending rate that a transmitting node v can achieve is to transmit once every wakeup of its successor node w, or $1/(T_{\mathrm{L}}+T_{\mathrm{S}})$ (*cf.* Figure 1). This is because v needs to wait for the ACK before

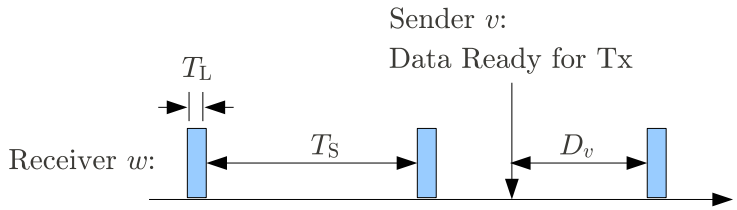


Figure 1: Illustrating the pre-transmission overhead of a transmission from v to w, denoted by D_v. The shaded boxes denote the wakeup intervals of w.

it can transmit the next packet and obviously, it can only receive the ACK when w is awake. Because of rate control, v does not transmit immediately (after receiving an ACK) and hence, v goes back to sleep. Once v transmits later (after an interval that is dependent on its packet sending rate), it must wait again for w to be awake to receive the ACK.

In addition to the low throughput, every packet transmission will have a very high pre-transmission overhead. To see why this is the case, consider Figure 1. The moment v becomes ready to transmit, it needs to wait for the next wakeup interval of w, which is D_v seconds into the future. If the probability of a packet becoming ready for transmission at v is the same any time, then

$$D_v \sim \mathcal{U}(0, T_{\mathrm{S}}), \tag{1}$$

where $\mathcal{U}(0, T_{\mathrm{S}})$ denotes the uniform distribution in $[0, T_{\mathrm{S}}]$. From (1), we can see that on the average, the pre-transmission overhead is $T_{\mathrm{S}}/2$. Hence, transmitting a single packet yields an average efficiency of $\tau/(\tau + T_{\mathrm{S}}/2) = 2\tau/(2\tau + T_{\mathrm{S}})$, where τ is the transmission time of a packet.

The use of packet trains can clearly remedy the deficiencies of single packet-based schemes. Because duty cycling somewhat limits the opportunities at which nodes can exchange packets, it makes sense to transmit as many packets as possible at every opportunity to improve efficiency. For clarity, we define the notion of a packet train in the context of asynchronous duty cycling as follows:

Definition 2 (Packet Train). *A series of packet transmissions, where only the first packet transmission is preceded by a pre-transmission overhead.*

If L packets are transmitted in a train, the average efficiency increases to $2L\tau/(2L\tau + T_{\mathrm{S}})$ while the throughput rises to $L/(T_{\mathrm{L}} + T_{\mathrm{S}})$ which is L times that of the single packet transmission approach. There are however several

problems that need to be addressed in the use of packet trains in EPWSNs where the nodes have strict duty cycle constraints:

Problem 1. *What packet train length should a transmitting node use, given the duty cycle constraints of itself and the receiving node?*

Problem 2. *For relay nodes, how should they allocate their respective duty cycles between packet train reception and packet train transmission?*

Problem 3. *Every node along the transfer path needs to periodically review the duty cycle allocation (and hence packet train lengths) to adapt to changes in duty cycle target and wireless link quality, and attain optimal performance over time.*

3. Pump-and-Nap Design

In this section, we introduce Pump-and-Nap, a bulk transfer scheme that addresses the challenges enumerated in the preceding section. In the design of Pump-and-Nap, we assume that at most one bulk transfer is permitted at any given time. This is the usual *modus operandi* in data collection, as simultaneous transfers cause *inter-flow interference* which can severely degrade the throughput performance [9] and in severely-constrained sensor nodes, this may entail excessive resource consumption leaving insufficient resources for sensing and data processing. The recommended strategy is to let the gateway or sink node initiate all data transfers to ensure that at most one transfer is on-going at any point in time.

While this paper focuses on single asynchronous bulk transfers, Pump-and-Nap can nevertheless be extended to support simultaneous transfers. At the end of this section, we describe one approach on how this extension can be undertaken. In the following, we introduce the fundamental design parameters of the proposed scheme.

Epoch. Time is divided into epochs with fixed duration T. The nodes need not be synchronized, *i.e.*, the start of epochs in nodes u and v need not occur simultaneously. The main reason for dividing time into epochs is to facilitate "periodic review" of Pump-and-Nap operating parameters at the start of every epoch.

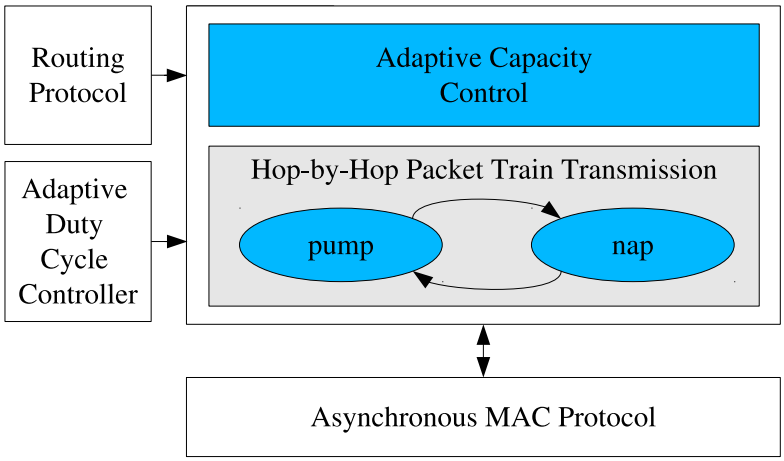


Figure 2: PUMP-AND-NAP architecture.

Target Duty Cycle. The nodes employ adaptive duty cycling to balance the dynamic energy supply and demand [6, 7, 8]. We let $\delta_v(k)$ denote the target duty cycle of v in epoch k which indicates the fraction of time that v can be active for reception and transmission. Note that $\delta_v(k) \in [0, 1]$.

MAC Protocol. PUMP-AND-NAP is designed to work with any asynchronous scheduling scheme that supports back-to-back packet transmissions or packet trains. We employ X-MAC [21] because of its implementation availability in TinyOS and more importantly, it supports packet trains. In the TinyOS implementation, this is possible because a duty-cycled node waits for a specified amount of time (`DELAY_AFTER_RECEIVE`) after its last packet reception before going back to sleep.

3.1. Architecture

Figure 2 shows the architecture of PUMP-AND-NAP with the major functional blocks. The two main functions provided by PUMP-AND-NAP are hop-by-hop packet train transmission using the pump and nap strategy, and dynamic computation of packet train length using adaptive capacity control. The former will be elaborated in Section 3.2 while the latter will be discussed in detail in Section 3.3.

PUMP-AND-NAP is specifically designed for dynamic duty cycling sensor networks and as such, it is assumed that an adaptive duty cycle controller provides the optimal operating duty cycle. Nevertheless, PUMP-AND-NAP

can also be used even in static duty cycle scenarios, as will be elaborated at the end of this section. In either case, PUMP-AND-NAP's goal is to ensure that the radio duty cycle will comply with the stipulated duty cycle to ensure long-term sustainability. To perform packet forwarding, PUMP-AND-NAP requires the knowledge of the successor node which can be obtained through a routing protocol. Finally, to control the wakeup scheduling of the wireless transceiver and perform efficient packet transmissions, PUMP-AND-NAP relies on an asynchronous MAC protocol that supports back-to-back packet transmissions.

3.2. Operation

PUMP-AND-NAP is a forwarding technique that can be used in conjunction with existing bulk transport protocols. As such, PUMP-AND-NAP focuses on two areas: (*i*) the computation of packet train lengths, and (*ii*) the manner by which packet trains are exchanged at every hop, from the source to the sink. In what follows, we describe the operation of PUMP-AND-NAP in a multihop bulk transfer from s to t along a path P_{st}. We assume that every node has a queue for storing packets. We also assume that the transfer has been initiated, and that every node $v \in P_{st}$ has started the operation of its *adaptive controller*, the details of which are presented in Section 3.3. For now, it is sufficient to know that the adaptive controller is responsible for computing $r_v(k)$ and $t_v(k)$ at every epoch k, the maximum number of packets that v can receive and transmit, respectively, given its current duty cycle $\delta_v(k)$.

To commence the transfer, s starts a nap cycle timer which will time out after T seconds (1 epoch). Node s then sends a *train request* to its successor node, say v. When v receives the request, it sends back a *train reply* to s indicating $r_v(k)$, the maximum number of packets that v can receive in the current epoch. Node s then *pumps* at most $\rho_s(k)$ packets back-to-back to v, where $\rho_s(k) = \min[t_s(k), r_v(k)]$. After this *pumping session*, s takes a *nap*, *i.e.*, stops transmissions until the next cycle. If $r_v(k) < Q_v(k)$, where $Q_v(k)$ is the queue length of v at k, then the remaining packets will stay in its queue and will be transmitted in the next epoch.

3.2.1. Basic Packet Train Forwarding

Let us look at how an arbitrary relay node v will perform packet transmissions. After receiving a train of packets from its predecessor node u, v performs its own pump-and-nap transmission strategy to its successor node

w. That is, v sends a train request to w. After receiving a train reply which indicates $r_w(k)$, v pumps at most $\rho_v(k)$ packets back-to-back to w, where

$$\rho_v(k) = \min[t_v(k), r_w(k)],$$

and immediately takes a nap after this. Note a subtle difference between how s and v performs the pump-and-nap strategy: while s uses a nap timer to trigger pumping sessions, v does not employ any such timer. This is because v's trigger for its pumping session is the end of its packet train reception from u. In the ideal case, the duty cycle usage of v, denoted by $d_v(k)$, is

$$d_v(k) := \frac{\tau_v(k)}{T}, \tag{2}$$

where $\tau_v(k)$ is the total time that v has been active. This includes the packet train reception time from u, pre-transmission overhead, train request/reply overhead, and packet train transmission time to w.

3.2.2. Wakeup-Synchronized Packet Train Forwarding

In the preceding approach, v commences packet train transmission to w immediately after completing a packet train reception from u, regardless of whether w is asleep or awake. Note that it is possible for v to optimize its duty cycle usage by timing its transmission to begin at the moment that w wakes up. This requires v to know the exact wakeup intervals of w. But for transferring large bulk data, this overhead is justified because it will reduce, if not eliminate, the pre-transmission overhead. This will result in v consuming a lower duty cycle for the same packet train length.

Regardless of whether the basic or wakeup-synchronized packet train transmission is employed, the hop-by-hop packet train transmission strategy is repeated until the sink node t. PUMP-AND-NAP relies on the link layer for reliability and error detection. When a node v fails to receive an ACK for the latest transmitted packet (that is part of the train), and after exhausting the specified retransmission limit, the packet being transmitted is *not* dropped; rather, v stops the packet train transmission and immediately takes a nap. Note that when a packet train transmission is abnormally terminated due to such failures, the subsequent packet train transmission will commence from the last unsuccessful packet.

3.3. Adaptive Capacity Control

We shall now discuss the design of an adaptive controller that can simultaneously address the three problems posed in Section 2. In our design, we adapted the methodology described by Goodwin and Sin [28]. First, we seek a dynamic model that describes the evolution of the quantity that we want to control, *i.e.*, the node duty cycle usage. This dynamic model will contain an unknown system parameter. Second, we formulate the problem as two parts: estimation of the unknown parameter, and calculation of optimal control law using the parameter estimate.

The motivating problem at node v is to determine $r_v(k)$ and $t_v(k)$, the maximum number of packets that v can receive and transmit, respectively, given its current duty cycle $\delta_v(k)$. Taken together, the sum of $r_v(k)$ and $t_v(k)$ is the *node capacity* $C_v(k)$, that is,

$$C_v(k) := r_v(k) + t_v(k).$$

The goal of the adaptive capacity controller is to let the duty cycle usage $\{d_v(k)\}$ track the target duty cycle $\{\delta_v(k)\}$, for all epoch k, while at the same time maximize $\{C_v(k)\}$.

3.3.1. Input-Output Model

As a matter of convention, we assume that control decisions are done at the start of every epoch k. There is a unit epoch delay before the effects of the control decision can be observed. Thus, if v decides to receive $r_v(k)$ packets and transmit $t_v(k)$ packets at epoch k, we can only ascertain the corresponding duty cycle usage at epoch $k+1$, denoted by $d_v(k+1)$, which can be obtained by measuring the active time of the radio and using (2).

If α and β are the duty cycle 'consumed' for every *successful* packet reception and transmission, respectively, then $d_v(k+1) = \alpha r_v(k) + \beta t_v(k)$. Note however that this formulation ignores two overheads: (i) the pre-transmission overhead as discussed in Section 2.1; and (ii) the duty cycle used for the intervals at which v wakes up to listen for transmissions (for transmitter-initiated schemes) or transmit beacons (for receiver-initiated schemes). Denoting U_v for the former and L_v for the latter, we have

$$d_v(k+1) = \alpha r_v(k) + \beta t_v(k) + U_v(k+1) + L_v. \tag{3}$$

The parameter L_v can be treated as a constant since v incurs the same overhead at every epoch. Because there is at most one packet train transmission

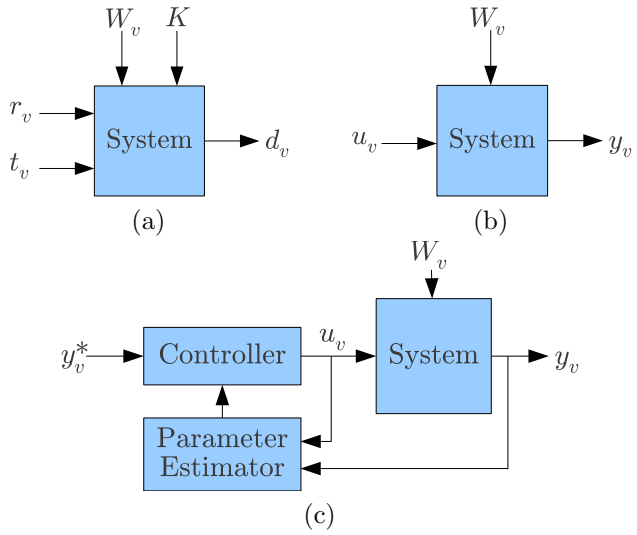


Figure 3: Modeling of the system for adaptive feedback control: (a) input-output model; (b) simplified model; and (c) system with adaptive controller.

every epoch, the duty cycle usage of pre-transmission overhead is simply

$$U_v(k) = \frac{D_v(k)}{T},$$

where $D_v(k)$ is a random variable defined in (1). With this, $U_v(k)$ is effectively uniform in $[0, T_S/T]$. Note that the index of U_v is $k+1$ in (3) because of the fact that its effect is only measured together with the measurement of $d_v(k+1)$. $U_v(k)$ can actually be expressed as the sum of a constant and uniform random variable, that is,

$$U_v(k) = \frac{T_S}{2T} + W_v(k),$$

where $W_v(k) \sim \mathcal{U}(-T_S/2T, T_S/2T)$. Lumping together all the constants as K, (3) can be rewritten as

$$d_v(k+1) = \alpha r_v(k) + \beta t_v(k) + W_v(k+1) + K, \tag{4}$$

where $K := L_v + T_S/(2T)$. A block diagram representation of (4) is shown in Figure 3(a).

3.3.2. Simplified Model

In what follows, we refine (4) to address important considerations such as queue stability and capacity maximization.

Queue Stability. Ensuring that queues are stable is important to reduce packet loss due to buffer overflows. For the queue at v to be stable in the long-run, the number of incoming packets must be at most equal to the number of packets that v can transmit [29], or

$$r_v(k) \leq t_v(k). \tag{5}$$

Capacity Maximization. As mentioned, a key objective of the adaptive controller is to maximize node capacity $C_v(k)$. Given the constraint provided by (5), it is easy to see that in order to maximize $C_v(k)$, v must be allowed to receive as much as possible. That is,

$$r_v(k) = t_v(k). \tag{6}$$

From (6), by letting $u_v(k)$ denote either $r_v(k)$ or $t_v(k)$ and introducing a parameter b, that is

$$u_v(k) := r_v(k) = t_v(k), \tag{7}$$

$$b := (\alpha + \beta), \tag{8}$$

(4) can be rewritten as

$$d_v(k+1) = bu_v(k) + W_v(k+1) + K. \tag{9}$$

We remark that as b encapsulates the "duty cycle cost" to successfully receive and transmit a packet, it is affected by the variations of its incoming and outgoing wireless links. For convenience, we make the following change of variables:

$$y_v(k) := d_v(k) - K. \tag{10}$$

Substituting this into (9) yields

$$y_v(k+1) = bu_v(k) + W_v(k+1) \tag{11}$$

which is our desired form and is pictorially depicted in Figure 3(b). Note that the original control objective is to let $\{d_v(k)\}$ track $\{\delta_v(k)\}$, for all epoch k. But because $d_v(k)$ is 'hidden' in (11) due to the change of variables, we also define the following for convenience:

$$y_v^*(k) := \delta_v(k) - K. \tag{12}$$

The above essentially means that the equivalent control objective is for $\{y_v(k)\}$ to track $\{y_v^*(k)\}$.

3.3.3. Estimation and Control

We shall now use (11) to obtain the optimal control $u_v(k)$ that maximizes the capacity of v while ensuring that the duty cycle usage $\{y_v(k)\}$ tracks the target duty cycle $\{y_v^*(k)\}$, for all epoch k. We structure the control system as in Figure 3(c). A key component of the control system is the parameter estimator, which is responsible for estimating the value of b and essentially makes the controller adaptive. Because (11) is linear and the "noise" term has zero mean (*i.e.*, $\mathsf{E}[W_v(k)] = 0$), the least squares estimate of b, denoted by $\hat{b}$, is given by [30]

$$\hat{b} = \frac{\sum_{i=0}^{k-1} y_v(i+1)u_v(i)}{\sum_{i=0}^{k-1} u_v^2(i)}. \tag{13}$$

The optimal control law can be obtained by invoking the *principle of certainty equivalence* [30]. It means that we use $\hat{b}$ as though it were the true parameter b. Hence, $u_v(k)$ can be obtained by replacing $y_v(k+1)$ and b in (11) with $y_v^*(k+1)$ and $\hat{b}$, respectively, and cancelling $W_v(k)$. This yields $u_v(k) = y_v^*(k+1)/\hat{b}$. Noting that $u_v(k)$ must be an integer, we simply take the floor and obtain

$$u_v(k) = \left\lfloor \frac{y_v^*(k+1)}{\hat{b}} \right\rfloor. \tag{14}$$

3.3.4. Estimation and Control for Wakeup-Synchronized Scheme

The estimator $\hat{b}$ in (13) and control law $u_v(k)$ in (14) are applicable when the basic packet train forwarding scheme is employed. When the wakeup-synchronized approach is used, we need to slightly modify the parameter estimate and control law. Note that in the latter, the pre-transmission overhead vanishes, hence, we can rewrite (9) as

$$d_v(k+1) = bu_v(k) + X_v(k+1) + L_v,$$

where $X_v(k)$ denotes the uncertainty between the time that v commences packet transmission and the exact time that w exactly wakes up. This uncertainty is present because even though v transmits at the wakeup intervals of w, errors in clocks of both v and w are still possible. Following the same arguments as in Section 3.3.1 and assuming that $X_v(k)$ are i.i.d. for all k with mean $\bar{X}_v$, the above can be rewritten as

$$d_v(k+1) = bu_v(k) + \tilde{W}_v(k+1) + \tilde{K},$$

where $\tilde{W}_v(k)$ is a zero-mean random variable and $\tilde{K} = L_v + \bar{X}_v$. We can therefore define analogues of (10) and (12) as:

$$\tilde{y}_v(k) := d_v(k) - \tilde{K} \tag{15}$$

$$\tilde{y}^*_v(k) := \delta_v(k) - \tilde{K} \tag{16}$$

Finally, the corresponding parameter estimator for the wakeup-synchronized packet train forwarding is

$$\tilde{b} = \frac{\sum_{i=0}^{k-1} \tilde{y}_v(i+1)u_v(i)}{\sum_{i=0}^{k-1} u_v^2(i)} \tag{17}$$

while the optimal control law is given by

$$u_v(k) = \left\lfloor \frac{\tilde{y}^*_v(k+1)}{\tilde{b}} \right\rfloor. \tag{18}$$

We have just completed the design of the adaptive controller at v, which will allocate $u_v(k)$ packets for both reception and transmission in the current epoch k. To ensure that v can store all the packets in a train, it must limit the number of packets that it indicates to its predecessor node to

$$r_v(k) = \min[u_v(k), Q - Q_v(k)],$$

where Q is the maximum queue size and $Q_v(k)$ is the queue length at v. Before ending the discussion, we remark the following desirable properties of the controller.

Applicability to any node type. The controller was initially designed for a relay node v. However, the use of a single control $u_v(k)$ makes the model applicable for the source and sink nodes as well. In the latter two types, $u_v(k)$ provides the optimal transmit and receive allocations, respectively, without any change.

Adaptation to wireless link variations. As mentioned, the parameter b encapsulates the effect of link quality variations. Since b is continuously estimated, the control $u_v(k)$ also automatically adjusts to the link quality fluctuations.

Support for synchronous and asynchronous MAC. Our design assumed that the underlying MAC is asynchronous. By removing the third term (due to pre-transmission overhead) in (3) and slightly re-defining $y_v(k)$ and $y^*_v(k)$, we can use the controller in conjunction with synchronous MAC protocols.

Usability in static and dynamic duty-cycling. In the development of the controller, we did not make any assumption about the target duty cycle $\delta_v(k)$, other than $\delta_v(k) \in [0, 1]$. As such, the controller can also be used in situations where $\delta_v(k)$ is constant, *i.e.*, static duty cycling.

3.4. Supporting Simultaneous Transfers

At the start of Section 3, we remarked that PUMP-AND-NAP is designed to work well in data collection scenarios wherein the bulk data transfer is managed by a single entity (*i.e.*, the gateway node) and that this single entity ensures that every node in the network is involved in at most one bulk transfer. Nevertheless, PUMP-AND-NAP can be extended to support simultaneous data transfers through the following modifications.

Suppose that a node v is currently supporting a single bulk transfer, and it currently allocates $r_v(k)$ for reception and $t_v(k)$ for transmission. When v receives another *train request*, it simply divides $r_v(k)$ equally into 2. Likewise, if the successor node is different, v divides $t_v(k)$ equally into 2. This process can be repeated for every new bulk transfer, dividing $r_v(k)$ and $t_v(k)$ equally among the distinct number of predecessor and successor nodes, respectively.

We note that the above modifications pose some challenges on the performance of the adaptive capacity controller especially in the case of the basic forwarding scheme. This is because in the design of the controller, only one pre-transmission overhead per epoch is considered. If a node v needs to perform packet train transmissions to several successor nodes, then every such successor node will entail a pre-transmission overhead. As such, the extension of PUMP-AND-NAP to support simultaneous bulk transfers will only work well for the wakeup-synchronized forwarding scheme. The basic scheme can only be employed in scenarios where there is a single data collection point.

4. Implementation and Experimental Design

We conducted experiments to characterize the performance of PUMP-AND-NAP and to compare it with state-of-the-art bulk transfer schemes. In this section, we briefly describe our implementation of PUMP-AND-NAP, the experiment settings used, and details about the experiment setups.

4.1. Pump-and-Nap *Implementation*

We implemented Pump-and-Nap in TinyOS 2.1.2 [15] and deployed it in TelosB motes. TelosB uses the CC2420 (an IEEE 802.15.4-compliant radio) which is duty cycled by a component called `PowerCycle`. To measure the duty cycle usage, we implemented two event "hooks" that are invoked from `PowerCycle`, namely `radioStarted()` and `radioStopped()` to indicate the exact instances at which the radio is turned on and off, respectively.

To implement the wakeup-synchronized scheme, we used a TinyOS component called `CC2420TimeSyncMessageC` which enables a node v to inform another node u of the exact time at which an event has occurred. Note that this timing information is piggy-backed in data packets, thus, no additional message overhead is generated. In our implementation, v always piggy-backs its last wakeup interval in any data packet transmission. This enables a receiving node u to deduce all future wakeup intervals of v, since T_{L} and T_{S} (*cf.* Figure 1) are fixed.

4.2. Experiment Settings

Pump-and-Nap and the underlying X-MAC protocol have several important parameters that need to be specified prior to deployment. For X-MAC, the wakeup interval T_{L} is set to 15 ms while the sleep interval T_{S} is varied between 485, 235, and 110 ms. These values correspond to wakeup rates of 2, 4 and 8 wakeups/second, respectively. The 15 ms wakeup interval was chosen because it provided a good trade-off between overhead and preamble reception probability. For reliability, we used CC2420 software-based ACKs (default setting) and set the retry limit to 7.

For Pump-and-Nap, the epoch duration T and packet buffer space Q are the two key parameters. We chose $T = 3$ s in our evaluation. Selecting a lower T requires more frequent computations but faster reaction to environmental changes while a longer T requires less frequent computations but slower reaction to environmental changes. T also has a direct impact on the efficiency of packet trains. A shorter (longer) T implies shorter (longer) packet trains and therefore lower (higher) efficiency. However, supporting longer packet trains requires nodes to maintain larger packet buffer space. In this work, we used a buffer size of 60 packets which was more than sufficient for the tested scenarios.

We focused our evaluation on the performance of the adaptive controller and packet train forwarding scheme so we used fixed network topologies. In addition, we used a packet size of 64 bytes for transmitting fragments of the

bulk data, which is generated on-the-fly at the source nodes. The choice of 64-byte packet is guided by recent results which show that in the context of IEEE 802.15.4, this size provides a good trade-off between efficiency and loss probability[31, 32].

Our implementation of PUMP-AND-NAP, together with the above-mentioned packet buffer and all the necessary software stack, generates a firmware that requires 26,960 bytes ROM and 9,944 bytes RAM. Note that this can fit in a TelosB mote which has 48 kilobytes ROM and 10 kilobytes RAM.

4.3. Experiments

We conducted three sets of experiments to rigorously evaluate the performance of PUMP-AND-NAP and to compare it with state-of-the-art bulk transfer schemes. Experiments to characterize the performance of PUMP-AND-NAP and simulate energy harvesting scenarios were carried out in the 139-node Indriya indoor testbed [16] while energy-harvesting experiments were conducted in indoor and outdoor locations.

4.3.1. Characterization of PUMP-AND-NAP *Performance*

To evaluate the performance of the controllers proposed in Sections 3.3.3 and 3.3.4, we performed experiments in the Indriya testbed involving 3 nodes, namely a source, a relay, and a sink. We selected 10 sets of combinations from the testbed, where the link delivery probabilities from source to relay, and from relay to sink were more than 0.8. The source and sink duty cycles were fixed at 50% whereas the relay duty cycle was changed every minute to a random value in $[1\%, 30\%]$ that had not yet been previously selected. This setup ensured that all possible duty cycle values in the range were tested, and that the relay duty cycle was the bottleneck. We did not use a sequentially increasing (or decreasing) relay duty cycle as we wanted to determine the response of the controller to abrupt duty cycle changes.

To see the effect of hop count, we ran experiments where the number of hops from the source to the sink is varied from 1 to 5 hops. For every hop count, we tested three duty cycle targets, namely 10%, 20%, and 30%. Each experiment was run for 1 minute and repeated 10 times.

4.3.2. Energy Harvesting Experiments

We performed experiments involving a real energy-harvesting node to determine whether PUMP-AND-NAP can indeed provide sustainable bulk transfer. The setup involves a 2-hop bulk transfer: the source and sink nodes are

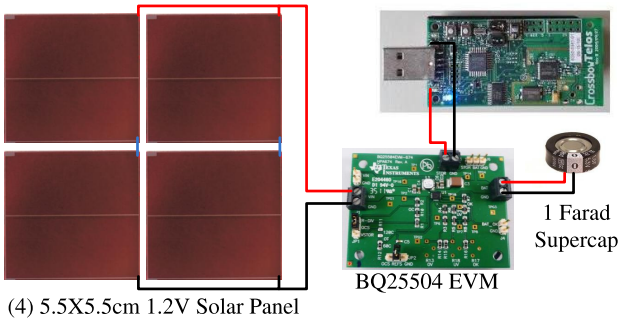


Figure 4: Energy harvesting experimental setup.

powered through the USB port while the relay node uses energy-harvesting. Figure 4 shows the schematic diagram of the energy-harvesting relay node. It uses 4 solar panels that can generate up to 22 mW, and a 1 Farad supercap as energy store[2]. BQ25504 EVM [33] is a power management circuit that controls the supercap charging and energy supply to the mote. It is configured to charge the supercap to 3.1 V. We implemented a simple *voltage-duty cycle mapping* to generate the target duty cycle $\delta(k)$ at every epoch k, given by

$$\delta(k) = \max[V(k) - 2.5, 0],$$

where $V(k)$ denotes the supercap voltage at epoch k. This simple mapping allows a maximum duty cycle of 60% since $V(k) \leq 3.1$. Note also that when $V(k) < 2.5$, $\delta(k) = 0$ because based on our observations, the node stops functioning when the supercap voltage drops below 2.5 V.

Two energy harvesting scenarios were used: (a) *indoor scenario* – solar panel was exposed to a lamp with 10 klux illuminance; and (b) *outdoor scenario* – solar panel was exposed under direct sunlight with 100 klux illuminance. (The Extech HD450 Lux Meter was used to measure the illuminance.) Every scheme was run 10 times, and every run was scheduled for at most 2 hours for practical reasons. For fairness, all experiments under (b) were performed when the sun was unobstructed by any cloud. We checked that the supercap was at 3.1 V prior to the start of every run.

We compared PUMP-AND-NAP with state-of-the-art bulk transfer techniques presented in Section 2.2: (i) packet train-based transmissions which

[2]Compared to SolarBiscuit [2], our energy-harvesting node has the same storage capacity and roughly four times the energy harvesting capacity.

is employed in [12], and (*ii*) Flush [9]. For the former, we tested both unsynchronized and wakeup-synchronized.

Rationale for Using 10 klux and 100 klux. We used these values because based on our measurements, outdoor daytime illuminance ranges from 10–100 klux during a fair sunny day. It is easy to consistently obtain 100 klux outdoors (which happens when the sun is not obstructed by any cloud at around 1-4 pm) but difficult to obtain a consistent 10 klux. Hence, we performed indoor experiments with a lamp that was placed at a distance such that the illuminance reaching the solar panel is around 10 klux. By choosing these two values, we can use the results to infer that (*i*) the scheme should be able to provide sustainable bulk data transfer within 10–100 klux of illuminance on a fair sunny day, and (*ii*) the scheme automatically adjusts the throughput according to the variations in energy availability.

4.3.3. Energy Harvesting Simulations

To further study the effect of energy harvesting rate and path length in a controlled setting, we performed experiments in the Indriya testbed, wherein the energy harvesting and consumption processes are emulated.

Harvesting Process. We model the energy harvesting process after the energy harvesting node shown in Figure 4. To characterize its harvesting process, the load (*i.e.*, the TelosB mote) is disconnected and the voltage across the supercap is sampled at every epoch (3 s), as the solar panel is exposed to 10 klux, and then repeating this for 100 klux light. The supercap is first discharged to around 2 V prior to the characterization. Figure 5 shows the voltage over time across the supercap. We only consider the charging rate at and above 2.5 V because once a node goes below this voltage, it becomes non-operational. Now, from elementary circuit theory,

$$I(t) = C\frac{dV(t)}{dt},$$

where $V(t)$ is the supercap voltage at time t, C is the capacitance, and $I(t)$ is the current flowing in or out of the supercap. Since $C = 1$, the harvesting rate is simply $I(t) = dV(t)/dt$ or the slope of $V(t)$. From Figure 5, we can see that the voltage increases almost linearly with time from 2.5–3.1 V, suggesting that the harvesting rate can be approximated by a constant value within this region. Using the measurements obtained, the average charging

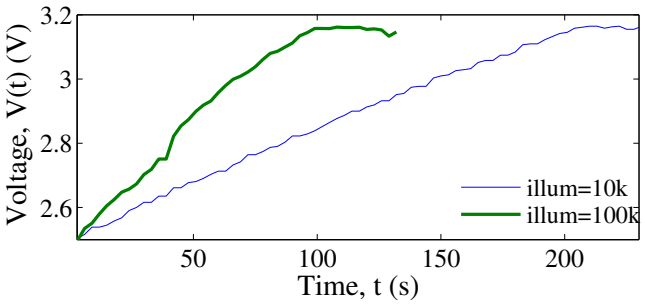


Figure 5: Voltage across the supercap, under 10 klux and 100 klux illuminance.

current at 10 klux and 100 klux are 3.81 and 6.48 mA, respectively. In the simulations, we vary the harvesting rate from 3.8–6.5 mA to mimic the above conditions.

Energy Consumption Process. To emulate the energy consumption, we measured the current consumption of TelosB at 3 V in three modes of operation and obtained the following: (*i*) MCU is active and radio is in deep sleep: $I_1 = 2$ mA; (*ii*) MCU is active and radio is in receive mode: $I_2 = 23$ mA; and (*iii*) MCU is active and radio is in transmit mode at 0 dBm: $I_3 = 21$ mA. These are similar to the findings in [34].

We implemented a TinyOS module to emulate the above harvesting and consumption processes. Briefly, the supercap voltage $V(k)$ evolves according to this difference equation:

$$V(k) = \mathcal{Q}_{\mathrm{C}}(k-1) - \mathcal{Q}_{\mathrm{D}}(k-1) + V(k-1), \tag{19}$$

where $\mathcal{Q}_{\mathrm{C}}(k)$ denotes the charge accumulated at k due to the energy harvesting process while $\mathcal{Q}_{\mathrm{D}}(k)$ denotes the total discharge at k due to the energy consumption process. Eq. (19) is the discrete version of the well-known relation $V(t) = \frac{1}{C}\int_{t_0}^{t} I(t)dt + V(t_0) = \int_{t_0}^{t} I(t)dt + V(t_0)$ since $C = 1$ in our case. To mimic the uncertainty in the harvesting process,

$$\mathcal{Q}_{\mathrm{C}}(k) = I_{\mathrm{C}}T + \omega(k),$$

where $I_{\mathrm{C}} \in [3.8, 6.5]$ mA is the simulated charging current, T is the epoch duration, and $\omega(k)$ is a random number generated using the `RandomC` component. Note that $\omega(k) \sim \mathcal{U}(-\Omega, \Omega)$, where Ω is chosen to capture the variability of the harvesting rate on an epoch by epoch basis. We make

this simplification because TinyOS only provides modules that can generate uniformly distributed random numbers. Meanwhile,

$$\mathcal{Q}_{\mathrm{D}}(k) = I_1 T + I_2 \tau_{\mathrm{rx}}(k) + I_3 \tau_{\mathrm{tx}}(k),$$

where $\tau_{\mathrm{rx}}(k)$ and $\tau_{\mathrm{tx}}(k)$ are the times spent in receive and transmit modes at epoch k, respectively.

We ran simulations in the Indriya testbed for PUMP-AND-NAP (wakeup-synchronized), packet train (wakeup-synchronized) and Flush, and fixed the X-MAC wakeup rate to 4 per second. All nodes in the simulations are emulated to be powered by energy harvesting. Every scheme was run 10 times.

5. Results

In this section, we present and discuss the results that we have obtained from the three sets of experiments that we have conducted, the details of which are presented in Section 4.3. The error bars in plots indicate the 95% confidence interval.

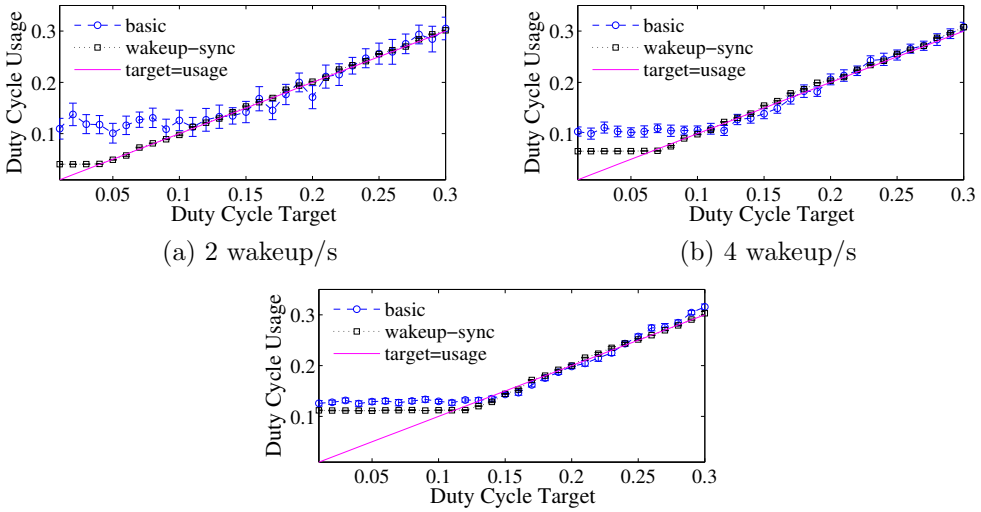
5.1. Characterization of PUMP-AND-NAP *Performance*

The first set of experiments was aimed at evaluating the performance of the controller designs presented in Sections 3.3.3 and 3.3.4 and was divided into two parts. The first part focused on the dynamic performance (*i.e.*, performance with respect to changing duty cycle targets) while the second part focused on the multihop performance.

5.1.1. Dynamic Performance

Figures 6 and 7 show the duty cycle usage and relay capacity, respectively, as functions of the duty cycle target, of *basic* and *wakeup-synchronized* under different X-MAC wakeup rates. These results indicate that both schemes can follow the duty cycle target, except at lower duty cycles. The latter is due to the X-MAC wakeup overhead and pre-transmission overhead (in the case of *basic*). To illustrate, at 4 wakeup/s, the wakeup overhead is $(4 \times 15)/1000 = 6\%$, hence we can see in Figure 6(b) that the usage of *wakeup-synchronized* does not go below 6%. For *basic*, there is an additional overhead of roughly $T_{\mathrm{S}}/(2T) = 235/(2 \times 3000) \approx 4\%$, hence, its usage is 10% at the minimum.

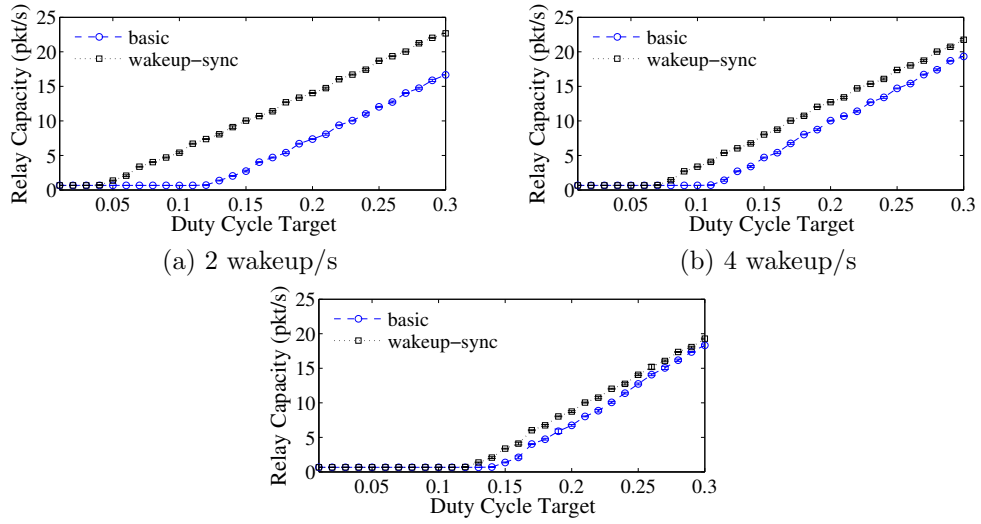
Another noticeable aspect is that *basic* shows higher variability especially at lower wakeup rates while *wakeup-synchronized* provides highly consistent



(a) 2 wakeup/s (b) 4 wakeup/s

(c) 8 wakeup/s

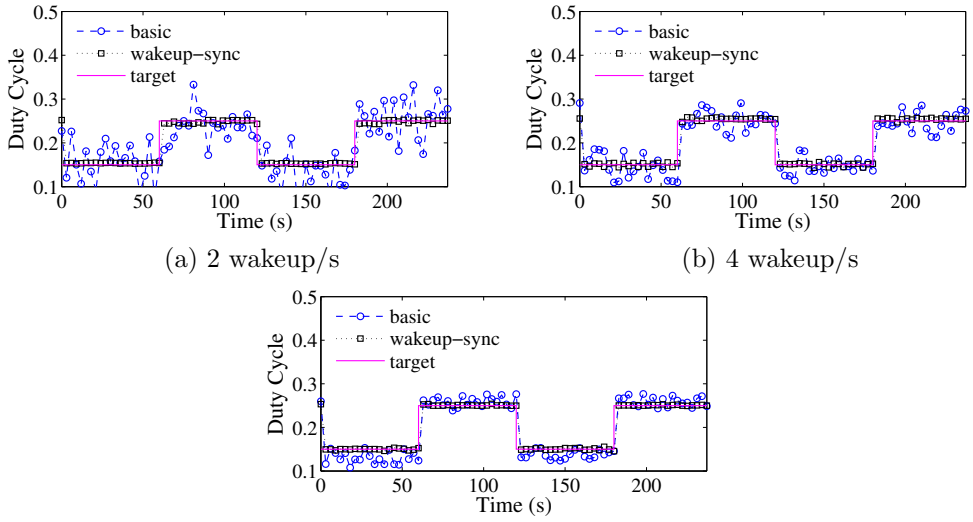
Figure 6: Comparing the duty cycle tracking performance of basic and wakeup-synchronized under different X-MAC wakeup rates.



(a) 2 wakeup/s (b) 4 wakeup/s

(c) 8 wakeup/s

Figure 7: Comparing the relay capacity of basic and wakeup-synchronized under different X-MAC wakeup rates.



(a) 2 wakeup/s

(b) 4 wakeup/s

(c) 8 wakeup/s

Figure 8: Snapshot of controller response when duty cycle target abruptly changes, under different X-MAC wakeup rates.

performance regardless of the X-MAC wakeup rate. The higher variability of *basic* is expected because the uncertainty due to the pre-transmission overhead is significantly higher than the clock uncertainty in *wakeup-synchronized*. Moreover, the former is sensitive to the X-MAC wakeup rate, *i.e.*, at lower wakeup rates, the variability is higher because T_{S} is larger (*cf.* Figure 1). To emphasize the sensitivity of *basic*, we show a snapshot of the controller response when the duty cycle target changes abruptly from 15% to 25% (and vice versa) every minute in Figure 8. Note the stable dynamic response of *wakeup-synchronized*, regardless of the X-MAC wakeup rate. Compare this with *basic* which is highly oscillatory because of the high pre-transmission overhead uncertainty. The lower the wakeup rate, the higher the uncertainty which ultimately results in wider oscillations.

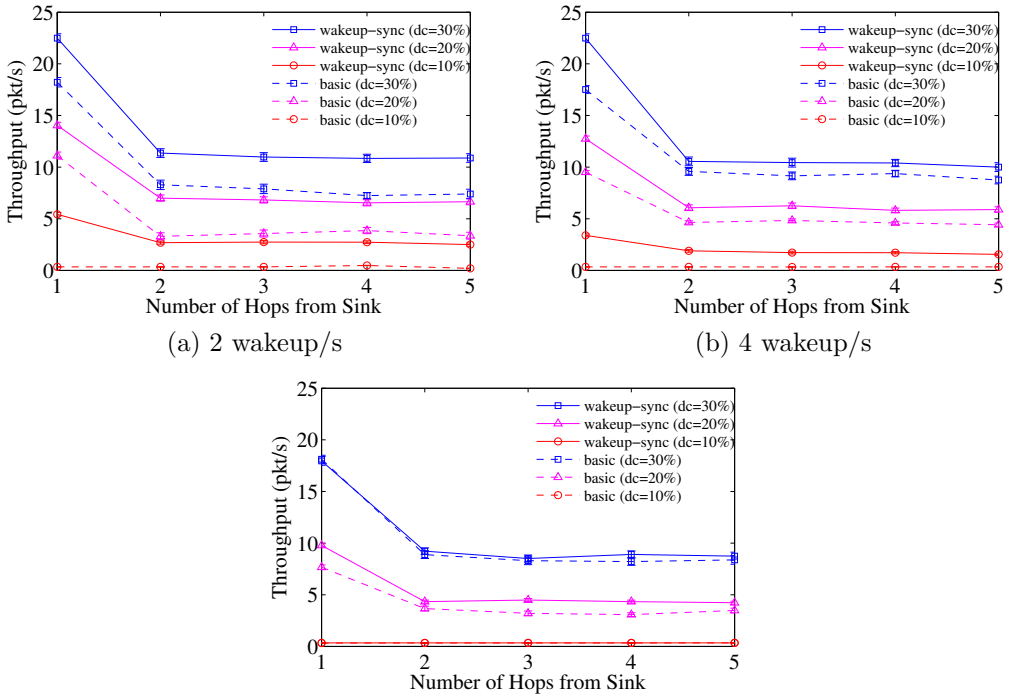
With respect to the relay capacity, we can observe that both schemes provide consistent (low variability) capacity. At lower duty cycles, both schemes yield negligible capacity because the X-MAC wakeup overhead and pre-transmission overhead (in the case of *basic*) use up the entire duty cycle. The advantage of synchronization is noticeable, as *wakeup-synchronized* shows better performance compared to *basic* in all wakeup rates due to the elimination of pre-transmission overhead. Its advantage is higher at lower

wakeup rates because of the lower X-MAC wakeup overhead in those settings. At 8 wakeup/s, the performance of both schemes are comparable because the X-MAC wakeup overhead becomes dominant.

5.1.2. Multihop Performance

Figure 9 shows the throughput of *basic* and *wakeup-synchronized*, for duty cycles of 10%, 20%, and 30%, as the hop count is varied from 1 to 5. Except for the settings that yielded negligible throughput (caused by the usage of the entire duty cycle for X-MAC wakeup overhead and pre-transmission overhead), we can see a big drop from 1 to 2 hops for the rest, with the latter throughput being just around half of the former. This is expected because for single hop transfers, the source (sink) does not need to allocate any duty cycle for reception (transmission) and that it can allocate its entire duty cycle for transmission (reception). For 2–5 hops, the throughput is almost the same for every scheme, due to the fact that the relay nodes are able to maximize the allocated duty cycle. Once again, we can see the distinct advantage of *wakeup-synchronized*, as its throughput is higher than *basic* especially at lower X-MAC wakeup rates. The lower throughput of *basic* is due to the pre-transmission overhead, which is relatively higher at lower wakeup rates.

The flat throughput results for 2–5 hops is counter-intuitive. We have expected it to decrease because as the number of hops increases, intra-flow interference worsens. To understand the result, we pictorially analyze the "airtime usage" of a 5-hop bulk transfer in Figure 10. We define airtime usage as the total time (in an epoch) that the nodes used for transmission and reception of packet train from the source to the sink. Suppose that the target duty cycle of all the nodes is 30%. Then node 1 will allocate all of its duty cycle, *i.e.*, 30% for transmission. Relay nodes 2, 3 and 4 will split their duty cycles accordingly, say 15% for reception and 15% for transmission, for simplicity. Finally, sink node 5 will allocate its entire 30% for reception. While node 1 could have utilized 30% to transmit to node 2, it will only be able to use 15% because node 2 limits the packet train transmission. Likewise, while node 5 can use 30% for reception, node 4 limits its usage to only 15%. Thus, the total airtime usage is 60% of the epoch duration which is the sum of the following: 15% from node 1 to 2; 15% from node 2 to 3; 15% from node 3 to 4; and 15% from node 4 to 5. We can use the same figure to analyze the airtime usage of 2, 3, and 4 hops to show that the airtime usage of these transfers are well below 100% and will therefore sidestep the



(a) 2 wakeup/s

(b) 4 wakeup/s

(c) 8 wakeup/s

Figure 9: Throughput performance of basic and wakeup-synchronized under different X-MAC wakeup rates.

problem of intra-flow interference. To clarify why intra-flow interference will not occur, note that because the airtime usage is below 100%, all packet train transmissions by nodes 2–4 have already completed before node 1 initiates a new packet train in the next epoch.

We now want to emphasize the following: for all transfers involving 2, 3, 4 and 5 hops, the relay nodes limit the packet train size or duration to 15% of the epoch duration. Since the source is allowed to initiate 1 packet train transmission every epoch, then the throughput is determined by the packet train duration. This explains the flat throughput for 2–5 hops.

5.2. Energy Harvesting Experiments

The second set of experiments involved the use of a real energy harvesting node, and was intended to determine the throughput and sustainability of Pump-and-Nap, packet train and Flush. Figures 11 and 12 show the throughput and mean time before the relay node failed due to energy

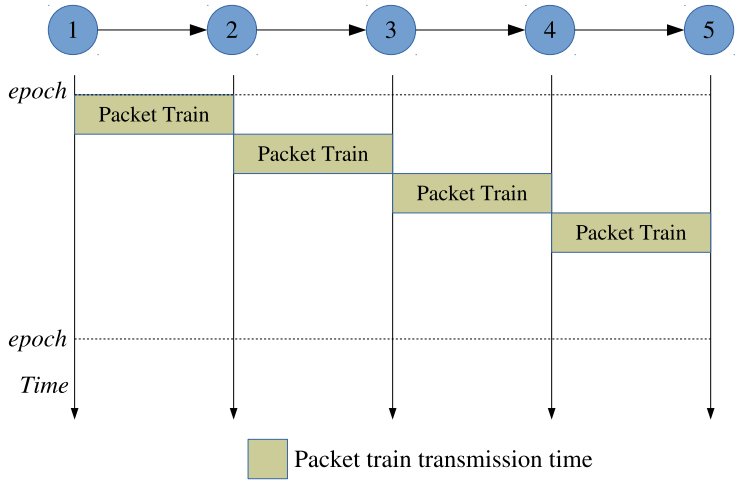
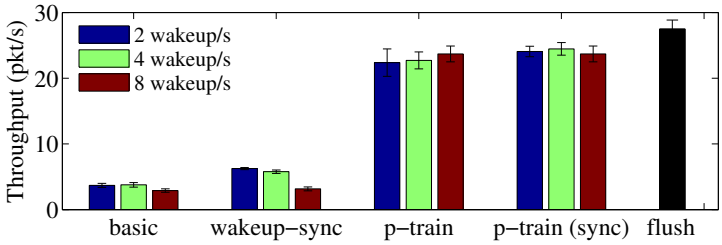


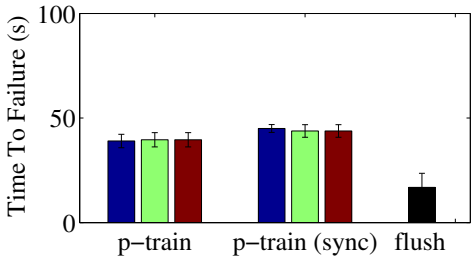
Figure 10: Airtime usage of a packet train transmission from the source to the sink. If the target duty cycle of the nodes is 30%, then each packet train requires 15% of the epoch. Hence the total airtime is $4 \times 15 = 60\%$ of the epoch duration.

exhaustion, in indoor and outdoor scenarios, respectively. *p-train* and *p-train (sync)* denote unsynchronized and wakeup-synchronized packet train forwarding schemes, respectively. In either scenarios, Flush yields the highest throughput at around 28 pkt/s because it does not incur sleep latency from duty cycling. However, the transfer is short-lived, lasting for only 16.8 s indoors and 36.4 s outdoors. Meanwhile, the use of packet trains can indeed improve the energy-efficiency of bulk transfer. Although its throughput is slightly lower than Flush by at most 18%, packet train can last more than twice that of the former in both illuminance conditions. Comparing packet train and wakeup-synchronized packet train, we can observe a slight advantage of the latter. While both schemes yield comparable throughput, the latter can last slightly longer by at most 17 s, due mainly to the energy savings from pre-transmission overhead.

As for Pump-and-Nap, it is the only scheme that can provide uninterrupted transfer, regardless of illuminance and X-MAC wakeup rate (the relay node did not run out of energy for the entire 2-hour experiment duration). It accomplishes this by adjusting its throughput according to the energy availability. This is evident in the results as we observe that the throughput of both *basic* and *wakeup-synchronized* in indoor experiments are around 1/2 that of outdoor experiments. Although Pump-and-Nap's best throughput

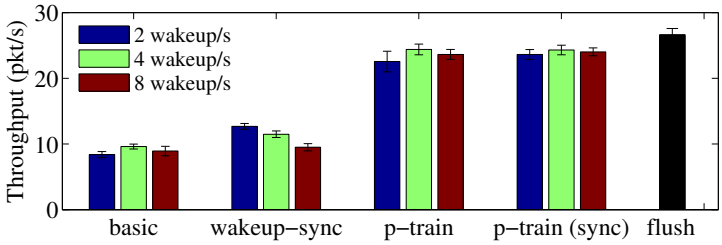


(a) Throughput

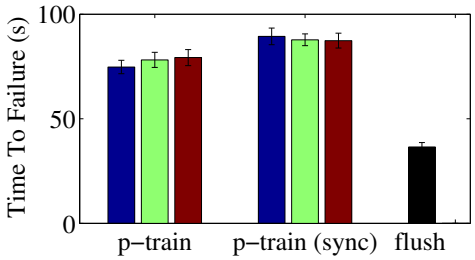


(b) Mean Time to Failure

Figure 11: Throughput of PUMP-AND-NAP, packet train and Flush, and mean time to relay node failure of the latter two, in indoor scenario (10 klux).



(a) Throughput



(b) Mean Time to Failure

Figure 12: Throughput of PUMP-AND-NAP, packet train and Flush, and mean time to relay node failure of the latter two, in outdoor scenario (100 klux).

is around 1/4 and 1/2 that of packet train in indoor and outdoor scenarios, respectively, the transfer can last for an indefinite amount of time. This will enable Pump-and-Nap to transfer bulk data of any size.

5.3. Energy Harvesting Simulations

The third and final set of experiments was purported to study the effects of energy availability and hop count on the throughput and sustainability of Pump-and-Nap, packet train and Flush.

5.3.1. Influence of Energy Availability

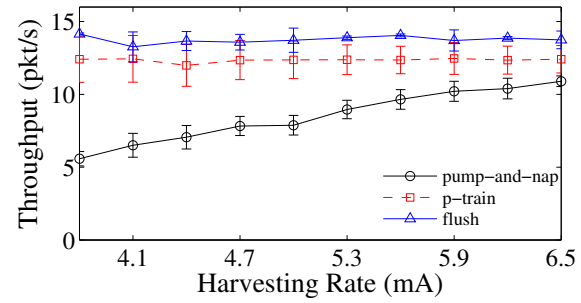
We conducted testbed simulations where the harvesting rate is varied from 3.8–6.5 mA, representing the energy that can be scavenged from 10–100 klux. Figure 13(a) shows the throughput of the three schemes, while Figure 13(b) shows the mean time to failure of packet train and Flush. Pump-and-Nap is not included in the latter plot because it can sustain the bulk transfer indefinitely. The results are obtained using a 5-hop bulk transfer.

The throughput results suggest that Pump-and-Nap is the only scheme that adapts to energy availability. While packet train and Flush respectively yields the same throughput regardless of the harvesting rate, Pump-and-Nap shows a throughput that increases as the harvesting rate increases. Specifically, its throughput almost doubles from 5.5 pkt/s at 3.8 mA to 10.9 pkt/s at 6.5 mA. This important result demonstrates the effectiveness of employing the adaptive capacity controller to automatically adjust the relay capacity of nodes according to what energy availability can support.

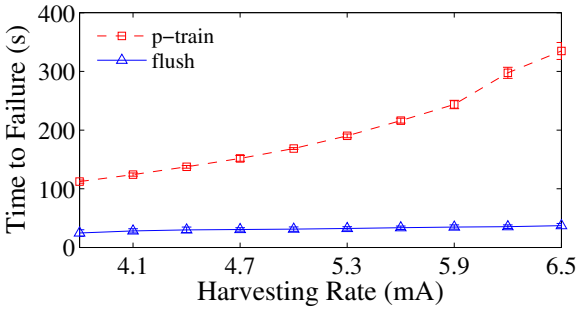
In terms of the mean time to failure, we can observe that Flush performs poorly compared to packet train. Moreover, its performance seems to be almost the same regardless of the harvesting rate. A closer inspection of the results, however, reveal that Flush slightly improves its performance from 24.6 s at 3.8 mA to 37.2 s at 6.5 mA. Whereas, the effect of harvesting rate is significantly noticeable in the case of packet train. In fact, its bulk transfer at 6.5 mA lasts 334.8 s, almost three times that at 3.8 mA which only lasts 112.2 s. The advantage of packet train over Flush can be attributed to its use of duty cycling.

5.3.2. Influence of Hop Count

Figure 14(a) shows the throughput of the Pump-and-Nap, packet train and Flush, as the number of hops between the source and the sink increases



(a) Throughput



(b) Mean Time to Failure

Figure 13: Throughput of PUMP-AND-NAP, packet train and Flush, and mean time to relay node failure of the latter two, of a 5-hop bulk transfer, as a function of energy harvesting rate.

from 1 to 5. Meanwhile, Figure 14(b) plots the mean time to failure of packet train and Flush. Once again, PUMP-AND-NAP is not included in Figure 14(b) because it can sustain the bulk transfer indefinitely. The results simulate 6.5 mA harvesting rate, which is equivalent to the harvesting rate at 100 klux.

The throughput of packet train and Flush are comparable, and both show a decline as the path length increases. This is expected because as the path length increases, the increasing contention due to intra-flow interference causes these transfer schemes to throttle down their respective sending rates. For PUMP-AND-NAP, we observe a slightly different trend. We can see a big drop from 1 to 2 hops, with the latter throughput being just around half of the former. This is expected because for single hop transfers, the source does not need to allocate any duty cycle for reception and that it can allocate its entire duty cycle for transmission. For 2–5 hops, the throughput remains flat because of the effect of controller action to limit the usage of the

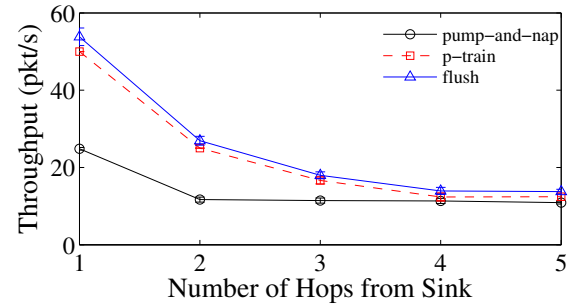
radio. Essentially, the duty cycle enforced is sufficiently low that intra-flow interference is avoided.

With respect to the mean time to failure, Flush shows a flat response regardless of the path length. This is because in Flush, the radios are always on, resulting in roughly the same energy consumption regardless of the path length. As for packet train, we observe an interesting trend where the nodes last longer as the number of hops increases. This curious result is due to the fact that as the number of hops increases, the frequency of packet train transmissions decreases, as evidenced by the decreasing throughput. This leads to the reduction of the amount of time that the radios need to be active. In other words, the duty cycle usage of packet train is highly dependent on the path length, with the duty cycle usage decreasing as the number of hops increases. This indicates the possibility for packet train to attain sustainable data transfer at a certain number of hops. We however remark that such sustainability is achieved *incidentally*, compared with the sustainability provided by Pump-and-Nap that is attained *intentionally* at all hop counts.

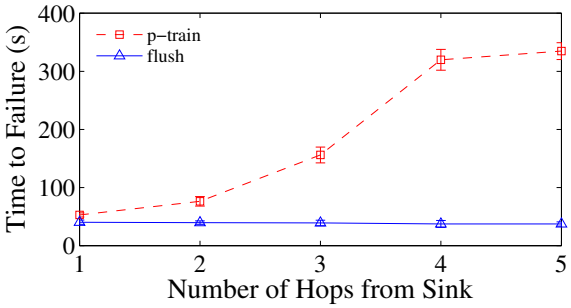
6. Conclusion and Future Work

This work addresses the problem of bulk data transfer in environmentally-powered wireless sensor networks where duty cycle compliance is critical. While several bulk transfer schemes have been proposed in the literature, they focus mainly on maximizing the transfer throughput, neglecting the duty cycle constraints of sensor nodes.

We proposed Pump-and-Nap, a packet train forwarding technique that uses adaptive feedback control to calculate the optimal packet train length for both reception and transmission. The adaptive feedback control aims to control the duty cycle usage of a node, modeled as a linear system with zero-mean disturbance. The latter is mainly due to the uncertainty induced by the pre-transmission overhead in asynchronous wakeup scheduling. We designed a controller that uses prior input-output observations (capacity allocations and their corresponding duty cycle usage) to continuously tune its performance and adapt to wireless link quality variations. Because of its reliance on local information, the controller is amenable to distributed implementation. We implemented Pump-and-Nap in TinyOS and evaluated its performance in real energy harvesting experiments and testbed simulations. Results show that Pump-and-Nap provides high transfer throughput while it simultaneously tracks the target duty cycle. More importantly, en-



(a) Throughput



(b) Mean Time to Failure

Figure 14: Throughput of PUMP-AND-NAP, packet train and Flush, and mean time to relay node failure of the latter two, at harvesting rate of 6.5 mA, as a function of path length.

ergy harvesting experiments show PUMP-AND-NAP is the only scheme that can provide sustainable bulk transfer compared to the other state-of-the-art techniques that we have tested, as the latter greedily maximize throughput at the expense of high and uncontrolled energy consumption.

To the best of our knowledge, this work is the first to consider duty cycle compliance as the primary aim. In the future, we will explore suitable methods to incorporate end-to-end reliability and flow control into PUMP-AND-NAP and study other control mechanisms to provide dynamic duty cycle compliance.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (2002) 393–422.

[2] S. Sudevalayam, P. Kulkarni, Energy harvesting sensor nodes: Survey and implications, IEEE Communications Surveys Tutorials (99) (2010) 1–19.

[3] F. Simjee, P. H. Chou, Efficient charging of supercapacitors for extended lifetime of wireless sensor nodes, IEEE Trans. Power Electronics 23 (3) (2008) 1526–1536.

[4] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, M. Welsh, Deploying a wireless sensor network on an active volcano, IEEE Internet Computing 10 (2) (2006) 18–25.

[5] K. Chebrolu, B. Raman, N. Mishra, P. K. Valiveti, R. Kumar, Brimon: A sensor network system for railway bridge monitoring, in: Proc. ACM MobiSys, 2008, pp. 2–14.

[6] A. Kansal, J. Hsu, S. Zahedi, M. B. Srivastava, Power management in energy harvesting sensor networks, ACM Trans. Emb. Comput. Sys. 6 (2007) 1–38.

[7] C. Vigorito, D. Ganesan, A. Barto, Adaptive control of duty cycling in energy-harvesting wireless sensor networks, in: Proc. IEEE SECON, 2007.

[8] T. Zhu, Z. Zhong, Y. Gu, T. He, Z.-L. Zhang, Leakage-aware energy synchronization for wireless sensor networks, in: Proc. ACM MobiSys, 2009.

[9] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, I. Stoica, Flush: A reliable bulk transport protocol for multihop wireless networks, in: Proc. ACM SenSys, 2007, pp. 351–365.

[10] B. Raman, K. Chebrolu, S. Bijwe, V. Gabale, PIP: A connection-oriented, multi-hop, multi-channel tdma-based mac for high throughput bulk transfer, in: Proc. ACM SenSys, 2010, pp. 15–28.

[11] G. Ekbatanifard, P. Sommer, B. Kusy, V. Iyer, K. Langendoen, Fast-forward: High-throughput dual-radio streaming, in: Proc. IEEE MASS, 2013, pp. 209–213.

[12] S. Duquennoy, F. Österlind, A. Dunkels, Lossy links, low power, high throughput, in: Proc. ACM SenSys, 2011, pp. 12–25.

[13] M. Doddavenkatappa, M. C. Chan, P3: A practical packet pipeline using synchronous transmissions for wireless sensor networks, in: Proc. IEEE IPSN, 2014, pp. 203–214.

[14] A. Varshney, L. Mottola, M. Carlsson, T. Voigt, Directional transmissions and receptions for high-throughput bulk forwarding in wireless sensor networks, in: Proc. ACM SenSys, 2015, pp. 351–364.

[15] J. Hill, R. Szewczyk, A. Woo, P. Levis, K. Whitehouse, J. Polastre, D. Gay, S. Madden, M. Welsh, D. Culler, E. Brewer, Tinyos: An operating system for sensor (2003).

[16] M. Doddavenkatappa, M. Chan, A. Ananda, Indriya: A low-cost, 3d wireless sensor network testbed, in: Proc. TRIDENTCOM, 2011.

[17] W. Ye, J. Heidemann, D. Estrin, An energy-efficient mac protocol for wireless sensor networks, in: Proc. IEEE INFOCOM, Vol. 3, 2002, pp. 1567–1576.

[18] T. van Dam, K. Langendoen, An adaptive energy-efficient mac protocol for wireless sensor networks, in: Proc. ACM SenSys, 2003, pp. 171–180.

[19] T. Zheng, S. Radhakrishnan, V. Sarangan, PMAC: An adaptive energy-efficient mac protocol for wireless sensor networks, in: Proc. IEEE Parallel and Distributed Processing Symposium, 2005.

[20] J. Polastre, J. Hill, D. Culler, Versatile low power media access for wireless sensor networks, in: Proc. ACM SenSys, ACM, New York, NY, USA, 2004, pp. 95–107.

[21] M. Buettner, G. V. Yee, E. Anderson, R. Han, X-MAC: A short preamble mac protocol for duty-cycled wireless sensor networks, in: Proc. ACM SenSys, 2006, pp. 307–320.

[22] Y. Sun, O. Gurewitz, D. B. Johnson, RI-MAC: A receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks, in: Proc. ACM SenSys, 2008, pp. 1–14.

[23] A. Dunkels, L. Mottola, N. Tsiftes, F. Österlind, J. Eriksson, N. Finne, The announcement layer: Beacon coordination for the sensornet stack, in: Proc. EWSN, 2011.

[24] Y. Wu, S. Fahmy, N. Shroff, Optimal sleep/wake scheduling for time-synchronized sensor networks with qos guarantees, IEEE/ACM Trans. Networking 17 (5) (2009) 1508–1521.

[25] A. C. Valera, W.-S. Soh, H.-P. Tan, A survey on wakeup scheduling in environmentally-powered wireless sensor networks, Computer Communications 52 (2014) 21–36.

[26] C. Wang, K. Sohraby, B. Li, M. Daneshmand, Y. Hu, A survey of transport protocols for wireless sensor networks, Network, IEEE 20 (3) (2006) 34–40.

[27] R. Musaloiu-E., C.-J. M. Liang, A. Terzis, Koala: Ultra-low power data retrieval in wireless sensor networks, in: Proc. IEEE IPSN, 2008, pp. 421–432.

[28] G. Goodwin, K. S. Sin, Adaptive Filtering Prediction and Control, Dover Publications, 1984.

[29] M. J. Neely, Stochastic Network Optimization with Application to Communication and Queueing Systems, Morgan & Claypool, 2010.

[30] P. Kumar, P. Varaiya, Stochastic Systems: Estimation, Identification, and Adaptive Control, Prentice-Hall, 1986.

[31] J. Paek, R. Govindan, Rcrt: Rate-controlled reliable transport protocol for wireless sensor networks, ACM Trans. Sensor Networks 7 (3) (2010) 20.

[32] M. Doddavenkatappa, M. C. Chan, B. Leong, Splash: Fast data dissemination with constructive interference in wireless sensor networks, in: Proc. USENIX NSDI, 2013, pp. 269–282.

[33] BQ25504 Battery Management Evaluation Board, [Online]. Available: `http://www.ti.com/tool/bq25504evm-674` [Accessed: Jan 22, 2016].

[34] J. Polastre, R. Szewczyk, D. Culler, Telos: enabling ultra-low power wireless research, in: Proc. IEEE IPSN, 2005.