Manuscript Number: ADHOC-D-11-205R2

Title: On Combining Duty-Cycling with Network Coding in Flood-based Sensor Networks

Article Type: Regular Paper

Keywords: wireless sensor networks, energy efficiency, duty cycling, network coding

Corresponding Author: Dr. Radu Stoleru, Ph.D.

Corresponding Author's Institution: Texas A&M University

First Author: Roja Chandanala, MS

Order of Authors: Roja Chandanala, MS; Wei Zhang, M.S.; Radu Stoleru, Ph.D.; Myounggyu Won

Abstract: Network coding and duty-cycling are two major techniques for saving energy in wireless sensor networks. To the best of our knowledge, the idea to combine these two techniques for even more aggressive energy savings, has not been explored. This is not unusual, since these two techniques achieve energy efficiency through conflicting means, e.g., network coding saves energy by exploiting overhearing (i.e., nodes are awake), whereas duty-cycling saves energy by reducing idle listening (i.e., nodes sleep). In this article, we thoroughly investigate if network coding and duty cycling can be used together for more aggressive energy savings in flood-based wireless sensor networks.

Our main idea is to exploit the redundancy sometimes present in flooding applications that use network coding, and put a node to sleep (i.e., duty cycle) when a redundant transmission takes place (i.e., the node has already received and successfully decoded a sequence of network-coded packets). We propose a scheme, called DutyCode, in which a multiple access control (MAC) protocol implements packet streaming and allows the network coding-aware application to decide when a node can sleep. We also present an algorithm for deciding the optimal coding scheme for a node to further reduce energy consumption by minimizing redundant packet transmissions. Finally, we propose an adaptive switching technique between DutyCode and an existing duty-cycling MAC protocol. We investigate our proposed solutions analytically and implement them on mote hardware. Our performance evaluation results, obtained from a 42-node indoor testbed, show that our scheme saves 30-46% more energy than network coding-based solutions.

On Combining Network Coding with Duty-Cycling in Flood-based Wireless Sensor Networks

Roja Chandanala[a], Wei Zhang[a], Radu Stoleru[a,*], Myounggyu Won[a]

[a]*Department of Computer Science and Engineering, Texas A&M University*

Abstract

Network coding and duty-cycling are two major techniques for saving energy in wireless sensor networks. To the best of our knowledge, the idea to combine these two techniques for even more aggressive energy savings, has not been explored. This is not unusual, since these two techniques achieve energy efficiency through conflicting means, e.g., network coding saves energy by exploiting overhearing (i.e., nodes are awake), whereas duty-cycling saves energy by reducing idle listening (i.e., nodes sleep). In this article, we thoroughly investigate if network coding and duty cycling can be used together for more aggressive energy savings in flood-based wireless sensor networks.

Our main idea is to exploit the redundancy sometimes present in flooding applications that use network coding, and put a node to sleep (i.e., duty cycle) when a redundant transmission takes place (i.e., the node has already received and successfully decoded a sequence of network-coded packets). We propose a scheme, called DutyCode, in which a multiple access control (MAC) protocol implements packet streaming and allows the network coding-aware application to decide when a node can sleep. We also present an algorithm for deciding the optimal coding scheme for a node to further reduce energy consumption by minimizing redundant packet transmissions. Finally, we propose an adaptive switching technique between DutyCode and an existing duty-cycling MAC protocol. We investigate our proposed solutions analytically and implement them on mote hardware. Our performance evaluation results, obtained from a 42-node indoor testbed, show that our scheme saves 30-46% more energy than network coding-based solutions.

Keywords: wireless sensor networks, energy efficiency, duty cycling, network coding

1. Introduction

Energy is a scarce resource in wireless sensor networks (WSN) and its conservation has been the subject of extensive research. While a variety of solutions have been proposed for saving energy in WSN, duty cycling and network coding have proven to be two of the most successful techniques.

Network coding is a technique that increases energy efficiency and reduces network congestion by combining packets destined for distinct users. Since the initial proposal by Ahlswede [2], many applications have incorporated this technique. Network coding is particularly well-suited for WSN due to the broadcast nature of their communications. Overhearing is effortless, propagation is usually symmetric, and energy efficiency is a priority. Network coding can also be found in applications including multi-cast, content distribution, delay tolerant networks, underwater sensing suites, code dissemination, storage, and security. As diverse as these applications are, they all share a common assumption: *nodes in a network are always awake*.

Duty cycling is a technique that increases energy efficiency by allowing *a node to turn off part or all of its systems* for periods of time. Encompassing a range of techniques from peripheral device management to almost complete system shutdown, duty cycling extends node lifetime and reduces maintenance. It has been shown that duty cycling can extend battery life by an order of magnitude or more. In WSN duty cycling is pervasive, and almost all deployed systems use it. Given the importance of duty cycling to WSN, the assumption that nodes will be awake cannot be made. *Since nodes will be asleep at least part of the time, i.e., the*

[*]Corresponding author

[**]A preliminary version of this article was presented at the IEEE International Conference on Networked Sensing Systems (INSS), 2010 [1].

Email addresses: roja@cse.tamu.edu (Roja Chandanala), wzhang@cse.tamu.edu (Wei Zhang), stoleru@cse.tamu.edu (Radu Stoleru), mgwon@cse.tamu.edu (Myounggyu Won)

time available for overhearing is reduced, network coding becomes more difficult.

In this article, we address the challenge faced when aggressive energy savings (i.e., both duty-cycling and network coding) are needed in flooding-based WSN applications. To the best of our knowledge, this is the first work that considers the simultaneous use of duty cycling and network coding. We particularly target code/program dissemination (i.e., distributing a new program/executable image to all sensor nodes), a flood-based application that needs a non-negligible amount of time for execution, e.g., tens to hundreds of minutes for large scale WSN.

Our main idea is derived from the intuition that, due to redundancy in network coding for flooding applications, there are *periods of time* when a node does not benefit from overhearing packets. We seek to precisely determine these periods of time and let nodes that do not benefit from overhearing, to be put to sleep, i.e., to duty-cycle.

Our solution to the aforementioned challenge is *DutyCode*, a cross layer scheme in which Random Low Power Listening (RLPL) – a new MAC protocol – facilitates streaming, elastic random sleeping and transmission arbitration, while the Network Coding-aware Application determines the time to sleep and the sleep duration. We also propose an *Enhanced Coding Scheme (ECS)* algorithm, which eliminates redundant packet transmissions by selecting appropriate network coding schemes for nodes. Finally, a novel technique, called *LPL/RLPL Mode Transition*, ensures the smooth transition between our RLPL protocol and Low Power Listening (LPL), a typical duty-cycling MAC protocol which is more energy efficient for non-flooding WSN applications. The contributions of this article are as follows:

- DutyCode – a cross layer scheme that supports packet streaming and a mechanism for randomizing sleep cycles using elastic intervals. These allow nodes to intelligently select sleep periods.
- ECS – an algorithm for deciding an efficient coding scheme in static networks. ECS assigns coding schemes to minimize the number of transmissions, thus allowing for more energy savings.
- LPL/RLPL Mode Transition - a completely adaptive solution allowing the application to smoothly switch between LPL and RLPL, without packet loss.
- Theoretical analysis of our proposed DutyCode and ECS schemes and extensive simulations demonstrating their energy efficiency and high throughput.
- An implementation of our schemes on mote hardware, and performance evaluation in a 42-node testbed where actual energy consumption is measured.

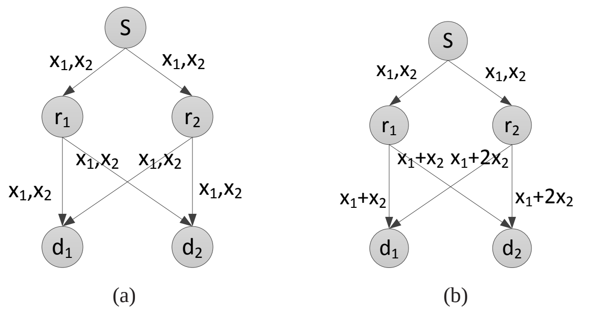


Figure 1: A flood-based application in which node s floods packets x_1 and x_2 in the entire network: a) transmissions when network coding is not used (a total of 6 packet transmissions); b) transmissions when network coding is used (4 packet transmissions).

This article is organized as follows. Section 2 provides background on network coding and duty cycling, and the motivation for our work. Sections 3, 4 and 5 present the design of our DutyCode protocol, ECS algorithm and LPL/RLPL transition technique, respectively. Section 6 presents theoretical analysis of DutyCode and ECS algorithm. Section 7 describes the implementation of our solutions, and Section 8 presents performance evaluation results. We review the state of art in Section 9 and conclude in Section 10.

2. Background and Motivation

Network coding enhances energy efficiency by reducing the number of packet transmissions. The basic concept of network coding, as applied to a flood-based application, can be explained using a simple scenario shown in Figure 1. Sender s wants to flood two packets x_1 and x_2. As shown in Figure 1(a), when network coding is not used, six packet transmissions are required to deliver the two packets to all nodes in the network, i.e., r_1, r_2, d_1, d_2. As shown in Figure 1(b), however, when network coding is used, only 4 transmissions are needed. This is because each of the two relays transmits only one coded packet. For network coding to work, receivers d_1, d_2 must be able to receive both coded packets, i.e. $(x_1 + x_2)$ and $(x_1 + 2x_2)$. Otherwise, they will be unable to decode the other packet received.

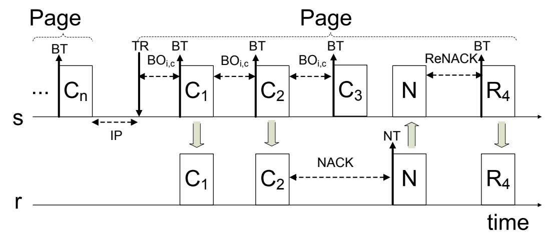


Figure 2: The AdapCode protocol: C$_i$ are coded packets, N is a NACK packet, due to missing packet C$_4$, R$_4$ is the reply to the NACK packet, sent when a ReNACK timer fires, IP is the Inter-Page Interval, BO$_{i,c}$ is the backoff interval - initial and congestion, NACK is a timer.

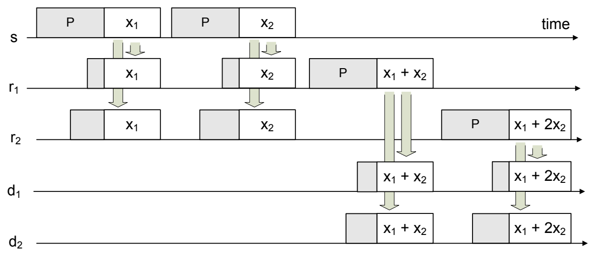


Figure 3: Network coding integrated Low Power Listening based on long preambles (P represents a Preamble), for the example shown in Figure 1(b). Source node s sends two packets x_1 and x_2. Receiver nodes r_1 and r_2 sends a coded packet. Destination nodes d_1 and d_2 are able to decode the coded packets, to retrieve the original packets x_1 and x_2.

It is important to note that, unlike normal broadcast/flooding packets, one missing coded packet can render a sequence of coded packets "useless" (i.e., they do not convey any information). Consider a scenario where a node receives the independent coded packets $(a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4)$, $(b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4)$, and $(c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4)$. For decoding these packets, it becomes critical to receive another coded packet, say $(d_1x_1 + d_2x_2 + d_3x_3 + d_4x_4)$. Otherwise all 3 received packets are useless. As the coding scheme increases (i.e., *coding scheme is defined as the number of different packets coded into a single packet*) the penalty for losing a single packet increases linearly.

2.1. AdapCode Design

In this subsection we present AdapCode [3], a flooding application which uses network coding and employs CSMA as its MAC protocol. Figure 2 describes the protocol. In this figure, a sender node s transmits a sequence of packets (i.e., labeled C_1, ..., C_4, and R_4), which are received by node r. Node r transmits packet N. The transmission and reception of packets are indicated by vertical gray arrows. *As will become apparent later, we depict both the transmission and reception of a packet because these are the time instances when a node needs to be awake.*

We are now ready to describe the AdapCode protocol in detail. When a source node, e.g., a base station, wants to flood a set of packets to all nodes in the network, it broadcasts the data as pages. Each page consists of a number of packets. The arrival of a transmission request at the MAC layer is depicted in Figure 2 as the vertical arrow "TR" (i.e., transmission request). In Figure 2, the TR only for the coded packet C_1 is shown to keep the figure simple. After transmitting a page, the source node waits for a short period of time, called "*Inter-Page Interval*" and labeled IP in Figure 2, for code propagation, and then transmits the next page. After receiving packets, a node adaptively chooses a "*coding scheme*" (i.e. coding scheme is defined above), based on the number of neighbors. If a node does not receive any packets for a fixed period of time (called "*NACK*" delay in Figure 2), it broadcasts a NACK packet (labeled N in Figure 2), which indicates the packets it missed. Upon receiving a NACK, all nodes having the page that contains the requested packets, set a random backoff timer (called "*ReNACK*" delay). The node with the smallest ReNACK delay interval wins and transmits all the requested packets (packet R$_4$ in Figure 2). As with existing CSMA protocols, AdapCode uses a "*Backoff Timer*" (labeled BT in Figure 2) for accessing the medium before transmitting any packet. This backoff timer has two values: an initial value, and a congestion value, selected randomly from BO$_i$ and BO$_c$, respectively.

2.2. Motivation

Most existing duty-cycling protocols achieve energy savings through Low Power Listening (LPL) [4]. However, simply integrating LPL with network coding is not energy efficient since overhearing, the fundamental building block of network coding, is difficult to achieve when nodes aim to sleep as much as possible. Figure 3 shows how network coding and LPL can be employed together in the network topology depicted in Figure 1(b). As shown, because LPL uses long preambles before sending the actual packets, the total transmission time and energy consumption (required for preamble transmission and reception) are expected to increase.

To validate this, we performed experiments in an indoor testbed of 42 Epic motes. We integrated AdapCode with LPL [4], a frequently used duty cycling MAC protocol. In our experiments we varied the LPL Sleep Interval. The Sleep Interval is the time interval that a node is asleep between two consecutive wake-ups - in LPL,

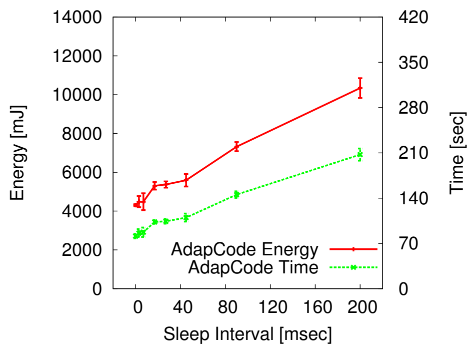


Figure 4: Impact of LPL sleep interval on energy consumption and total execution time (i.e., for flooding a new executable image to all nodes in the network) of AdapCode, an application which uses network coding. Results are obtained from a 42 mote testbed.

nodes wake up briefly, to identify if a preamble is transmitted. If a preamble is detected, the node stays awake to receive the packet transmitted immediately following the preamble. If no preamble is detected, a node goes immediately to sleep. As an example, if a node needs to be awake for 2ms to detect a preamble (this value is dependent on hardware), then a sleep interval SI of 200ms is equivalent to a duty cycle of 1% for the node.

The results of our experiments are depicted in Figure 4. As shown, even very short sleep intervals (i.e., 80msec) nearly doubled the delay and energy consumption. This is in contradiction to the general belief that, for LPL, longer sleep intervals result in higher energy efficiency. From these results, it was clear that: i) a node should select sleep intervals intelligently at non-static intervals (in contrast with LPL which has fixed sleep intervals); and ii) long preamble-based MAC solutions are not suitable for network coding applications.

During our experiments, a significant number of redundant packet transmissions was detected. This redundancy in AdapCode is attributed to its simplified assumption that the network is uniform, where the number of neighbors is assumed to be the same for all nodes. This assumption, however, is not practical and results in inefficient coding scheme assignments. Network topology, e.g., the number of parents or children, should be carefully taken into account when coding schemes are determined to avoid unnecessary packet transmissions. Those considerations motivated us to devise a new technique to assign efficient coding schemes for any network topology such that all nodes can decode all packets with the minimum number of packet transmissions.

It is important to remark that LPL is more energy efficient than DutyCode for non-flooding applications (i.e., when the network traffic is low, there are more opportunities for nodes to sleep). In Section 5 we will present a technique that ensures an adaptive transition between LPL and RLPL, based on network traffic. Such technique also needs to avoid packet loss during the LPL/RLPL transitions.

3. DutyCode Protocol Design

Combining duty-cycling with network coding in flood-based applications is inefficient because nodes use long preambles for communicating with neighbors, and stay awake whenever there is a transmission (i.e., even if the node has received the transmitted packet). Our solution tackles this problem by providing a framework in which nodes are informed of future packet transmissions, without using control packets. This framework allows sensor nodes that run network coded applications to employ timely, smart duty-cycling.

3.1. Main Ideas

Similar to AdapCode, our proposed solution DutyCode groups packets in pages and transmits all packets in a page as a stream using CSMA. The stream sent by a node is *useful* for neighboring nodes lacking the data contained in the page, and *useless* for neighboring nodes that have already received the data in the page. Upon receiving the first packet of a page, a node decides if it should stay awake and receive the remaining packets in the page, or it should sleep while the remaining packets in the page are streamed. The computation of the time to be asleep is based on a control field present in each packet, which indicates the number of remaining packets of the stream.

DutyCode integrates Random Low Power Listening (RLPL), a novel MAC protocol, with the Network Coded Application (NC App). RLPL facilitates *streaming*, *elastic random sleeping* and *transmission arbitration*, while the NC App determines when a node can sleep and for how long (i.e., sleep duration). When requested by the NC App, RLPL turns off the radio for the requested duration if there is no pending transmission. The NC App specifies the sleep duration when it requests the node to sleep. Importantly, RLPL does not put the node to sleep periodically. Unlike other duty-cycling protocols RLPL does not perform Clear Channel Assessment (CCA) before turning off the radio. This is because the CCA would not have any meaning, since requests for sleep come from the NC App when there is, typically, ongoing radio communication (e.g., streaming of useless packets).

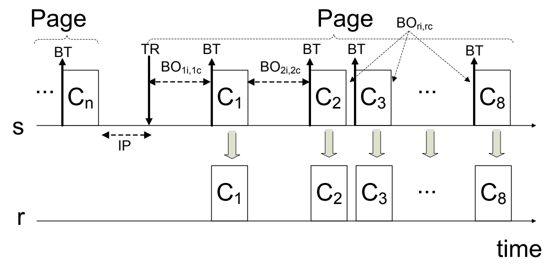


Figure 5: DutyCode streaming: after the backoff intervals $BO_{1i,1c}$, coded packets C_2, through C_8 are streamed, with backoff intervals $BO_{ri,rc}$. BT indicates when a Backoff Timer fired, TR indicates when a Transmit Request has been received by RLPL, and IP represents Inter-Page time Interval. Vertical gray arrows depict the direction of packet transmission.

In the remaining part of this section we describe the main components of RLPL and illustrate DutyCode's execution with an example.

3.2. Packet Streaming

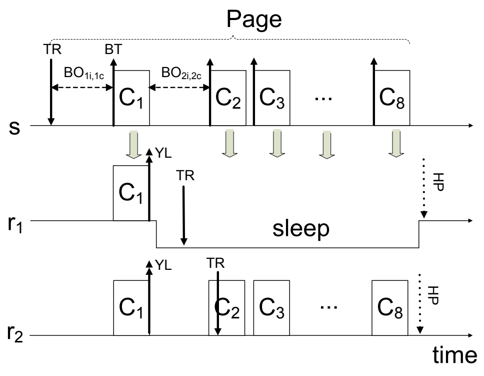
The streaming operation of RLPL is depicted in Figure 5. The meaning of "Page", "IP", "BT" and "C_i" is the same as for AdapCode, shown in Figure 2. As shown, node s receives a request for transmission (i.e., an arrow pointing down, and labeled TR, in Figure 5) and transmits a page (i.e., coded packets C_1 through C_8) to receiver node r. Following the notation used for describing AdapCode, the sender and receiver of a packet transmission can be identified by the gray arrow pointing down and by the BT timer. As shown, the BT timer fires only for node s, the sender of the packets.

When streaming coded packets, the penalty for collision is high. To reduce collisions, in DutyCode, the first two packets of a stream are sent with large random backoff intervals (i.e., $BO_{1i,1c}$ and $BO_{2i,2c}$). The rest of the stream is sent with very small backoff interval $BO_{ri,rc}$. The subscripts "i" and "c" stand for the "initial" and "congestion" backoff, respectively. Thus, BO_{1i} and BO_{ri} are initial backoff intervals, and BO_{1c} and BO_{rc} are congestion backoff intervals. It is important to mention again that the first packet in a page contains the page ID, and that each packet contains a counter indicating how many packets in the stream remain to be sent.

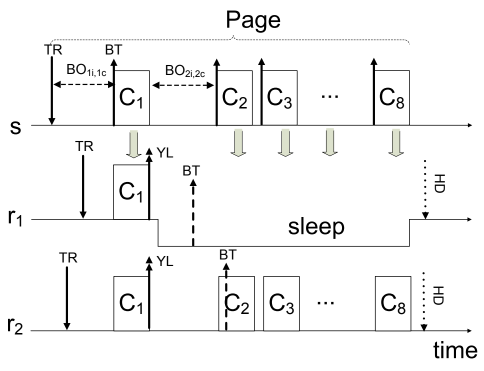
3.3. Elastic Random Sleeping

In this subsection we describe *Yielding*, the main feature of RLPL that enables the elastic random sleeping of nodes, and *transmission defer*. We use Figure 6 to explain these important concepts in RLPL.

Upon receiving the first packet of a stream, a node *yields*. As shown in Figure 6(a) upon receiving the first packet of a stream from node s, nodes r_1 and r_2 decide to yield (i.e., the vertical double arrow labeled YL). While yielding, nodes r_1 and r_2 do not process any transmit requests from the NC App, for the duration of the stream from s. Since node r_1 has the page being transmitted by s, it sleeps for the duration of the stream. This is the elastic random sleeping. In Figure 6(a), the NC App at node r_1 intends to send a packet (i.e., the vertical arrow labeled TR) while the node's radio is turned off (i.e., it sleeps). After r_1 wakes up, it tries to transmit pending packets (i.e., dotted vertical arrow labeled HP - Handle Pending). Because node r_2 does not have the page being transmitted, it stays awake and receives all streamed packets. Similar to node r_1, node r_2 yields (i.e., the vertical double arrow labeled YL in Figure 6(a)). Node r_2

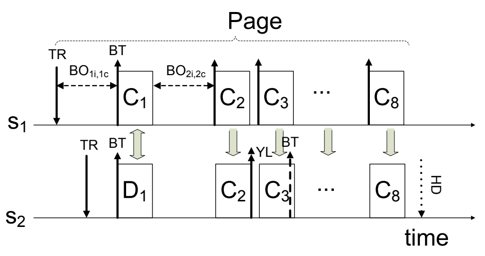


(a)

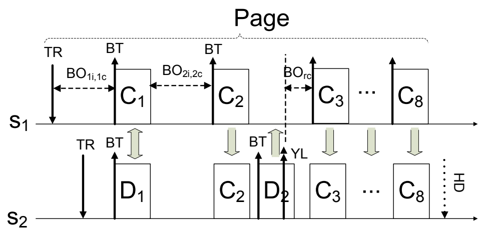


(b)

Figure 6: (a) Yielding and Transmission Defer in DutyCode. (a) The packet transmit requests TR arrive at nodes r_1 and r_2 *after* they yield. This is handled by nodes r_1 and r_2 as a pending request, labeled HP - Handle Pending, after the stream from node s finishes. Since node r_1 has the data in the page, it sleeps; (b) The packet transmit requests TR arrive at nodes r_1 and r_2 *before* they yield. This is handled by nodes r_1 and r_2 as a packet defer, labeled HD - Handle Defer. Similarly with (a) node r_1 sleeps, because it has the data, while node r_2 stays awake to receive the page.



(a)



(b)

Figure 7: Transmission arbitration: ID(s_1) > ID(s_2) and (a) s_2 learns about the stream from s_1 and successfully defers its transmission (i.e., vertical double arrow labeled YL). (b) s_2 learns about the stream from s_1 and fails to defer its transmission (i.e., vertical double arrow labeled YL).

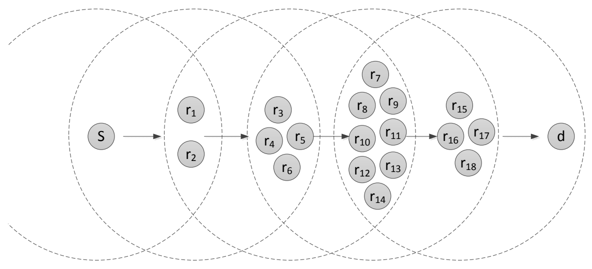


Figure 8: Network topology example to illustrate the operation of DutyCode.

handles its packet transmission request (the receipt of the request is marked as a vertical TR arrow pointing down) after it receives the last stream packet from node s (i.e., vertical dotted arrow labeled HP in Figure 6(a)).

A *transmission defer* is a decision made by RLPL to postpone a scheduled packet transmission for a future time. As shown in Figure 6(b), nodes r_1 and r_2 yield (i.e., the vertical double arrow labeled YL) because of the stream initiated by node s. When they yield, they already have transmission requests from the NC App (to be noted, this scenario is different then the one shown in Figure 6(a) when the transmit request TR reaches RLPL *after* yielding). The backoff timers for these transmission requests fire (i.e., depicted by vertical dotted arrows labeled BT) after the yield operation. When the BT timer fires, the nodes decide to defer their transmissions, until after the stream from node s finishes (i.e., the vertical dotted arrow labeled HD - Handle Defer). Similar with Figure 6(a), node r_1 decides to sleep after yielding (because it has the data in the page), and node r_2 decides to stay awake, to receive the page.

3.4. Transmission Arbitration

Transmission arbitration occurs when two nodes transmit their packets almost simultaneously. More specifically, let's assume that node s_1 receives one packet from node s_2, and that s_1 is awaiting its transmission backoff timer to fire (because s_1 also has packets to send). When s_1 receives the packet from s_2, based on the contents of the packet (e.g., sender ID), s_1 can decide whether it can defer its transmission or not. In our scheme, nodes with smaller node IDs will yield to nodes with higher node IDs.

Figure 7 illustrates two transmission arbitration scenarios where nodes s_1 and s_2 (ID(s_1) > ID(s_2)) compete for a channel. In Figure 7(a), the transmissions of packets C_1 and D_1 are simultaneous, and will result in a collision (represented as the grey double arrow). Because the backoff timers are random, in Figure 7(a) node s_2 learns about the stream from node s_1 first, i.e., packet C_2 is transmitted well before the packet D_2 is attempted to be transmitted by node s_2 (as represented by the dotted vertical arrow labeled BT for node s_2). Since ID(s_1) > ID(s_2), node s_2 decides to yield to node s_1. In this case, the transmission timer of node s_2 (as mentioned, the dotted vertical arrow labeled BT) fires after it makes the defer decision. Therefore it successfully defers its transmission.

Figure 7(b) presents a scenario where the transmission timer of node s_2 fires before it makes the defer decision. As in Figure 7(a) packets C_1 and D_1 collide, but packet D_2 is sent very shortly after packet C_2 is received, without enough time to yield. Consequently, node s_2 cannot prevent packet D_2 from being sent. After D_2, however, node s_2 defers the transmission of its following packets. This scenario shows that node s_2 transmits almost immediately after receiving the packet C_2 from node s_1, which implies that node s_1 has likely received packet D_2 from node s_2. Consequently, node s_1 will need to decide if it should defer its transmission or not. Now, node s_1 will obviously not decide to defer its transmission because ID(s_1) > ID(s_2). Consequently, the remaining packets C_3 through C_8 will be sent successfully.

3.5. *DutyCode Example*

In this subsection we explain how DutyCode works through an example, depicted in Figure 8. As shown, there is a source node s that floods packets in a 5 hop WSN. We assume that each page consists of 8 packets. For ease of explanation, we assume that 2 nodes are 1 hop away from s, 4 nodes are 2 hops away from s, 8 nodes are 3 hops away from s and lastly, 4 nodes are 4 hops away from s. Using the AdapCode algorithm to decide the coding scheme (which is based on the number of neighbors), nodes r_1 and r_2 have $cs = 2$. This means that the number of packets each node sends is $pp/cs = 4$, if the number of packets per page pp is 8. Similarly, for nodes 2 hops away from s the coding scheme cs is 4 (i.e., each node will send $8/4 = 2$ coded packets), and for nodes 3 hops away from s the coding scheme cs is 8 (i.e., each node sends $8/8 = 1$ coded packet). After the nodes decide their coding schemes (again, the technique we use here is the same as of AdapCode's, based on the number of neighbors), the flooding operation can start.

The sender s sends the packets in a page continuously (i.e., as a stream). When sender s finishes streaming, it waits for a period of time for potential NACK packets (which indicate that receivers missed some packet(s)). In response to a NACK packet, sender s resends the missing packets. After nodes r_1 and r_2 receive all 8 packets in the page, using their coding scheme, they code and send 4 packets each. Assuming the backoff timer of node r_1 fires first, the streaming of the 4 packets from it will start. After receiving the first packet (of the 4), node r_2 realizes that it has already seen the page, and it will decide to sleep for the remaining time of the stream (in this example 3 packets). Similarly, when node r_2 streams its 4 coded packets, node r_1 will sleep. When nodes r_1 and r_2 finish sending their coded packets, they will wait for NACK packets. NACK packets might be sent by nodes r_3 through r_6 if they missed a coded packet, and could not decode all packets received. If a NACK packet is received, either r_1 or r_2 resends the missing packet. If nodes r_3 through r_6 are able to decode the 8 packets received, then they have the entire page. Based on their coding scheme (which is $cs = 4$), each node will send 2 coded packets. Whenever one of nodes streams its packets (let's assume r_3), the nodes that have received already the page (i.e., nodes r_1, r_2, r_4, r_5 and r_6) decide to sleep.

4. An Enhanced Coding Scheme (ECS)

While our proposed solution, presented in the previous section, saves a considerable amount of energy there is still an opportunity to save more energy by reducing the number of unnecessary packet transmissions. This is primarily because, thus far, we used the same algorithm as AdapCode, for deciding the coding scheme each node uses. The simple algorithm used by AdapCode uses the number of neighbors. The algorithm, at high node densities, increases the coding scheme. Hence, each node sends fewer coded packets. This scheme, however, does not identify unnecessary packet transmissions, which is the purpose of our Enhanced Coding Scheme (ECS) algorithm.

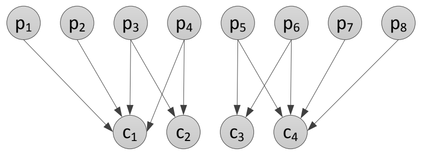


Figure 9: An example of a network topology that causes redundant packet transmissions.

In order to analyze the unnecessary transmissions, we define the *Preferred Coding Scheme (PCS)* for each node. *PCS is defined as the maximum coding scheme that a node's parents can use such that all parent's children can successfully decode all encoded packets.*

Consider Figure 9 which shows a hypothetical network with four leaf nodes ($c_1 - c_4$) and their parents ($p_1 - p_8$), where c_1 and c_4 receive packets from four parents whereas c_2 and c_3 receive packets from only two parents. In our example, consider the PCS for c_1. Since c_1 receives 4 coded packets (i.e., each from parent p_1 through p_4) at most 4 packets can be encoded into a single packet, yielding a PCS of 4 for c_1. Similarly, the PCS for c_2, c_3, and c_4 are 2, 2, and 4, respectively. Since the PCS for c_2 and c_3 is 2, the coding scheme of $p_3 - p_6$ must be 2. Consequently, all children $c_1 - c_4$ can decode all packets from the transmissions from $p_3 - p_6$. The result, and this is how ECS removes unnecessary transmissions, is that transmissions from p_1, p_2, p_7, and p_8 are all useless.

In order to avoid such extraneous transmissions and save energy, we propose an Enhanced Coding Scheme (ECS) Algorithm, which decides the optimal coding scheme of each node. Our algorithm is centralized and uses information about the network topology. We represent the network as a directed graph $G = (N, E)$, where N is the set of all nodes n_i, and E is the set of edges (n_i, n_j) such that n_i is the one-hop parent of n_j. Each edge has a Link Quality (LQ) scalar value, which represents the link's successful packet delivery ratio. Only edges with LQ greater than a pre-defined Link Quality Threshold (LQT) are considered. Different LQTs re-

Algorithm 1 Enhanced Coding Scheme (ECS)

```
1:  for each $n_i \in N$ do
2:      $n_{i,j}^{PCS} \leftarrow |P_i|, \forall n_j \in P_i$.
3:  end for
4:  for each $n_i \in N$ do
5:      $n_i^{CS} \leftarrow min\{n_{i,j}^{PCS} \mid n_j \in C_i\}$
6:  end for
7:  for each $n_i \in N$ do
8:      $n_i^{EQNS} \leftarrow 0$
9:      for each $n_j \in P_i$ (in an ascending order of $n_j^{CS}$) do
10:         if $n_i^{EQNS} \leq$ page_size then
11:             $n_i^{EQNS} \leftarrow n_i^{EQNS} + \frac{\text{page\_size}}{n_j^{CS}}$
12:         else
13:             $n_{i,j}^{PCS} \leftarrow$ null_coding_scheme
14:         end if
15:     end for
16: end for
17: for each $n_i \in N$ do
18:     if $\forall n_j \in C_i, n_{j,i}^{PCS}$ = null_coding_scheme then
19:         $n_i^{CS} \leftarrow$ null_coding_scheme
20:     else
21:         $n_i^{CS} \leftarrow min\{n_{j,i}^{PCS} \mid n_j \in C_i\}$
22:     end if
23: end for
```

sult in different network topologies, thereby affecting the performance of ECS. Let $P_i = \{n_j \mid (n_j, n_i) \in E\}$ be the set of all one-hop parents of node n_i and $C_i = \{n_j \mid (n_i, n_j) \in E\}$ be the set of all one-hop children of node n_i. We denote by $n_{i,j}^{PCS}$ the PCS of n_i for its parent $n_j \in P_i$, and by n_i^{CS} the coding scheme of n_i.

The pseudocode for the proposed ECS algorithm is shown in Algorithm 1. ECS runs in 2 phases. In the first phase, the algorithm computes the PCS value $n_{i,j}^{PCS}$ for all $n_j \in P_i$. (Line 1-3).

Then, the initial coding scheme for each node n_i is decided as the $min\{n_{j,i}^{PCS} : n_j \in C_i\}$, i.e., the minimum PCS value among all PCS values of its children (Line 4-6). If $|C_i| = 0$, then the "null coding scheme" is chosen for n_i (in a "null coding scheme" no coded packets are forwarded), preventing a leaf node from sending unnecessary packets. In the second phase, the algorithm checks for possible redundant transmissions for each node n_i, by examining the initial coding schemes of its parents. If for node n_i there is a redundant transmission from one of its parent n_j, the algorithm updates the PCS value for the parent, $n_{i,j}^{PCS}$ to the null coding scheme (Line 7-16). In the last step, the algorithm checks the PCS value for the children of each node. If all children suggest a null coding scheme, then the algorithm sets the null coding scheme as the final one for the parent; otherwise the coding scheme assigned to the parent is the minimum PCS value among its children (Line 17-23).

Although ECS is a centralized algorithm where coding schemes are decided at a central entity, it can be modified to run in a distributed manner through additional message exchanges between nodes. We leave the development of a distributed ECS algorithm as future work.

5. LPL/RPLP MAC Transitioning

Aggressive energy saving can be achieved by DutyCode in flood-based WSN (i.e., high network traffic). For low network traffic, however, LPL is more energy efficient. Thus, there is a need for an efficient technique to switch between RLPL and LPL. Such switching technique needs to be carefully designed to ensure a smooth and timely transition with no packet loss.

In our solution, each node starts executing LPL. When a node receives the first packet of the flood, it attempts to switch to RLPL. To minimize packet loss during the transition from LPL to RLPL, we use a transient mode, called NoSleepLPL. In NoSleepLPL, after receiving the first packet of the flood, a node does not sleep. This is because the node tries to avoid missing packets. The received packets are relayed utilizing long preambles (as LPL does), to ensure that node's children, in turn, do not miss any packets. Similarly, the children nodes transition to NoSleepRLPL successfully. At a fixed time interval after receiving the first packet of the flood, a node switches from NoSleepLPL to RLPL. The node switches back from RLPL to LPL mode when it has received all packets in the flood. Switching back from RLPL to LPL is relatively easier due to low traffic, thus not incurring packet loss.

6. DutyCode Protocol and ECS Algorithm Analysis

In this section we derive performance bounds for the DutyCode protocol and ECS algorithm. The aim of our analysis is to:

- Show that DutyCode does not have any overhead, when compared with AdapCode. The two metrics we investigate are *the total number of packets per page* transmitted (Section 6.1), and *the total execution time* of flooding (Section 6.2).

- Show that the coding schemes assigned by ECS are optimal (Section 6.3).
- Compute an upper bound on the energy savings of DutyCode (Section 6.4).

For our analysis we use the following notations: pp is number of packets per page; cs is the coding scheme; cp is the collision probability in AdapCode; cp_1 and cp_2 are the collision probabilities associated with BO_1 and BO_2 backoff intervals in DutyCode, respectively; BO_c is the CSMA congestion backoff interval (as it will be noted below BO_i is not of interest for AdapCode); and t_{tr} is the transmission time per packet.

We assume that the sleep interval per page SI is chosen such that there is no time overhead due to sleeping (i.e., stream duration is much longer than the sleep interval):

$$SI \leq (BO_{1i}/2 + pp/cs \cdot t_{tr}) \tag{1}$$

For the analysis in this section it is paramount to observe the following: i) because AdapCode is message intense, there is always a node waiting to transmit a packet. *Consequently, the congestion backoff interval for AdapCode will be chosen randomly between 0 and BO_c*. Because the backoff is uniformly distributed, the average backoff interval is $BO_c/2$ and the average collision probability is cp; ii) in DutyCode the first packet of a stream is always transmitted with a backoff interval randomly chosen between 0 and BO_{1i}. Hence, the average backoff interval for the first packet is $BO_{1i}/2$. Since the average collision probability for the first packet of the stream is cp_1, the collision probability for the second packet of the stream is $cp_1 \cdot cp_2$ (i.e., a collision will occur during the transmission of second packet if and only if there was a collision during the first packet transmission).

6.1. *Total Number of Packets per Page Transmitted*

In this section we investigate the relationship between P_p^d and P_p^a, the total number of packets per page *transmitted* by DutyCode and AdapCode, respectively (we note here that P_p^d and P_p^a are different than pp, which is a constant describing how large a page is). Our result is expressed by the following theorem.

Theorem 6.1. *If $BO_{1i} \geq BO_c$ then $P_p^d \leq P_p^a$.*

Proof. In both DutyCode and AdapCode, three types of packets contribute to the total number of packets per page: a) coded packets; b) NACK packets; and c) Re-NACK packets. We now derive the number of packets of each type.

6.1.1. *Coded Packets*

Coded packets are packets transmitted by a node as a result of network coding. The number of coded packets per page is $C_p = pp/cs$. Assuming the coding scheme is identical for DutyCode and AdapCode (i.e., we do not consider here the Enhanced Coding Scheme (ECS)), we have:

$$C_p^a = C_p^d \tag{2}$$

where C_p^a and C_p^d are the number of coded packets for AdapCode and DutyCode, respectively.

6.1.2. *NACK Packets*

A node sends a NACK packet when it is unable to decode a page. This can happen because of two factors: i) the node does not receive enough independent coded packets needed for decoding a page; and ii) collisions cause loss of coded packets.

Since DutyCode uses the same coding scheme as AdapCode, the first factor, i.e., not enough independent coded packets received, has no impact on the total number of packets. Hence, we do not consider it.

We now analyze how collisions impact the number of NACK packets in AdapCode and DutyCode.

In AdapCode, the maximum number of NACKs per packet is $(cp + 2cp^2 + ...)$. This is because with collision probability cp a coded packet will be lost, hence 1 NACK will need to be sent. Additionally, if the transmission of the NACK packet causes a collision with a regular packet (this will occur with probability cp^2) then two NACK packets will need to be sent. Consequently, the total number of NACKs per page for AdapCode is:

$$N_p^a = (pp/cs)(cp + 2cp^2 + ...). \tag{3}$$

For DutyCode, NACK packets can be sent as a result of: i) collision of the first packet transmission, which occurs with probability (cp_1); ii) collision of the second packet transmission, which occurs with probability $(cp_1 \cdot cp_2)$. Thus, the total number of packets per page for DutyCode is:

$$\begin{aligned} N_p^d &= (cp_1 + cp_1 \cdot cp_2) + 2cp_1 \cdot (cp_1 + \\ &\quad cp_1 \cdot cp_2) + 3cp_1^2(cp_1 + cp_1 \cdot cp_2) + ... \\ &= (1 + cp_2) \cdot (cp_1 + 2cp_1^2 + ...). \end{aligned} \tag{4}$$

One can remark that collision probabilities are inversely proportional with backoff intervals. Thus, we can express $cp = k/BO_c$ and $cp_1 = k/BO_{1i}$, where k is a proportionality constant (assumed the same for DutyCode and AdapCode). Consequently, the ratio of collision probabilities cp and cp_1 is:

$$\frac{cp_1}{cp} = \frac{BO_c}{BO_{1i}}.$$

Consequently, if backoff intervals are chosen such that $BO_{1i} \geq BO_c$, then:

$$cp_1 \leq cp. \tag{5}$$

Considering Equation 5, if $pp/cs \geq (1 + cp_2)$ then the relation between N_p^d and N_p^a (given by Equation 3 and Equation 4) is:

$$N_p^d \leq N_p^a. \tag{6}$$

It is important to reemphasize the required condition $pp/cs \geq (1 + cp_2)$. This condition basically says that pp needs to be different than cs. If this condition is not true, i.e., $pp = cs$, then there is only one packet in the stream, and there is no opportunity for sleeping.

6.1.3. *ReNACK Packets*

ReNACK packets are uncoded packets, sent in response to NACK requests. We now analyze the number of ReNACK packets for AdapCode and DutyCode.

In AdapCode, if there is a collision while transmitting a coded packet (i.e., with probability of collision cp) all packets in the page need to be resent, without being coded. Consequently, for each coded packet (and there are pp/cs coded packets in each page), the node needs to retransmit cs packets. Similar to scenario described for NACKs, the node needs to transmit these packets twice with probability cp^2. Hence, the number of ReNACKs per page for AdapCode is:

$$R_p^a = cs \cdot pp/cs \cdot (cp + 2cp^2 + ...). \tag{7}$$

For DutyCode, a collision during a stream transmission can occur: i) during the transmission of the first packet (with probability cp_1). In this scenario cs packets will be retransmitted; and ii) during the transmission of the second packet (with probability $cp_1 \cdot cp_2$). In this scenario the entire page will be retransmitted. Consequently, the number of ReNACK packets sent by DutyCode is $R_p^d = cs \cdot cp_1 + cp_2 \cdot cp_1 \cdot pp$. Assuming the worst case, in which pp packets need to be retransmitted in case of collision during the transmission of first packet, the total number of ReNACKs per page per node is:

$$\begin{aligned} R_p^d &= cp_1 \cdot pp + 2cp_1^2 \cdot pp + ... \\ &= pp \cdot (cp_1 + 2cp_1^2 + ...). \end{aligned} \tag{8}$$

From Equations 7, 8 and 5 the relation between the number of ReNACK packets sent by AdapCode and DutyCode is:

$$R_p^d \leq R_p^a. \tag{9}$$

From Equations 2, 6 and 9, then $P_p^d \leq P_p^a$, where $P_p^a = C_p^a + N_p^a + R_p^a$ is the total number of packets per page of AdapCode, and $P_p^d = C_p^d + N_p^d + R_p^d$ is the total number of packets per page of DutyCode. □

If we denote by s the total number of streams in the network, the total number of packets per node for AdapCode can be written, in terms of s, as:

$$P^a = s \cdot pp/cs + N_p^a \tag{10}$$

where N_p^a is the number of NACK packets for AdapCode.

In DutyCode, coded packets and ReNACKs (note: not NACK packets) are transmitted as streams. Consequently, the total number of packets transmitted per node is:

$$P^d = s \cdot pp/cs + N_p^d \tag{11}$$

where the first term represents the number of coded and ReNACK packets a node transmits, and N_p^d is the number of NACK packets for DutyCode.

6.2. *Total Execution Time*

In this subsection we investigate the relation between T_n^d and T_n^a, the total execution time of flooding, per node, for DutyCode and AdapCode, respectively. Our analysis omits the delay that a packet defer causes. Our result is expressed by the following theorem.

Theorem 6.2. *If $pp/cs \cdot BO_c \geq BO_{1i}$ and $BO_c = BO_{1c}$, then $T_n^d \leq T_n^a$.*

Proof. As mentioned earlier, the average backoff time per packet in AdapCode is $BO_c/2$. Hence, the total execution time, per node, for the flooding operation is:

$$T_n^a = P^a(BO_c/2 + t_{tr})$$

which, after the substitution of P^a with Equation 10, becomes:

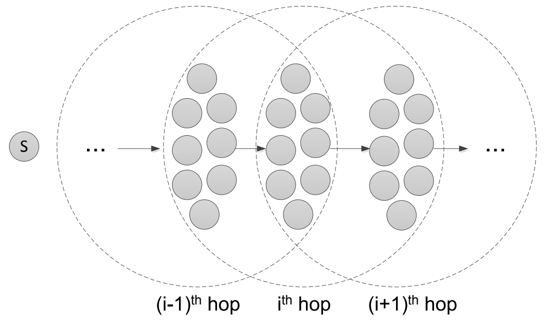


Figure 10: A multihop network topology for a flood-based WSN application. Large dotted circles represent communication range. Groups of nodes are $(i-1)$, i and $(i+1)$ hops away from s.

$$T_n^a = s \cdot pp/cs \cdot BO_c/2 + N_p^a \cdot BO_c/2 + P^a \cdot t_{tr} \quad (12)$$

For DutyCode, the average backoff interval for a stream is $BO_{1i}/2$ except for the stream which is transmitted after a NACK packet (for a NACK packet, yielding is not done because it is not a stream). Hence, the wait time of the stream transmitted right after the NACK is $BO_{1c}/2$. In DutyCode, based on Equation 1, the total time for flooding is the total time required for all packet transmissions. Consequently the total time per node is:

$$\begin{aligned} T_n^d &= (s - N_p^d) \cdot BO_{1i}/2 + N_p^d \cdot BO_{1c}/2 + \\ &\quad + N_p^d \cdot BO_{1i}/2 + P^d \cdot t_{tr} \\ &= s \cdot BO_{1i}/2 + N_p^d \cdot BO_c/2 + P^d \cdot t_{tr} \end{aligned}$$

since $BO_{1c} = BO_c$.

From Equation 6 and Theorem 6.1, when $pp/cs \cdot BO_c \geq BO_{1i}$, then:

$$T_n^d \leq T_n^a.$$

□

6.3. Total Energy Saving

This section presents an analytical upper bound on the total energy saving of a node in DutyCode. To prove the upper bound, we consider a particular network topology depicted in Figure 10, where nodes in i^{th} hop are within the interference range of nodes in $(i+1)^{th}$ hop and $(i-1)^{th}$ hop. We assume that the Inter-Page interval (IP) is minimal, i.e., the source node sends one page after the previous page has been successfully flooded by the nodes that are 3 hops away from it, thus preventing the hidden terminal problem. We denote by S_i the set of nodes that are i hops away from the source. The total energy saving for a node is defined as follows:

$$E_{save} = T_{sleep}/T_{total} \quad (13)$$

where T_{total} is the total flooding time, and T_{sleep} is the total time that a node is in sleep mode.

Since flooding is a pipelined process, given the minimal IP interval, T_{total} can be estimated as the total time taken for any node in S_{i-1}, S_i, or S_{i+1} to finish the flood. Consider a node $n_i \in S_i$. In this topology, n_i cannot transmit a packet if any node in S_{i-1} (parents), S_i (peers), or S_{i+1} (children) transmits. Thus, the expected total flooding time $E(T_{total})$ is given by:

$$\begin{aligned} E(T_{total}) &= \left(\frac{|S_{i-1}| + |S_i - 1| + |S_{i+1}|}{2}\right) \cdot T_{page} \cdot P \\ &= \frac{3n}{2} \cdot T_{page} \cdot P \end{aligned} \quad (14)$$

where T_{page} is the time taken for one page, i.e., $T_{page} = (BO_{1i}/2 + pp/cs \cdot t_{tr})$, and P is the total number of pages. The first term is divided by 2, to account for senders and receivers. Essentially, at most $\frac{|S_{i-1}|+|S_i-1|+|S_{i+1}|}{2}$ transmissions take place.

If we assume a coding scheme with no redundant transmissions, only packets transmitted by the nodes in S_{i-1} are useful. Consequently, n_i can sleep while the nodes in S_i and S_{i+1} are transmitting their packets. The expected total sleep time $E(T_{sleep})$ is thus:

$$\begin{aligned} E(T_{sleep}) &= \left(\frac{|S_i - 1| + |S_{i+1}|}{2}\right) \cdot T'_{page} \cdot P \\ &= n \cdot T'_{page} \cdot P \end{aligned} \quad (15)$$

where T'_{page} is the sleep time per page, i.e., $T'_{page} = SI$.

Considering Equations 1, 14 and 15, the expected total saving in energy is:

$$\begin{aligned} E(E_{save}) &= E\left(\frac{T_{sleep}}{T_{total}}\right) \\ &= \frac{n \cdot SI}{\frac{3n}{2}(BO_{1i}/2 + pp/cs \cdot t_{tr})} \leq \frac{2}{3}. \end{aligned} \quad (16)$$

6.4. Enhanced Coding Scheme Analysis

In this section we prove the optimality of the coding scheme assigned by our proposed ECS algorithm.

Theorem 6.3. *ECS produces the optimal coding schemes for all nodes, such that all packets can be decoded successfully.*

Algorithm 2 Streaming a Packet Received

```
1:  if (# of remaining pkts > 0) then
2:      if (any yield cond. is true) then
3:          if (# pkts awaiting transmit > 0) then
4:              attempt to DEFER pkt transmission
5:              RLPL saves DEFER result
6:          end if
7:          RLPL starts NoSend timer
8:      end if
9:  else
10:     RLPL stops NoSend timer and handles packet
11: end if
```

Proof. Assume by contradiction that there exists a better coding scheme *BCS* . This implies that in *BCS* , there exists a node that transmits fewer packets than ECS. Let this node be n_i. Note that, in ECS, each parent chooses the minimum coding scheme among those proposed by its children (as explained in Algorithm 1 (Line 21)). Thus, if n_i transmits fewer packets, then at least one child would not be able to decode all packets successfully, which is a contradiction. Consequently, *BCS* does not exist. □

7. Implementation

We implemented DutyCode and LPL/RLPL Transition in nesC for TinyOS 2.1. We modified several existing TinyOS modules: CC2420ReceiveC, CC2420TransmitC and CC2420CsmaC - from now on, we will refer to these modules as `Receive`, `Transmit` and `Csma`, respectively. We also added DutyCode specific modules: `RandomLPL` and `RPowerCycleC`. The ECS algorithm was implemented in Java, on a central server. It uses network topology information collected from the nodes to decide the coding scheme for each node.

7.1. Packet Streaming

We modified the `Transmit` and `Receive` modules and implemented the `RandomLPL` module for packet streaming. When streaming is achieved, an application sends packets one after another, without any significant delay (i.e., as soon as `sendDone()` event is signaled). Each packet header contains the number of remaining packets in the stream, computed based on the coding scheme used.

The `Receive` module notifies the `Transmit` module when the node receives a stream packet from another node. Upon receiving a signal from `Receive`, `Transmit` runs Algorithm 2. First, `Transmit` checks if it has pending packets or the end of the stream has been reached (Line 1). If there are remaining packets in the stream, `Transmit` checks if the stream satisfies the yielding conditions discussed in Section 3 (Line 2). If the stream does not satisfy the yielding conditions, no action is taken. Otherwise, if it is in the middle of transmission, the node tries to defer the packet transmission and it informs the `RandomLPL` module about two things: i) details of the stream transmission; and ii) the defer status, if there was a need for packet defer (Line 4-5). The `RandomLPL` module then sets a NoSend timer (Line 7) and keeps future transmit requests pending until the stream duration is over or it is informed by `Transmit` that the stream it is yielding to, has completed. `Transmit` informs `RandomLPL` about the completion of stream upon receiving a signal from `Receive` about the last packet of the stream (Line 10). We implemented the packet defer in the `Transmit` module because it is the only module that maintains the transmission internal state.

Upon receiving a transmit request from the application with the result of clear channel assessment (CCA), the `Transmit` module copies the message to the radio chip and waits for the backoff interval corresponding to the CCA result. `Transmit` can defer a packet transmission until the actual transmission has been started. The application would not be aware about this packet defer and RLPL handles the deferred packet as soon as possible.

7.2. Elastic and Random Sleeping

We implemented/modified the `RandomLPL` and `RPowerCycleC` modules so that sleep requests are no longer handled periodically, as done for LPL. Instead, the requests to put the radio in low power mode (i.e., requests to sleep) are treated as one time requests. The `RPowerCycleC` module is also modified not to perform clear channel assessment (CCA) before turning off the radio. This is because the network coding application decides to put a node to sleep, based on the knowledge it has about the currently transmitted stream. This stream is useless to the node. Hence, performing CCA is unnecessary - we know that a transmission is ongoing, but it is useless. We also modified the `Csma` module to turn off the radio only if `Transmit` decides to defer the transmission.

7.3. The ECS Algorithm

We implemented the ECS algorithm in JAVA. We execute the algorithm on a central server, after we collect connectivity information from the entire network.

We leave the development of a distributed ECS algorithm as future work. When the algorithm starts, the network topology is constructed by receiving neighbor tables from the motes, via serial ports. Based on the network topology, ECS computes the optimal coding scheme for each node as described in Section 4. The new coding schemes are transmitted back to the nodes via serial ports. When the flooding starts, each node uses the new coding scheme.

7.4. LPL to RLPL Mode Transition

In order to achieve a smooth transition between LPL and RLPL (as described in Section 5), we built two independent modules, one containing the LPL protocol, and the other RLPL. We also created a wrapper module that provides a common interface both MAC protocols. The wrapper handles the LPL to RLPL mode transition smoothly: if the current MAC protocol is in the middle of transmission, the transition happens after receiving the `sendDone()` event. Switching is accomplished by stopping the current MAC protocol and then starting the other MAC protocol. Especially for the transition from LPL to RLPL, the transient state noSleepLPL is introduced to minimize the packet loss, as explained in Section 5. The noSleepLPL is implemented such that the `DefaultLPL` and `RPowerCycleC` modules do not turn off the radio when requested by an application through the wrapper module.

8. Performance Evaluation

We evaluated the performance of DutyCode, ECS algorithm and LPL/RLPL mode transition in a testbed consisting of 42 Epic motes [5] deployed in an indoor environment of approximately 500ft^2. Among the 42 nodes, 14 are instrumented for power consumption measurements. The map of our testbed is shown in Figure 11. The testbed has a diameter of at most 5 hops. We varied the network density (and implicitly the diameter of the network) by changing the radio TX Power (i.e., the TXCTRL.PA_LEVEL register of the CC2420 transceiver [6]). The TX Power can be varied from 2 to 31, the lowest and the highest transmit power levels, respectively. Each experimental point represents the mean of 5 executions of the protocol. Standard deviation is depicted in all performance evaluation results.

As state of art for our performance evaluation we chose AdapCode [3], a flooding protocol that uses network coding. The metrics we used for performance evaluation are the *per node energy consumption* and

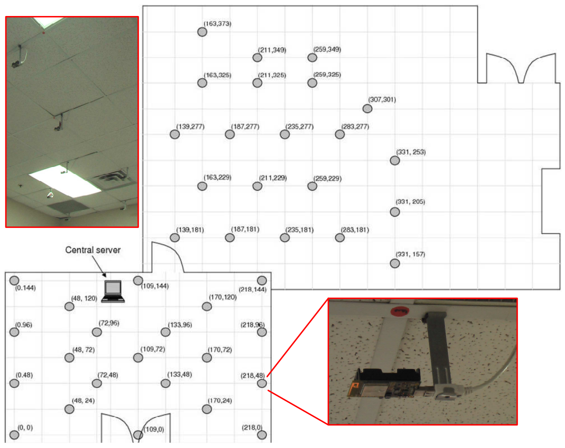


Figure 11: The map of our testbed. For each node, (x, y) represents its relative coordinate with respect to the origin (0,0), in the lower left corner. The origin is where the source node, initiating the flood operation, was located. The network is at most 5 hops (when the radio transmit power is the lowest). The central server is where measurements for power consumption, neighborhood are collected.

total flooding time. While we are interested in energy consumption, we also aim not to increase the total flooding time. The parameters that we vary are the sleep interval (SI), node density (ND), the size of packet (SP), the number of packets (NP), the length of the NACK timer, i.e., NACK delay ($NACKD$), and the Inter-Page Interval (II). From Theorems 6.1 and 6.2, the BO$_{1i}$ should satisfy the condition: $pp/cs \cdot BO_c \geq BO_{1i} \geq BO_c$. In order to decrease the collision probability of DutyCode and reduce the penalty for retransmission, the greater bound for BO$_{1i}$ is used for the experiments. In our experiments, the default back-off intervals were chosen as follows: BO$_{1i}$ and BO$_{2i}$ are chosen randomly in the interval [0.3msec, 8msec], BO$_{1c}$ and BO$_{2c}$ are chosen randomly in the interval [0.3msec, 2msec], BO$_{ri}$=(2+node_id)/32msec, and BO$_{rc}$=(5+ node_id)/32msec.

The default values for the parameters are: SI = 17msec, ND = 4, SP = 28bytes, NP = 256, $NACKD$ = 640msec, II = 300msec. The effects of these parameters on the performance of DutyCode are investigated in the remaining part of this section.

8.1. Preliminary Evaluation

We verified the correct operation of DutyCode protocol using three regular nodes and a source node, forming a single hop network. An oscilloscope was used to measure the actual power consumption and sleep intervals. Figure 12 depicts the oscilloscope view of coded packet

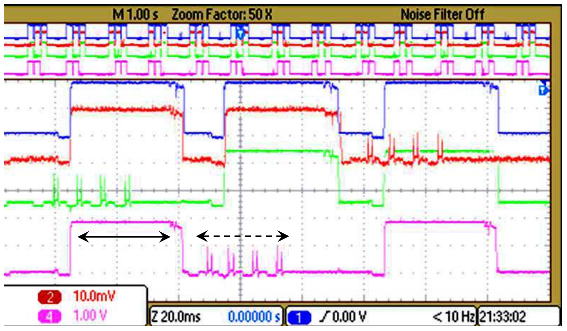


Figure 12: Packet streaming captured on oscilloscope shows the current consumption during sleeping and during packet transmission. The current consumptions increases from top to bottom. The sleeping interval is indicated by a solid arrow and the packet streaming is denoted by a dotted arrow (i.e., sleeping interval is low current consumption, packet streaming requires high current consumption).

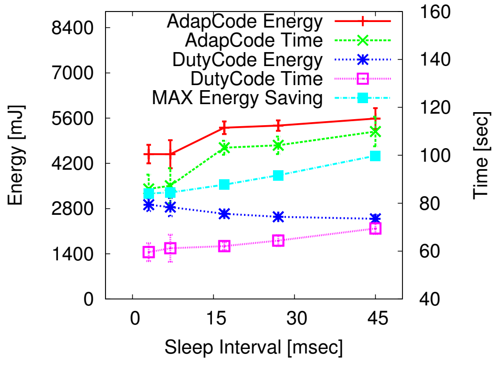


Figure 13: The effect of the sleep interval SI on energy consumption and execution time of flooding. The bound on the energy efficiency of DutyCode is also plotted.

transmissions of the 3 nodes after receiving a page from the source. As shown, two small spikes under the dotted arrow indicate a packet transmission. A cluster of such spikes represents a "stream". In our experiment, a page consisted of 4 packets, hence the 4 sets of spikes. The solid arrow indicates the sleep duration of a node. As the figure shows, during a stream transmission, other nodes are in the sleep state. As soon as the transmission is finished, one of the remaining nodes starts its stream transmission, while the remaining nodes sleep (since they already have the page).

8.2. *Sleep Interval*

In this experiment we investigate how sleep interval SI affects energy consumption and total flooding time. It is important to remark that AdapCode and DutyCode should consume the same amount of energy if SI is 0msec. This is because both AdapCode and DutyCode use the same network coding scheme, and at SI = 0msec, neither scheme allows nodes to sleep.

Before presenting the experimental results, we would like to build the intuition that although energy savings in DutyCode are expected to increase with higher sleep interval, this can only occur until a certain point/threshold (i.e., a certain sleep interval). Below this threshold, nodes sleep very little, and are still awake while useless packet transmissions are taking place. Longer sleep intervals (i.e., duty-cycling) will allow them to save more energy. After the threshold point, however, nodes lose opportunities for network coding, i.e., there are packets being transmitted, but because nodes sleep, they do not receive the packets. Consequently, nodes will not be able to decode coded packets, and more NACKs and ReNACKs will be transmitted. At this point the energy efficiency of duty-cycling is much less than the energy lost because of retransmissions caused by sleeping when network coding is taking place.

We measured energy consumption and total flooding time by varying SI in the [4msec, 60msec] range, while keeping other parameters constant. Figure 13 depicts our results, which confirm our intuition. As SI increased, the energy consumption of DutyCode gradually decreased until SI was 45ms, after which it started to increase. Following our analysis in Section 6, the maximum energy saving is achieved when the sleep duration matches the stream duration. The explanation for the 45ms optimal sleep interval is as follows. In these set of experiments, we do not employ ECS - we simply use the default coding scheme that AdapCode uses (i.e., based on neighbor density). In our testbed, due to a relatively uniform deployment, all nodes have a coding scheme of 2. This means that the 8 packet page is coded in 4 packets. Since a stream has 4 packets, the opportunity for sleeping is only for the last 3 packets in the page. Transmitting a single packet in TinyOS takes about 8ms. Hence transmitting 3 packets, coupled with the additional small backoffs between streaming packets (i.e., BO_r) will total around 45ms.

The experimental results follow the analytical result, as the maximum energy saving for our testbed was achieved when SI = 45msec. The theoretical upper bound of energy saving is also shown in the figure for comparison. The maximum energy savings achieved for our testbed was 42%, while the theoretical bound is 66%.

We also compare the energy consumption of DutyCode with that of NoCode, a modified version of AdapCode which does not use network coding, but uses duty

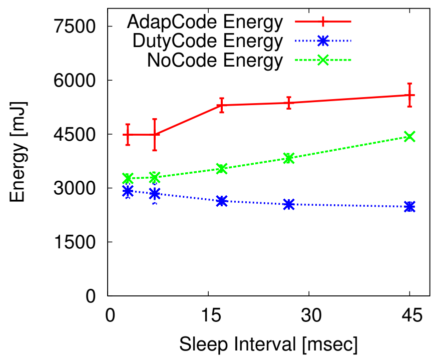


Figure 14: Energy consumption of DutyCode, AdapCode and NoCode.

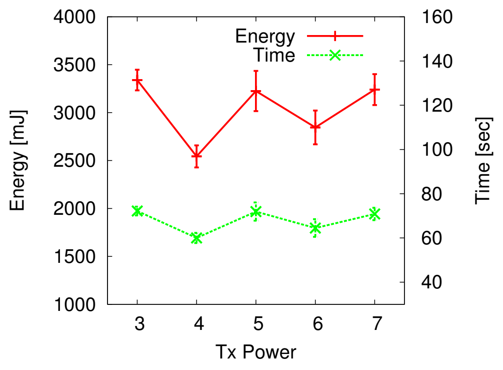


Figure 15: The effect of node density on energy consumption and total flooding time for DutyCode.

cycling. The results are presented in Figure 14. As shown, DutyCode consumes less energy than NoCode. Interestingly, NoCode consumes less energy than AdapCode, revealing that, for our scenario, duty-cycling can be more energy efficient than network coding. Nevertheless, by combining network coding with duty cycling, DutyCode can achieve more aggressive energy savings.

8.3. *Node Density*

In this experiment we explore the impact of node density on energy efficiency and total time for flooding in DutyCode. We are interested in the effects of node density because a higher node density causes more collisions, thus increasing energy consumption and total time for flooding. At the same time, however, a higher node density might also decrease network diameter, assuming that the network size is fixed, like our testbed. This decrease in the network diameter would result in lower energy consumption and shorter time for flood

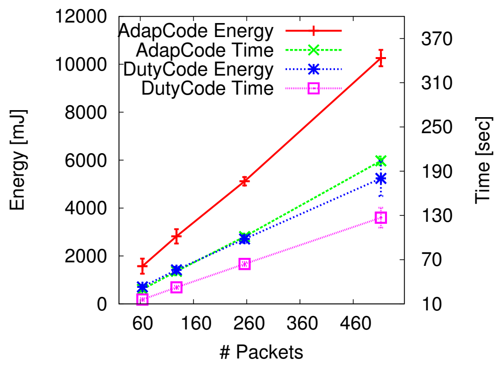


Figure 16: The effect of total number of packets on energy consumption and total flooding time.

propagation.

In our experiments we varied the TX Power from 3 to 7. For TX Power levels above 7, the network becomes a single hop network. We keep all other parameters constant. Our experimental results are shown in Figure 15. The aforementioned effects can be observed from Figure 15. As TX Power increased from 3 to 4 and from 5 to 6, the energy consumption and time for flooding decreased, due to a reduced network diameter (i.e., fewer hopes were needed to reach all nodes in the network from the source), despite the higher number of collisions. However, for increases of TX Power from 4 to 5 and from 6 to 7, the network diameter remained constant. Consequently, only the higher number of collisions influenced the performance, increasing energy consumption and time for flooding to complete.

8.4. *Total Number of Packets*

In this experiment we evaluate the impact the total number of packets has on energy efficiency and flooding time. We expect that flooding more packets in the network will result in higher energy consumption and longer time for flooding.

Our performance evaluation results are depicted in Figure 16. We can observe that, as the total number of packets increased, both energy consumption and flooding time increased. This is because more packets being transmitted increase total transmission time and also the probability of collisions. Interestingly, as the number of packets increased from 64 to 512 (700%), power consumption increased by 600%. This increment is not strictly linear because, as the number of packets increases, nodes find more appropriate coding schemes, thus decreasing the likelihood of redundant/useless transmissions. As a reminder, we note here

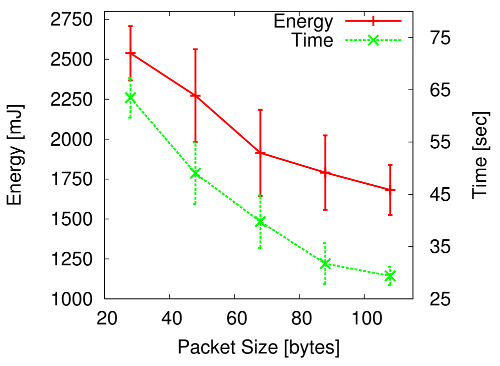


Figure 17: The effect of packet size on energy consumption and total time for flooding.

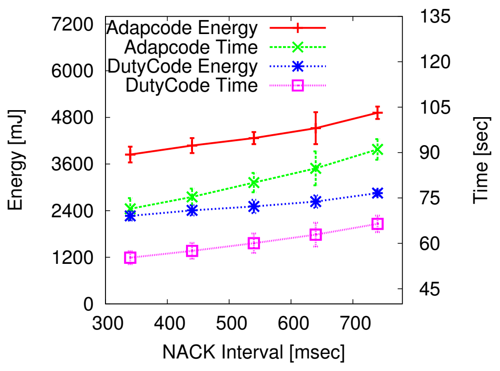


Figure 18: The effect of NACK interval on energy consumption and total time for flooding.

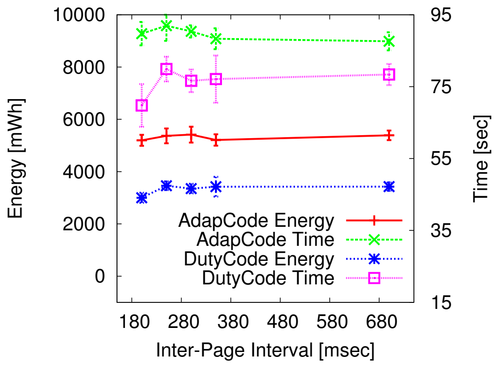


Figure 19: The effect of Inter-Page Interval on energy consumption and total time for flooding.

that in these sets of experiments we employ the same algorithm for deciding the coding scheme, as AdapCode. We investigate the performance of our ECS algorithm in Section 8.8.

It is also interesting to note that in this experiment, although energy savings increase from 866mJ to 5,021mJ, when compared to AdapCode, our solution's savings reduced from 55% to 48%. We attribute this reduction in power savings to reduced redundant/useless transmissions.

8.5. Packet Size

In this experiment we investigate the effect of packet size on energy efficiency and total time for flooding. We obtained performance results for packet sizes in the range [28bytes, 108bytes]. We keep all other parameters constant.

The results are depicted in Figure 17. As shown, as the packet size increased, both energy consumption and total time for flooding decreased. One can observe that when the packet size increases 3-fold, the energy consumption decreases by 33%. This emphasizes that in a packet transmission, the backoff intervals are significantly longer than the time taken for actual packet transmission. From our experiments, it appears that more energy can be saved by sending a few large packets instead of many small ones. A possible explanation is the good link quality in our testbed.

8.6. NACK Interval

In this experiment we investigate the effect NACK interval has on energy efficiency and total time of flooding. As mentioned before, a node waits for "*NACK Interval*" to receive a useful packet without transmitting a NACK). An increase in NACK Interval time is expected to generate additional delays and potential energy inefficiencies. The NACK intervals we chose are in the range [340msec, 740msec].

The results are depicted in Figure 18. As shown, the increase in the NACK Interval results in an increased energy consumption and total time for flooding, for both DutyCode and AdapCode. One can also observe that DutyCode is slightly less affected by longer NACK Intervals, i.e. the slopes of curves depicting DutyCode energy consumption and total time for download are smaller than for AdapCode.

8.7. Inter-Page Interval

In AdapCode and DutyCode, the source node maintains a time gap, called Inter-Page Interval, between subsequent page transmissions. In this section we investigate the effect this interval has on the energy consumption and total time for flooding. For this evaluation, we use Inter-Page Intervals in the range [180msec, 700msec] while keeping all other parameters constant.

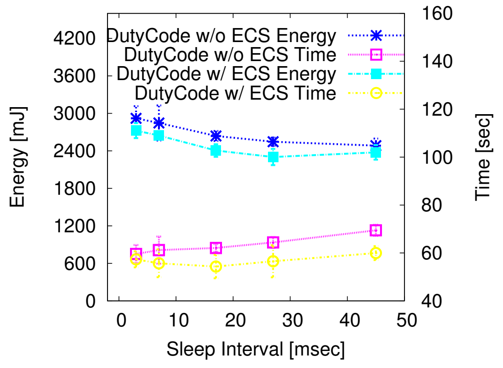


Figure 20: The effect of sleep interval in DutyCode with and without ECS.

The results of our evaluation are depicted in Figure 19. We observe that the Inter-Page Interval, over the rage we considered, does not have a noticeable influence on the energy consumption or the total time for flooding of DutyCode and AdapCode. Consequently, we infer that as long as the increase in Inter-Page Interval does not increase the idle time of nodes (i.e., time when nodes do not have anything to transmit) in the network, the Inter-Page Interval does not significantly affect energy efficiency and total time for flooding of DutyCode and AdapCode.

8.8. DutyCode with ECS

.

In this section, we investigate the performance gain from integrating ECS with DutyCode. We also examine the impact of Link Quality Threshold (LQT) on ECS. As presented in Section 4, LQT is a design parameter of ECS. Different LQT values result in different topologies, thereby affecting the performance of ECS.

We measured energy consumption and total flooding time for both DutyCode with ECS and DutyCode without ECS by varying sleep interval. For these experiments, LQT = 0.95. We chose this high value (i.e., indicating we only used highly symmetric links) to obtain accurate coding schemes (i.e., children and parent nodes can communicate symmetrically) and ensure sufficient redundancy for packet decoding. The results are depicted in Figure 20. The patterns for energy consumption and flooding time of DutyCode with ECS was similar to that of DutyCode. The graph also shows that DutyCode with ECS outperforms DutyCode without ECS, by approximately 10%.

We also measured the total number of packet transmissions for both DutyCode with ECS and DutyCode without ECS, by varying LQT. The results are shown

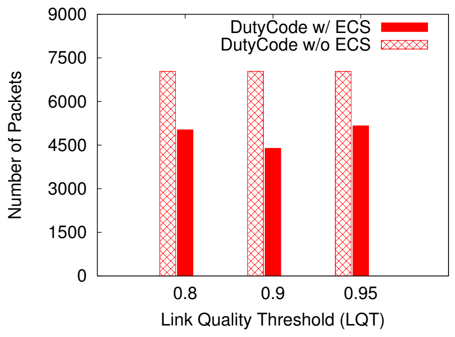


Figure 21: The effect of LQT on the total number of packets transmitted in the network.

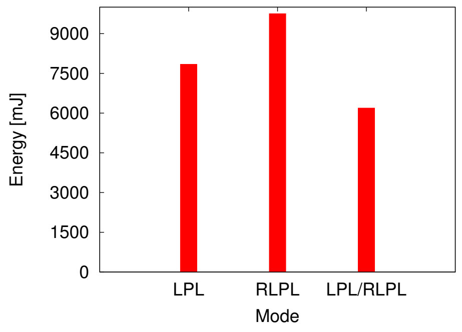


Figure 22: The effect of MAC protocol on the energy consumption of DutyCode.

in Figure 21. As expected, DutyCode with ECS outperforms DutyCode without ECS, in terms of the total packet transmissions, regardless of LQT. As LQT increases from 0.8 to 0.9, the total number of packet transmissions for DutyCode with ECS decreased. This is because at higher LQT (i.e., higher symmetry in communication), the coding schemes derived are more accurate. Interestingly, the increase of LQT from 0.9 to 0.95 actually increased the total number of transmissions. This is because the total number of valid links (i.e., links above the threshold) tends to decrease with extremely high LQT, thereby allowing only very few redundant transmissions.

8.9. LPL/RLPL Mode Transition

To assess the effects of LPL/RLPL Mode Transition, we evaluate DutyCode is LPL mode, in RLPL mode and in the LPL/RLPL “protocol transition” mode. The Sleep Interval value we chose is 45msec for these experiments. This is because DutyCode in RLPL mode achieves higher energy savings, when compared to DutyCode in LPL mode, at this value for the Sleep Interval. We measure the energy consumption over a time period of 150sec. To be noted is the fact that the execution of the flooding application is less than 150sec. The longer time interval allows us to illustrate inefficiencies in RLPL, when network flooding is not present.

The results are shown in Figure 22. As expected, the energy consumption of RLPL mode is high as the nodes are awake most of the time, even when the flooding application does not execute. As shown, the LPL/RLPL “protocol transition” mode is 20% more energy efficient than the LPL mode.

9. Related Work

In the area of duty cycling, research has often examined low power listening (LPL) and scheduling. B-MAC [4] is a simple LPL protocol with periodic listening that requires no synchronization. However, in high traffic networks, throughput is impacted. X-MAC [7] improves B-MAC by using ACKs, but suffers similar inefficiencies in networks using broadcast. Wise-MAC [8] enhances efficiency by creating opportunities for synchronization, but is designed for low traffic networks. In SPAN [9], average sleep time is lengthened but common network configurations cause power exhaustion in nodes on high traffic routes. S-MAC [10] uses adaptive, periodic sleep, and clustering. Although it is efficient at low bandwidth, performance degrades at higher network loads. T-MAC [11] enhances S-MAC by reducing the awake period even more. However, nodes frequently miss useful packets while asleep. SCP [12] saves power by scheduling coordinated transmission and listen periods. However, high network loads reduce sleep opportunities. DW-MAC [13] is another scheduling protocol that allows nodes to wake up on demand. AS-MAC [14] achieves scheduling through periodic hello packets but fails to optimize efficiency because the hello packet has to be transmitted at the wake up intervals of each neighbor. RI-MAC [15] is a receiver initiated MAC protocol with an aim to reduce the idle-listening. But, scheduling algorithms do not apply for broadcast applications. The sleep and awake durations (i.e., duty cycle) for each node are computed as an optimization problem for unicast transmissions [16]. Opportunistic flooding [17] and Schm-Dist [18] save energy in a low duty-cycling networks by treating broadcast transmissions as unicasts. ADB [19] achieves efficient broadcast in asynchronous duty-cycling networks, through collaboration among nodes achieved by additional information in the packet footer. These technique may not scale to large scale and message intense networks because each transmission is handled as a transmission to each neighbor individually.

A variety of network coding approaches have also been proposed. With COPR [20], Cui, et.al. maximize throughput by combining several unicast packets into a single broadcast packet. BEND, Zhang, et.al. [21] improves packet delivery rates, reducing retransmissions, but negates much of the energy savings by forwarding multiple copies of the same packet. Energy-efficiency at intermediate nodes was examined in [22] where Markov chains were used to determine bounds on energy consumption. Inspired by Reed-Solomon codes, network coding based on raptor codes is proposed for video streaming on lossy packet networks in [23]. Decreased errors contribute to higher throughput and reduced power consumption in [24]. Multimedia throughput and energy-efficiency in wireless networks is examined in [25]. However, existing coding schemes did not take duty cycling into consideration. CODEB [26] uses Reed-Solomon based coding algorithm for achieving optimal coding. Cluster based network coding scheme is proposed in [27], to minimize the redundancy in messages transmitted. But these schemes are done for unicast message patterns. Similarly, TEEM [28], Max-MAC [29] and BEAM [30] are traffic aware MAC protocols that deal with unicast messages. P-MAC [31] may not scale to a large, message-intense network, as it requires periodic traffic pattern update to achieve traffic aware duty-cycling.

In [1], we investigated the integration of network coding with duty-cycling in flood-based WSN. This article improves the energy efficiency of [1] by proposing ECS, a coding decision algorithm that minimizes the redundant packet transmissions, thereby saving more energy. Furthermore, an adaptive transition technique accomplishes a smooth and timely transition between LPL and RLPL without packet loss.

10. Conclusions

Network coding and duty-cycling are two popular techniques for saving energy in wireless sensor networks. In this article, we demonstrate that although they achieve energy efficiency by conflicting means, they can be combined for more aggressive energy savings in flood-based sensor network applications. To achieve aggressive energy savings we propose DutyCode, a network coding friendly MAC protocol which implements packet streaming and allows the application to decide when a node can sleep. Through analysis and real system implementation we demonstrate that DutyCode does not incur higher overhead than state of art solutions, and that it achieves up to 46% more energy savings when compared with network coding-based solutions that do not use duty-cycling. The proposed scheme requires minimal changes to existing network coding applications. We also present ECS, a technique that optimizes the network coding scheme of each node. We develop an integrated network coding with duty cycling solution, which allows smooth MAC protocol transition between LPL (i.e., more energy efficient when flooding is not taking place) and RLPL, which is more energy efficient when network flooding occurs. We demonstrate the effectiveness and practically of our proposed solutions analytically and through real system implementation and evaluations.

Acknowledgements

The authors would like to thank Manish Kumar Singh, who participated in the initial discussions of the project. This work was supported in part by NSF grants #0923203, #1127449, #1145858.

References

[1] R. Chandanala, R. Stoleru, Network coding in duty cycled sensor networks, in: International Conference on Networked Sensing Systems (INSS), 2010.
[2] R. Ahlswede, N. Cai, S.-Y. R. Li, R. W. Yeung, Network information flow, IEEE Trans. Inf. Theory 46 (2000) 1204–1216.
[3] I.-H. Hou, Y.-E. Tsai, T. F. Abdelzaher, I. Gupta, AdapCode: Adaptive network coding for code updates in wireless sensor networks, in: Proceedings of INFOCOM, 2008.
[4] J. Polastre, J. Hill, D. Culler, Versatile low power media access for wireless sensor networks, in: Proceedings of SenSys, 2004.
[5] P. Dutta, J. Taneja, J. Jeong, X. Jiang, D. Culler, A building block approach to sensornet systems, in: Proceedings of SenSys, 2008.
[6] Texas Instruments Inc., CC2420 Data Sheet.
[7] M. Buettner, G. V. Yee, E. Anderson, R. Han, X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks, in: Proceedings of SenSys, 2006.
[8] A. El-Hoiydi, J.-D. Decotignie, WiseMAC: An ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks, in: Proceedings of ISCC, 2004.
[9] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris, SPAN: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks, Wirel. Netw. 8 (5) (2002) 481–494.
[10] W. Ye, J. Heidemann, D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, in: Proceedings of INFOCOM, 2002.
[11] T. V. Dam, K. Langendoen, An adaptive energy-efficient MAC protocol for wireless sensor networks, in: Proceedings of SenSys, 2003.
[12] W. Ye, F. Silva, J. Heidemann, Ultra-low duty cycle MAC with scheduled channel polling, in: Proceedings of SenSys, 2006.
[13] Y. Sun, S. Du, O. Gurewitz, D. B. Johnson, DW-MAC: a low latency, energy efficient demand-wakeup MAC protocol for wireless sensor networks, in: Proceedings of MobiHoc, 2008.
[14] B. Jang, J. B. Lim, M. Sichitiu, AS-MAC: An asynchronous scheduled MAC protocol for wireless sensor networks, in: Proceedings of MASS, 2008.
[15] 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: Proceedings of SenSys, 2008.
[16] S. C. Ergen, C. Fischione, D. Marandin, A. L. Sangiovanni-Vincentelli, Duty-cycle optimization in unslotted 802.15.4 wireless sensor networks, in: Proceedings of GLOBECOM, 2008.
[17] S. Guo, Y. Gu, B. Jiang, T. He, Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links, in: Proceedings of Mobicom, 2009.
[18] J. Hong, J. Cao, W. Li, S. Lu, D. Chen, Sleeping schedule-aware minimum latency broadcast in wireless ad hoc networks, in: Proceedings of IEEE International Conference on Communications (ICC), 2009.
[19] Y. Sun, O. Gurewitz, S. Du, L. Tang, D. B. Johnson, ADB: an efficient multihop broadcast protocol based on asynchronous duty-cycling in wireless sensor networks, in: Proceedings of SenSys, 2009.
[20] T. Cui, L. Chen, T. Ho, Energy efficient opportunistic network coding for wireless networks, in: Proceedings of INFOCOM, 2008.
[21] J. Zhang, Y. Chen, I. Marsic, Network coding via opportunistic forwarding in wireless mesh networks, in: Proceedings of WCNC, 2008.
[22] J. Goseling, R. Boucherie, J.-K. van Ommeren, Energy consumption in coded queues for wireless information exchange, in: Proceedings of Network Coding, Theory, and Applications, 2009.
[23] N. Thomos, P. Frossard, Raptor network video coding, in: MV '07: Proceedings of the international workshop on Workshop on mobile video, 2007.
[24] Y. Xiao, L. Ma, K. Khorasani, A. Ikuta, A new robust narrowband active noise control system in the presence of frequency mismatch, IEEE Transactions on Audio, Speech & Language Processing 14 (6) (2006) 2189–2200.
[25] X. Tao, C. Zhang, J. Lu, Network coding for energy efficient wireless multimedia transmission in ad hoc network, in: Proceedings of International Conference on Communication Technology (ICCT), 2006.
[26] L. Li, R. Ramjee, M. Buddhikot, S. Miller, Network coding-based broadcast in mobile ad hoc networks (2007).
[27] T.-G. Li, C.-C. Hsu, C.-F. Chou, On reliable transmission by adaptive network coding in wireless sensor networks, in: Proceedings of ICC, 2009.
[28] H. Gong, J. Cao, M. Liu, L. Chen, L. Xie, A traffic aware, energy efficient MAC protocol for wireless sensor networks, Int. J. Ad Hoc Ubiquitous Comput. 4 (3/4) (2009) 148–156.
[29] P. Hurni, T. Braun, MaxMAC: A maximally traffic-adaptive MAC protocol for wireless sensor networks, in: Proceedings of EWSN, 2010.
[30] M. Anwander, G. Wagenknecht, T. Braun, K. Dolfus, Beam: A burst-aware energy-efficient adaptive mac protocol for wireless sensor networks, in: International Conference on Networked Sensing Systems (INSS), 2010.
[31] T. Zheng, S. Radhakrishnan, V. Sarangan, PMAC: An adaptive energy-efficient MAC protocol for wireless sensor networks, in: Proceedings of IPDPS, 2005.