Network Coding in Duty-Cycled Sensor Networks

Roja Chandanala, Radu Stoleru, *Member, IEEE*

***Abstract*—Network coding and duty-cycling are two popular techniques for saving energy in wireless adhoc and sensor networks. To the best of our knowledge, the idea to combine these two techniques, for more aggressive energy savings, has not been explored. One explanation is that these techniques achieve energy efficiency through conflicting means, e.g., network coding saves energy by exploiting overhearing, whereas duty-cycling saves energy by cutting idle listening and, thus, overhearing. In this paper, we evaluate the use of network coding in duty-cycled sensor networks. We propose a scheme, called DutyCode, in which a MAC protocol implements packet streaming and allows the application to decide when a node can sleep. We investigate our scheme analytically and implement it on mote hardware. We evaluate our solution in a 42-node indoor testbed. Performance evaluation results shows that our scheme saves 20-30% more energy than solutions that use network coding, but do not use duty-cycling.**

I. Introduction

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

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 [1], many applications have incorporated this idea. 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 be found in applications including multi-cast, content distribution, delay tolerant networks (DTN), underwater sensing suites, code dissemination, storage, and security. As diverse as these applications are, they all share a common assumption: nodes in the 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 management to almost complete system shutdown, duty cycling extends node lifetime and reduces maintenance. When done properly, duty cycling can extend battery life by an order of magnitude or more. In WSN, duty cycling is pervasive and almost all deployed systems integrate some level of 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, network coding becomes more difficult because the time available for overhearing is reduced. Sleep cycles using a fixed intervals as short as 3ms result in increased energy consumption and delay instead of shrinkage.

In this paper we address the challenge faced when aggressive (i.e., both duty-cycling and network coding are employed) energy savings are mandated in flooding-based WSN applications. We particularly target applications such as code dissemination that require a non-negligible amount of time, possibly tens of minutes in large scale sensor networks. Since network coding mandates nodes to be awake to make the maximum use of coding/decoding opportunities, it may seem inefficient to allow nodes to sleep. Our main idea is derived from the intuition that, due to the redundancy of coding, there are periods of time when a node does not benefit from overhearing coded packets being transmitted. We seek to precisely determine these periods of time, and let nodes that do not benefit from these "useless" packets, to sleep. Our solution is a cross layer approach, where the MAC layer facilitates streaming, random sleeping and synchronization, and the network coding-aware application layer determines, based on the stream being transmitted, the time to sleep and the sleep duration. The prerequisite of our solution is that network coding be applied individually to a sequence of packets, called "page". The packets that are to be coded within a page is random, however.

The contributions of our paper include:

- A media access control (MAC) protocol that supports streaming. This allows nodes to use streams to predict packet arrival.
- A mechanism for randomizing sleep cycles using elastic intervals. This allows nodes to intelligently select sleep periods.
- Energy and delay models and simulations demonstrating the energy efficiency and throughput of this solution.
- An implementation on mote hardware and performance evaluation in 42-node testbed where actual energy consumption is measured.

This paper is organized as follows. Section II motivates our work and provides background on network coding. Sections III and IV present our scheme for network coding in duty-cycled environments, and its analysis, respectively. Sections V and VI describe the implementation and performance evaluation of our scheme. We review the state of art in Section VII and conclude in Section VIII.

II. Motivation and Background

Network coding enhances energy efficiency by reducing the number of transmissions. The basic concept of network coding can be explained using a simple scenario [2]. Sender s_1 wants to send a packet x_1 to t_1 and sender s_2 wants to send another packet x_2 to t_2, as shown in Figures 1(a) and 1(b). At total of

Roja Chandanala and Radu Stoleru are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77840 USA (e-mail:{rojac, stoleru}@cse.tamu.edu)

978-1-4244-7910-8/10/$26.00 ©2010 IEEE

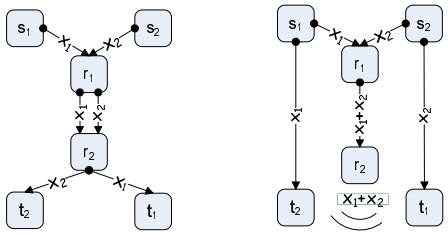

Fig. 1. a) Without network coding 6 transmissions are required for sending packets x_1 and x_2 from s_1 and s_2 to t_1 and t_2, respectively; b) with network coding, only 4 transmissions are sufficient.

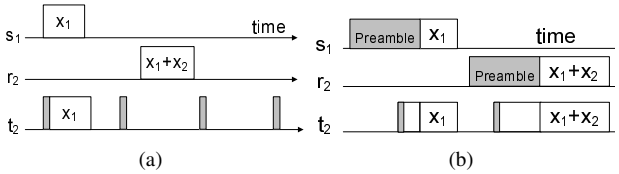

Fig. 2. Network coding integrated with (a) static sleeping schedule (b) Low Power Listening based on long preambles.

six transmissions are required to deliver the two packets when network coding is not used (Figure 1(a)). However, Figure 1(b) shows that, when network coding is used, only 4 transmissions are needed because the two relays can transmit only one coded packet (x_1+x_2) instead. For network coding to work, receivers t_1, t_2 must be able to overhear packets x_1 and x_2 from s_1 and s_2, respectively. Otherwise, they will be unable to decipher anything from the coded packet received.

It is important to note that, unlike normal broadcast 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 following independent coded packets:

$$a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4$$
$$b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$$
$$c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$$

Receiving another coded packet: $d_1x_1+d_2x_2+d_3x_3+d_4x_4$, is critical for this node in order to decode all the packets. Otherwise all 3 received packets are useless. As the coding scheme (i.e the number of different packets coded into a single packet) increases, the penalty for loosing a single packet increases linearly.

As previously noted, most existing duty-cycling protocols achieve energy saving through static sleeping or scheduling. However, for network coding, scheduling is not a feasible solution because overhearing is required. If senders were aware of the destinations of each packet, it might be possible to only transmit when the appropriate receivers were awake. However, senders are typically unaware of the identity of every receiver. Also, the coordination required to ensure that all intended receivers were awake at the same could be prohibitively costly. Static sleeping is similarly unsuited due to the high likelihood of missing a useful packet while asleep. Consider the example in Figure 1(b) when network coding is used with duty-cycling, and two solutions, (illustrated in Figure 2(a) and Figure 2(b)):

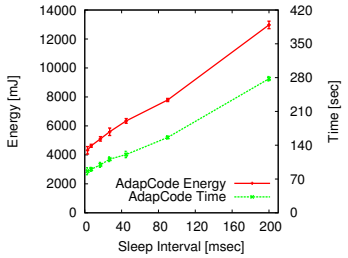

Fig. 3. Execution of X-MAC-enabled AdapCode on a 42 epic mote testbed. Energy per node and total execution time as functions of the Sleep Interval parameter of X-MAC.

a rigid, static duty cycle, or Low Power Listening [3]. As shown in Figure 2(a) if node t_2 does not receive the coded packet, it will not be able to obtain the packet destined for it, namely x_2. If LPL is used, as depicted in Figure 2(b) the total time for execution is increased, causing considerable energy expenditures because of the overhead caused by preamble transmission. And this overhead increases with the increase in LPL interval.

To validate our observations we performed experiments on a testbed of 42 epic motes and 256 packets of data to be transmitted to all the motes from a source. We integrated AdapCode [4], a code dissemination application that uses network coding, with X-MAC [5], a frequently used MAC protocol that allows duty cycling. The results of our experiment, in which we varied the LPL sleep interval parameter (indicative of the duty-cycle desired), are depicted in Figure 3. Although some degradation due to missed transmissions was expected, especially at longer sleep intervals, energy consumption was generally expected to decrease. Instead, even using very short static sleep intervals nearly doubled the execution time and energy consumption. From these results, it was clear that: i) a node should select sleep intervals intelligently at non-static intervals; and ii) long preamble based MAC solutions are not suitable for network coding applications. The problem formulation that emerged was for each node to predict the likelihood of receiving packets and decide to sleep if the packets are likely to be useless.

III. Network Coding in Duty-Cycled WSN

The core problem of combining duty-cycling and network coding is non-intelligent duty-cycling. Our solution tackles this problem by providing a framework to keep a node informed of the future packets without any use of control packets. This knowledge makes a node capable of employing smart duty-cycling. The fundamental principle of our solution is that each node streams all the packets of a logical entity (i.e., page) in a row. This stream is useful for nodes lacking the data being transmitted, otherwise it is useless. Upon receiving the first packets a node stays awake and receives all packets if they are useful, otherwise the node sleeps for the duration of the stream. A node can compute the duration of stream as the transmission time per packet times the remaining packets of stream.

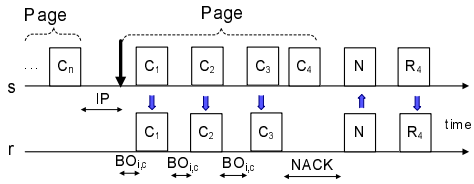

Fig. 4. The NetCode protocol: C_i are coded packets, N is a NACK packet, due to missing packet C_4, R_4 is the ReNACK packet - the non coded packet missing, IP is the Inter-Page Interval, $BO_{i,c}$ is the backoff interval - initial and congestion (NOTE: the backoff is between any two packets), NACK is a timer.

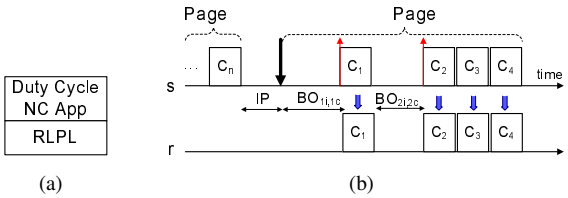

Fig. 5. (a) DutyCode architecture; (b) Streaming in DutyCode: after the backoff intervals, BO_1 and BO_2 packets in a page are streamed.

A. Preliminaries

In this subsection we introduce a typical application, called NetCode, that uses network coding for energy efficiency. We describe its operation in detail because we aim to analytically demonstrate, in the sections that follow, that the introduction of our smart duty-cycle does not come with any major overhead. The operation of NetCode is depicted in Figure 4. In NetCode, when a source node, e.g., a base station, wants to disseminate a new program image in the network, broadcasts the data as pages. Each page consists of a number of packets. After transmitting a page, the source waits for a short period of time for the code propagation, and subsequently sends the next page. The time interval that separates two pages is called "*Inter-Page Interval*" (IP in Figure 4) . NetCode typically uses CSMA as a MAC protocol.

All nodes, after receiving packets, adaptively choose an appropriate "*coding scheme*" (i.e. the number of packets to be coded in a single packet) or have a predefined one. The appropriate coding scheme is chosen based on the number of neighbors. If a node does not receive any packets for a random period of time (called "*NACK delay*" in Figure 4), it broadcasts a NACK packet (labeled N in Figure 4), indicating the exact packets that it misses. 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 4). As with existing CSMA protocols, NetCode uses a "*Backoff timer*" for accessing the medium before transmitting any packet. This backoff timer has, typically, two values: an initial value, and a congestion value, selected randomly as depicted in Figure 4 from BO_i and BO_c range, respectively.

B. Proposed Solution

Our solution, called DutyCode and shown in Figure 5(a), is an integrated scheme (i.e., "MAC - NetworkCoding application") in which the MAC layer facilitates streaming, random sleeping and synchronization. The application determines the time to sleep and the sleep duration based on its knowledge about the stream being transmitted. The prerequisites of our solution are: i) packets are grouped into logical entities, called pages; and ii) network coding is limited to the packets from same logical entity.

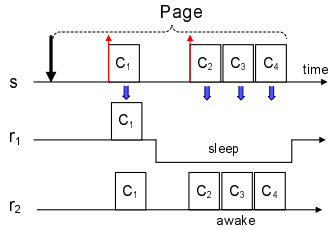

Fig. 6. Streaming when no packets are awaiting transmission at r_1 and r_2

The proposed MAC protocol, Random Low-Power-Listening (RLPL), allows: i) packet streaming; ii) transmission defer; and iii) transmission arbitration. When requested by network coding (NC) application, RLPL turns off the radio for the requested duration if there is no pending transmission. The NC application specifies the sleep duration when it requests the node to sleep. RLPL does not put the node to sleep periodically. When requested, and if feasible, RLPL shuts down the radio for the requested period. Unlike other duty-cycling protocols (e.g., DefaultLPL in TinyOS 2.1) it does not perform Clear Channel Assessment (CCA) before turning off the radio. The reason for this is that the CCA would not have any meaning, since requests for sleep come from the NC application when there is, typically, ongoing radio communication (e.g., streaming of useless packets). We define "useless packets" as the packets pertaining to a page which was already decoded by the receiving node. We define SI,"Sleep Interval" as the duration per packet, for which a node sleeps upon receiving a packet from a useless stream.

C. Packet Streaming

For streaming, RLPL sets different initial and congestion backoff intervals for packet transmission. The operation of DutyCode is depicted in Figure 5(b). The first two packets of the stream are transmitted normally with random backoff intervals chosen from different ranges and the rest of the stream is sent without any backoff. In streaming, the penalty for transmission collision is high. To reduce collisions, the first two packets of the stream are sent with large random backoff intervals. As shown, backoff intervals for the first packet and second packet are selected from BO_1 and BO_2 respectively (each has one initial, and one congestion value: BO_{1i}, BO_{1c} and BO_{2i}, BO_{2c}). Application can set these values according to the reliability of the network.

Upon receiving a stream packet, a node yields if it has no packets awaiting transmission. As shown in Figure 6, upon receiving a stream from node s, nodes r_1 and r_2 do not process any transmit requests coming from their NC application, for the duration of the stream from s. As shown, because r_1 has the page being transmitted by s, it sleeps for the duration

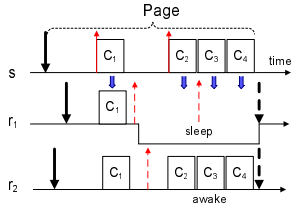

Fig. 7. Streaming with transmission defer (vertical dotted arrows) at nodes r_1 and r_2.

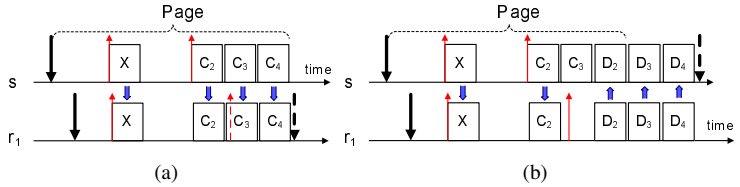

Fig. 8. Transmit arbitration (on packet X collision) based on node ID: a) when $r_1 > s$; and b) node's s ¡ r_1. Vertical dotted arrows represent packet transmissions deferred.

of the stream. After r_1 wakes up, it tries to transmit any pending packet. Similarly, node r_2 stays awake and receives all streamed packets. Node r_2 handles its packet transmission request, either after it receives the last packet of the stream from node s or after the expected stream duration is over, whichever happens first.

D. Transmission Defer

In Figure 7, nodes r_1 and r_2 yield to node s. This is similar to the case of regular streaming (Figure 6, except that nodes r_1 and r_2 defer packet transmissions. At any time the application can only have one pending transmission. Hence, r_1 and r_2 handle the deferred packet transmission when they realize that the stream transmission by node s is over. Deferred packet is handled as a new transmission request i.e., the node backs off for the duration selected randomly from initial backoff interval. A transmission defer is similar to transmission cancelation except that it is completely handled in MAC. RLPL handles the deferred and pending packets after the sleep interval is over and radio is turned on.

E. Transmission Arbitration

Upon receiving a packet of a stream from node s, node r_1 yields to s if $r_1 > s$, where s and r_1 are unique node IDs. In Figure 8(a), r_1 yields to s because there should only be one active stream transmission. When a node yields to another node, it defers its transmission, if feasible. A node does not transmit any packets until it receives the last packet of the stream it is yielding to, or the expected duration of the stream is over, whichever happens first. Similarly, as shown in Figure 8(b), node s yields to node r_1 because $r_1 < s$. It is important to observe that, because of this arbitration, a node which decides to sleep because it detects a stream by node s, may miss useful packets being transmitted by r_1, if node s yields to r_1. We chose to allow this scenario because, instead of under-utilizing the channel by not transmitting any stream, we prefer to allow some node transmit its stream.

IV. DutyCode Protocol Analysis

In this section we analyze the operation of the proposed DutyCode scheme. The aims of our analysis are i) To show that the proposed duty-cycling enabled network coding does not have any overhead, when compared with existing network coding applications, such as NetCode. ii) To analyze the effects of backoff interval on the execution time. The two metrics we investigate are the total number of packets per page, and the total execution time of our scheme.

We denote by pp the number of packets per page, cs the coding scheme, cp the collision probability in NetCode, and cp_1 and cp_2 the collision probabilities associated with BO_1 and BO_2 in DutyCode (as described in the previous section), BO_c the CSMA congestion backoff interval and t_{tr} the actual transmission time per packet. We assume that the sleep interval per packet, SI is chosen such that there is no time overhead due to sleeping (i.e., stream duration is much longer than sleep interval): $SI \cdot pp/cs \leq (BO_{2i}/2 + pp/cs \cdot t_{tr})$.

Because, NetCode is message intense there is always a node waiting to transmit a packet with congestion backoff interval 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.

Similarly, in DutyCode, the average backoff time for the first packet of stream is $BO_{1i}/2$. The reason for this is that in DutyCode, because of packet defer, the first packet of a stream is always transmitted with a backoff interval chosen from 0 to BO_{1i}. Hence, the average backoff interval for the first packet is $BO_{1i}/2$ and 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 $cp_1 \cdot cp_2$, because there could be collision during the transmission of second packet if and only if there was collision during the first packet transmission.

A. Total Number of Packets Transmitted

Theorem 1. *If $BO_{1i} \geq BO_c$ then $P_p^d \leq P_p^a$, where P_p^d and P_p^a are the total number of packets for DutyCode and NetCode, respectively.*

Proof 1. *In both DutyCode and NetCode, three types of packets can contribute to the total number of packets: s) coded packets; b) NACK packets; and c) ReNACK packets.*

Coded Packets: *Coded packets are the packets transmitted by a node, obtained after coding (i.e., based on a coding scheme). Hence, the number of coded packets per page C_p is given by: $C_p = pp/cs$. Since the coding scheme is the same in DutyCode and NetCode:*

$$C_p^a = C_p^d \tag{1}$$

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

NACK Packets: *A node sends a NACK when it is unable to decode a page. There are two reasons for NACKs: i) unable to receive enough independent packets needed for decoding a page; and ii) collisions;*

i) independent packets: *Because DutyCode uses the same coding scheme as NetCode, this factor has no impact on the total number of packets. Hence, we do not consider it.*

ii) collisions: *In NetCode, the maximum the number of NACKs per packet is $N = cp + 2cp^2 + ...$, i.e., with probability cp , the node needs to send 1 NACK. If a collision occurs during the NACK transmission, the node may have to transmit 2 NACK packets with cp^2 probability. The total number of NACKs per page is:*

$$N_p^a = (pp/cs)(cp + 2cp^2 + ...)$$

For DutyCode, the NACKs can be sent as a result of two scenarios: i) collision during first packet transmission (cp_1); ii) collision during second packet transmission ($cp_1 \cdot cp_2$). Thus, the total number of packets per page is:

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

Since the collision probability of a transmission, cp, is inversely proportional to the backoff interval range BO, i.e., $cp \propto 1/BO$, then $cp = k_1/BO$ and $cp_1 = k_2/BO_{1i}$ when DutyCode and NetCode are run in the same network with similar parameters $k_1 = k_2$.

$$\frac{cp_1}{cp} = \frac{BO_c}{BO_{1i}} \tag{2}$$

If backoff intervals are chosen such that $BO_{1i} \geq BO_c$, then, from Equation 2:

$$cp_1 \leq cp \tag{3}$$

From Equation 3, when $pp/cs \geq (1 + cp_2)$ and $BO_{1i} \geq BO_c$ then:

$$N_p^d \leq N_p^a \tag{4}$$

For DutyCode, it is necessary to ensure $pp/cs \geq (1 + cp_2)$ otherwise there would only be 1 packet in the stream, and there would not be any opportunity for sleeping. It would be very easy to increase packets per page with an increase in coding scheme. One might argue that a node may miss some useful packets while asleep. A node goes to sleep, however, only after receiving a useful packet stream. If there is only one stream then the node would not miss any useful packets. Only because of collision there could be multiple streams at a time. Hence, the NACKs due to collisions cover this part.

ReNACK Packets: *ReNACKs are the regular packets with no coding.*

In NetCode, if there is a collision while transmitting a coded packet, it may need to send all the packets coded into that message which is the coding scheme. So, for each coded packet which is pp/cs per page, the node needs to retransmit cs with probability cp. Similar to NACKs it needs to transmit these packets twice with probability cp^2. Hence, the number of ReNACKs per page is: $R_p^a = cs \cdot pp/cs \cdot (cp + 2cp^2 + ...)$.

For DutyCode, a collision during a stream transmission can be attributed to one of the following: i) a collision during the transmission of first packet (probability cp_1), requires that cs packets be retransmitted; and ii) a collision during the transmission of the second packet (probability $cp_1 \cdot cp_2$), requires that an entire page be retransmitted: $cs \cdot cp_1 + cp_2 \cdot cp_1 \cdot pp$.

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:

$$R_p^d = pp \cdot (cp_1 + 2cp_1^2 + ...)$$

From Equation 3, when $BO_{1i} \geq BO_c$ then:

$$R_p^d \leq R_p^a \tag{5}$$

From Equations 1, 4 and 5, and since $P_p^{a/d} = C_p^{a/d} + N_p^{a/d} + R_p^{a/d}$, then $P_p^d \leq P_p^a$.

B. Total Execution Time

Theorem 2. *If the backoff intervals satisfy $pp/cs \cdot BO_c = BO_{1i}$ and $BO_c = BO_{1c}$, then the total time for DutyCode is less than or equal to that of NetCode.*

Proof 2. *In DutyCode, except for NACKs all other packets (i.e., coded packets and ReNACKs) can be transmitted as streams. If s is the total number of streams in DutyCode then the number of packets to be transmitted per node is:*

$$P^d = s \cdot pp/cs + N^d \tag{6}$$

In NetCode, the total number of packets can be written, in terms of s as:

$$P^a = s \cdot pp/cs + N^a \tag{7}$$

As explained, the average backoff time per packet in NetCode is $BO_c/2$. Hence, the total time per node is:

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

where D_t^a is the average time delay in NetCode (because NetCode packets are not streamed, a small time delay is maintained between successive transmissions). After substituting Equation 7, we obtain:

$$T_n^a = s \cdot pp/cs \cdot BO_c/2 + N^a \cdot BO_c/2 + P^a \cdot D_t^a + P^a \cdot t_{tr} \tag{8}$$

For DutyCode, for each stream the average backoff interval 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 stream which is transmitted right after the NACK is $BO_{1c}/2$. Consequently the total time per node is:

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

since $BO_{1c} = BO_c$. This may give a false impression that, by decreasing the backoff intervals, the total execution time can be decreased. However, with a decrease in backoff interval, the collision probability increases, which results in an increase in the number of NACKs and ReNACKs.

Given that $N^a \geq N^d$, $P^a \geq P^d$, $pp/cs \cdot BO_c = BO_{1i}$.

$$T_n^d \leq T_n^a \tag{9}$$

Algorithm 1 Streaming 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
```

Actually $pp/cs \cdot BO_c \geq BO_{1i} \geq BO_c$ is sufficient for proving 9. But, smaller BO_{1i} may result in increased collision probability. In order to avoid this, we chose the maximum bound for BO_{1i}, $pp/cs \cdot BO_c = BO_{1i}$. Another reason for this is that the increase in pp/cs increases the penalty for collision while transmitting the first packet of the stream. In order to minimize this penalty, BO_{1i} should also be increased sufficiently. When the backoff intervals are selected such that they satisfy the condition above, the total time for DutyCode is equal to that of NetCode.

V. Implementation

We implemented the DutyCode protocol in nesC for TinyOS 2.1.0. The implementation was done in the CC2420ReceiveC (Receive) and CC2420TransmitC (Transmit) modules. A new module RandomLPL (RLPL) was created, which differs from the existing DefaultLPL, as presented in Section III.

Due to space constraints, and the complexity of the design being concentrated in the Transmit module we describe only its implementation in detail. Specifically, we describe the scenario in which a node receives a stream packet. When streaming is achieved, the application sends packets one after the other without any significant delay. (i.e., sendDone() is signaled). Each packet header contains the number of remaining packets in the stream, computed based on the coding scheme used.

Upon receiving a stream packet from neighbors, the Receive module informs the Transmit. The module interaction is explained in Algorithm 1. After receiving a stream signal from Receive, 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 (line 2) if the stream satisfies the yielding conditions discussed in Section III. If not, no action is taken. Otherwise, if it is in the middle of transmission the node tries to defer the packet transmission (line 4) and informs RLPL about: i) the details of the stream transmission; and ii) the defer status, if there was a need for packet defer. RLPL then sets a NoSend timer (line 7) and keeps future transmit request pending until the stream duration is over or informed by transmit about completion of stream it is yielding to. Transmit informs RLPL about the completion of stream upon receiving the signal from receive for the last packet of the stream (line 10).

VI. Performance Evaluation

We evaluate the performance of DutyCode in an indoor testbed consisting of 42 epic motes [6] deployed in an approximately 500ft^2 area. Out of the 42 nodes, 14 are instrumented for power consumption measurements. The experiments are performed in a 5 hop network, obtained by a TX power setting of 4 for the sensor nodes. Each experimental point represents the mean of 3 executions of the protocol. Standard deviation is depicted in all our performance evaluation results.

For comparison with state of art we chose AdapCode [4], a flooding protocol that uses network coding and is representative of our NetCode model. The metrics we use for performance evaluation are the per node energy consumption and the total time required for the code update. While we are interested in the energy consumption, we also aim to not increase the total download time. The parameters that we vary are the sleep interval (SI), the node density (ND), the size of the packet (SP), the number of packets (NP), the NACK delay ($NACKD$), and the interpage and backoff intervals (II and BI, respectively). For these we use the following default values: SI=17, ND=4, SP=28, NP=256, $NACKD$=640, II=300, BI=320. The effects of these parameters is investigated and described in the remaining part of this section.

A. Sleep Interval

In this experiment we investigate how sleep interval SI affects the energy consumption and total dissemination time. SI is a critical parameter for DutyCode. We varied the sleep interval in the [4msec, 45msec] range, while keeping other parameters constant. The results are presented in Figure 9. Our initial intuition is that the energy savings should be incurred with non-zero sleep intervals, i.e., when a node has the opportunity to sleep. Furthermore, the energy consumption should decrease further with an increase in sleep interval, up to a certain point, after which, the energy consumption would start to increase again. Experimental results confirm our intuition. Confirming our analysis in Section IV, energy savings reach maximum when sleep interval is 35-44msec where the sleep duration matches stream duration. As shown, energy savings of 30% are achieved.

B. Node Density

In this experiment we varied the network density by changing the radio TX power. The number of packets NP used was 64. The results are depicted in Figure 10 where the transmission power is varied from 4 to 31 and the all other parameters are kept constant. The expectation was that at higher node densities, the opportunities for network coding and sleeping increases. However, the collision probability cp increases as well. As expected, energy savings increased with higher node density (from 4 to 22) and then deteriorated with increased density. As the node density increases pp, packets per page should be increased so that stream per page should at least contain more than one coded packet.

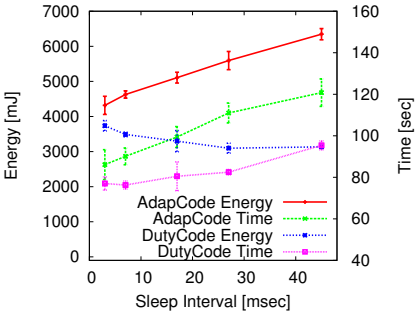

Fig. 9. The effect sleep interval has on energy consumption and time.

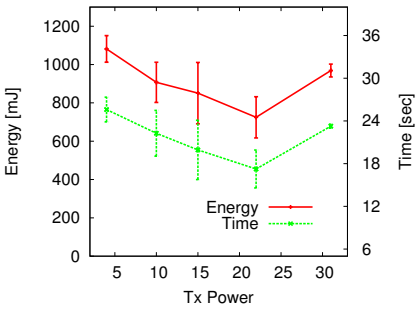

Fig. 10. The effect node density has on energy consumption and total time.

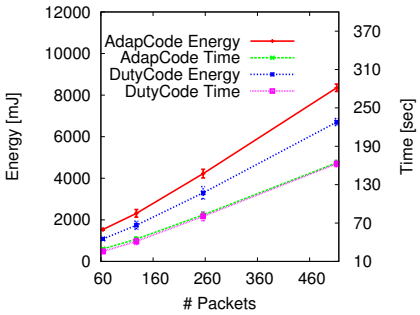

Fig. 11. Effects total number of packets has on energy consumption and total time.

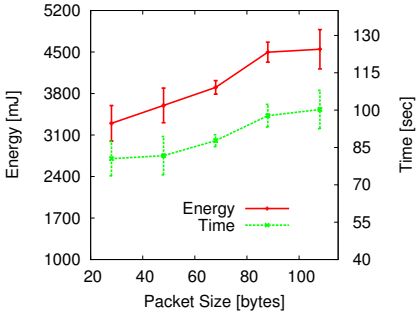

Fig. 12. Effects total number of packets has on energy consumption and total time.

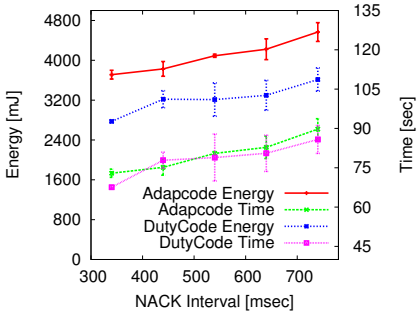

Fig. 13. Effects of NACK interval on energy consumption and total time.

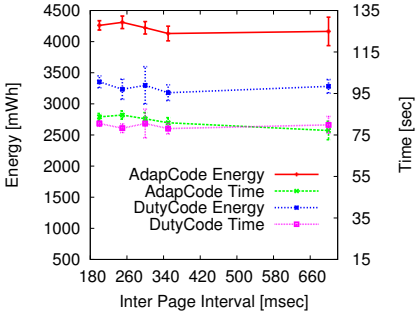

Fig. 14. Effects page interval has on energy consumption and total time

C. Number of Packets

We evaluated how the size of the data to be disseminated (i.e., number of packets) affects the energy savings. The results are depicted in Figure 11. As the total number of packets to be transmitted increases, the increase in the chances of packet collision and the total time for transmission results in increased power consumption. In our experiment, as the number of packets increased from 64 to 512, power consumption increased by 500%. This increment is not linear because, as the number of packets increases, nodes find the most appropriate coding scheme and useless transmissions are decreased. Although energy savings increase from 450mJ to 1650mJ, when compared to AdapCode, our solution's savings reduced from 30% to 20%. This reduction in power savings can be attributed to reduced redundant transmissions.

D. Packet Size

In this experiment we investigated the effect of packet size on energy efficiency and total time. The results are depicted in Figure 12, where equally spaced packet size values are selected in the range [28-108]. As the packet size increases, the computation time required also increases and the node needs a gap between successive packet receptions. Hence, the minimal backoff intervals for the stream packets is increased by 2msec for packet sizes 88 and 108 to minimize the penalty for losing useful packets. From our experiments, we observed that the energy consumption is not increased proportionally with the packet size.This is because with large packets time wasted on backoffs and computation is less compared to that of small packets.

E. NACK Interval

In this experiment we investigated the effect NACK interval has on energy efficiency and total time of DutyCode (as explained before, in NetCode a node waits for "*NACK Interval*" to receive a useful packet without transmitting a NACK). The results are depicted in Figure 13. The NACK intervals are chosen from the range [340-740]. An increase in the NACK time is expected to generate additional delays. As the NACK interval increases from 340 to 740, the energy savings of our solution decrease from 30% to 25%, a result of the additional delay and less opportunity for sleeping.

F. Inter-Page Interval

In typical flooding-based applications that use network coding, the source node maintains a gap between subsequent page transmissions. This is a design parameter of AdapCode. For this evaluation, we tested NetCode with different inter-page intervals ranging from 300-700 while keeping all other parameters constant, including $NACKD$. The results of our evaluation of inter-page intervals are depicted in Figure 14. As long as the increase in inter-page interval does not increase the idle time in the network (i.e., increase the time where nodes do not have anything to transmit), the interpage interval does not affect the total time taken and power consumption of both AdapCode and DutyCode.

G. Backoff Interval

Backoff interval is a design parameter of DutyCode. Experiments were run with different backoff intervals selected from the range [80, 480] to evaluate the effects of backoff interval on power consumption of each node and the total time delay.

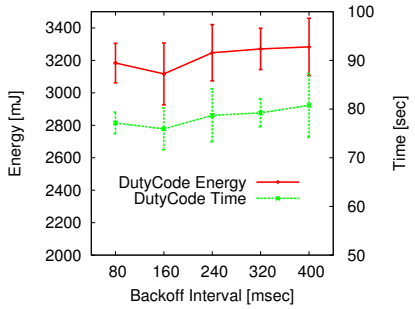

Fig. 15. Effects backoff interval has on energy consumption and total time.
.

As explained in Section IV, decreasing the backoff interval beyond an optimal value, does not decrease the total time taken. Our experimental results are depicted in Figure 15. As shown, as backoff interval increases from 80 to 160, total time and energy consumption decrease. However, a further increase in backoff interval resulted in an increased total time taken for the download and, thus, increased energy consumption. Hence, the optimal value of the backoff interval BO_{1i} is 160. For the rest of the performance evaluation experiments, the BO_{1i} is set to 320msec so that there is no increase in the total time and collision probability. It can be observed that the time taken by DutyCode is less than or equal to that of AdapCode in all performance evaluation figures except for few outliers.

VII. Related Work

In the area of duty cycling, research has often examined low power listening (LPL) and scheduling. B-MAC [3] is a simple LPL protocol with periodic listening that requires no synchronization. However, in high traffic networks, either throughput or sleep is impacted. X-MAC [5] improves B-MAC but suffers similar inefficiencies in networks using broadcast. Wise-MAC [7] enhances efficiency but is designed for low traffic networks. In SPAN [8], average sleep time is lengthened but common network configurations cause power exhaustion in nodes on high traffic routes. S-MAC [9] uses adaptive, periodic sleep, and clustering. Efficient at low bandwidth, performance degrades at higher network loads. T-MAC [10] enhances S-MAC by reducing the awake period even more. However, nodes frequently miss useful packets while asleep. SCP [11] saves power by scheduling coordinated transmission and listen periods. However, high network loads reduce sleep opportunities. Packet streaming increases throughput but limitations on streaming starts can cause delays.

A variety of network coding approaches have also been proposed. With COPR [12], Cui, et.al. maximize throughput by combining several unicast packets into a single broadcast packet. BEND, Zhang, et.al. [13] 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 [14] where Markov chains were used to determine bounds on energy consumption.

VIII. Conclusions

Network coding and duty-cycling are two popular techniques for saving energy in wireless adhoc and sensor networks. In this paper we demonstrate that although they achieve energy efficiency by conflicting means, they can be combined for more aggressive energy savings. To achieve these 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, and that it achieves 20-30% more energy savings when compared with network coding-based solutions that do no use duty-cycling. The proposed scheme requires minimal changes to existing network coding applications. We leave for future work an extension of our solution in which the length of the stream can be learned.

Acknowledgements

This work was funded, in part, by NSF grant CNS 0923203.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, 2000.

[2] C. chun Wang, "Beyond the butterfly a graph-theoretic characterization of the feasibility of network coding with two simple unicast sessions."

[3] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of SenSys*, 2004.

[4] I.-H. Hou, Y.-E. Tsai, T. F. Abdelzaher, and I. Gupta, "Adapcode: Adaptive network coding for code updates in wireless sensor networks," in *Proceedings of INFOCOM*, 2008.

[5] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *Proceedings of SenSys*, 2006.

[6] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler, "A building block approach to sensornet systems," in *Proceedings of SenSys*, 2008.

[7] A. El-Hoiydi and J.-D. Decotignie, "Wisemac: An ultra low power mac protocol for the downlink of infrastructure wireless sensor networks," in *Proceedings of ISCC*, 2004.

[8] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," *Wirel. Netw.*, vol. 8, no. 5, pp. 481–494, 2002.

[9] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *Proceedings of INFOCOM*, 2002.

[10] T. van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proceedings of SenSys*, 2003.

[11] W. Ye, F. Silva, and J. Heidemann, "Ultra-low duty cycle mac with scheduled channel polling," in *Proceedings of SenSys*, 2006.

[12] T. Cui, L. Chen, and T. Ho, "Energy efficient opportunistic network coding for wireless networks," in *Proceedings of INFOCOM*, 2008.

[13] J. Zhang, Y. Chen, and I. Marsic, "Network coding via opportunistic forwarding in wireless mesh networks," in *Proceedings of WCNC*, 2008.

[14] J. Goseling, R. Boucherie, and J.-K. van Ommeren, "Energy consumption in coded queues for wireless information exchange," in *Proceedings of Network Coding, Theory, and Applications*, 2009.