ON COMBINING DUTY-CYCLING WITH NETWORK CODING IN FLOOD-BASED SENSOR NETWORKS

A Thesis

by

ROJA RAMANI CHANDANALA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Computer Science

ON COMBINING DUTY-CYCLING WITH NETWORK CODING IN FLOOD-BASED SENSOR NETWORKS

A Thesis

by

ROJA RAMANI CHANDANALA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Computer Science

ABSTRACT

On Combining Duty-cycling with Network Coding in Flood-based Sensor Networks. (December 2010)

Roja Ramani Chandanala, B.Tech, National Institute of Technology, Warangal, India

Chair of Advisory Committee: Dr. Radu Stoleru

Network coding and duty-cycling are two popular 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. One explanation is that these two 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 thesis, we thoroughly 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. Additionally, a novel, efficient coding scheme decision algorithm, ECSDT, assists DutyCode to reduce further energy consumption by minimizing redundant packet transmissions, while an adaptive mode switching algorithm allows smooth and timely transition between DutyCode and the default MAC protocol, without any packet loss. We investigate our solution analytically, implement it on mote hardware, and evaluate it in a 42-node indoor testbed. Performance evaluation results show that our scheme saves 30-46% more energy than solutions that use network coding, without using duty-cycling.

To my parents,

To my teachers,

and

To my friends

ACKNOWLEDGMENTS

I owe my deepest gratitude to my advisor Dr. Radu Stoleru, without whom this thesis would not have been possible. He was always accessible and willing to help his students with their research. His consistent guidance, perpetual enthusiasm in research and intuitive thinking kept me inspired and motivated. I would like to thank him for accepting me into LENSS Lab, which provided me with needed equipment for my research and exposure to the field of sensor networks. I am grateful to Dr. Alex Sprintson, Dr. Andrew Jiang and Dr. Jennifer L. Welch for being part of my committee. Dr. Ghassemi Ahmad deserves a special thank you for hiring me as a research assistant in the PE Rock Mechanics Laboratory. The experience broadened my horizon and my perspective immeasurably.

All my lab mates were friendly and made it a great place to work, I would like to thank all my lab mates, especially Manish Kumar, Mike George and Myounggyu Won, for their support and help in the completion of this project. This work would have been an unfulfilled dream without their constant feedback and help.

I am indebted to my father, Yadagiri and my mother Ramamani. I wish to thank my siblings, especially Shankar and Ravindranath for their invaluable love, support and encouragement. Without them, this would not have been possible. Last, but not the least, I would like to acknowledge my friends, especially Bharath, Geetha, Pranitha and Anurag for making my stay in College Station cherishable and enjoyable.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER 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 multicast, 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. 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 integrate it.

Given the importance of duty cycling to WSN, the assumption that nodes will

---

This thesis follows the style of *IEEE/ACM Transactions on Networking.*

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 fixed intervals as short as 3msec result in increased energy consumption and delay instead of shrinkage.

In this thesis, 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. To the best of our knowledge, this is the first work that considers both duty cycling and network coding. 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 requires 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 Random Low-Power Listening (RLPL), the new MAC layer, facilitates streaming, elastic random sleeping (ERS) 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 pre-requisite of our solution is that network coding be applied individually to a sequence of packets, called a "page" (also known as a "generation"). The packets that are to be coded within a page are random.

Redundant packet transmissions can be reduced by selecting appropriate coding schemes for nodes. We propose a novel efficient coding scheme decision algorithm "ECSDT" that computes coding scheme for each node that minimizes the extraneous packet transmissions. ECSDT is integrated with DutyCode and achieves more energy savings. In addition, a novel technique ensuring the transitions between LPL, a normal duty-cycling protocol, and RLPL, our proposed protocol, happen in a smooth and timely manner is devised in an attempt to avoid packet loss during the transitions.

The contributions of this thesis 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.
- ECSDT, a new efficient coding scheme decision mechanism for any static network topology. This assigns coding schemes to minimize the number of transmissions and make the solution more energy efficient.
- A complete adaptive solution allowing the application to smoothly switch from LPL to RLPL based on message traffic.
- Theoretical analysis and extensive simulations demonstrating the energy efficiency and higher throughput of this solution.
- An implementation on mote hardware and performance evaluation in 42-node testbed where actual energy consumption is measured.

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

CHAPTER II

STATE OF ART

Energy-efficiency in WSN is an area of active research. Multipath routing schemes for energy-efficiency transmissions are examined in [2]. Error correction increases packet delivery rates and decreases retransmissions in [3] [4]. Peripheral driver optimizations are proposed in [5]. However, two promising directions for increasing energy-efficiency include duty cycling and network coding.

In the area of duty cycling, research has often examined low power listening (LPL) and scheduling. Whereas LPL protocols demand the use of long preambles, scheduling protocols require periodic transmission of control packets for synchronization. B-MAC [6] is a simple LPL protocol with periodic listening that requires no synchronization. However, in high traffic networks, either throughput or sleep is impacted because of overhearing caused by long preambles. X-MAC [7] improves B-MAC with the help of short preambles and acknowledgements thus minimizing the overhearing problem, but suffers similar inefficiencies in networks using broadcast messages. 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. Efficient at low bandwidth, performance degrades at higher network loads because of fixed duty cycling and adaptation to neighbors' schedules. SCP [11] saves power by scheduling coordinated transmission and listen periods. However, high network loads reduce sleep opportunities. T-MAC [12] enhances S-MAC by reducing the awake period even more. However, nodes frequently miss useful packets while asleep. A-MAC [13] introduces advertisement window to provide prior knowledge of the future transmissions to nodes. However, scheduling through advertisement is not a feasible solution for broadcast applications with higher network loads. DW-MAC [14] is another scheduling protocol that allows nodes to wake up on demand. AS-MAC [15] 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 [16] is a receiver initiated MAC protocol with an aim to reduce the idle-listening. But, scheduling algorithms do not apply for broadcast applications.

Broadcasting in low duty-cycling networks has similarities to the problem addressed in this thesis. Because in flood-based network coding applications typically all messages are broadcasted. The Sleep and Awake durations for each node are computed as a optimization problem for unicast transmissions [17]. Opportunistic flooding [18] and Schm-Dist [19] save energy in a low duty-cycling networks by treating broadcast transmissions as unicasts. Opportunistic flooding [18] utilizes probabilistic flooding based on the delay distribution of neighbors. Hong, et.al. [19] prove that broadcasting in a low duty-cycling network is an NP-hard problem and provide approximation algorithms based on top-down layered approach and D2-coloring solution. OTAB [20] is a centralized approximation algorithm for duty-cycle aware minimum latency broadcast scheduling. ADB [21] 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 [22], Cui, et.al. maximize throughput by combining several unicast packets into a single broadcast packet. BEND [23] 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 [24] 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 [25]. Decreased errors contribute to higher throughput and reduced power consumption in [26]. Multimedia throughput and energy-efficiency in wireless networks is examined in [27]. However, existing coding schemes did not consider the duty cycling into consideration. Many applications in wireless sensor networks adapted network coding for efficiency. AdapCode [28] is a code dissemination protocol that incorporates network coding. When combined with a low-power-listening MAC protocol, packets are forwarded between asynchronized sleep periods using network coding to reduce total transmission. However, inflexible scheduling increases energy use. Widmer, et.al. [29] proposed a new flooding application that uses network coding for energy efficiency in extreme networks. Packets are divided into generations in [29] which is similar to pages in AdapCode [28]. CODEB [30] uses Reed-Solomon based coding algorithm for achieving optimal coding. ReedSolomon (RS) codes are non-binary cyclic error-correcting codes invented by Irving S. Reed and Gustave Solomon [31]. Cluster based network coding scheme is proposed in [32], to minimize the redundancy in messages transmitted. But these schemes are designed for unicast message patterns.

In the past few years, dynamic MAC protocols that modify adapt based on the traffic pattern has been the prime topic of research. TEEM [33], MaxMAC [34] and BEAM [35] are traffic aware MAC protocols in WSN. TEEM [33] builds traffic awareness on top of S-MAC. MaxMAC [34] uses features of WiseMAC and XMAC for energy efficiency. BEAM [35] uses short preambles and long preambles based on the traffic patterns and employs schedule adaptation for energy efficiency. However all these protocols deal with unicast messages and are not suitable for flood-based applications. P-MAC [36] may not scale to a large, message-intense network, as it requires periodic traffic pattern update to achieve traffic aware duty-cycling.

In [37], we proposed a network coding protocol that achieves energy saving of 20-30% in duty-cycled wireless sensor networks. This thesis improves the energy efficiency of the protocol by proposing a coding decision algorithm, ECSDT, that minimizes the redundant packet transmissions, thereby saving more energy. Furthermore, a sophisticated adaptive transition technique accomplishes a smooth and timely transition between LPL mode and RLPL mode without any packet loss. The effectiveness of the new algorithms is proven by extensive experiments on real hardware.

CHAPTER III

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. 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 r_2, as shown in Figures 1(a) and 1(b). A total of 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_2, t_1 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 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 $(c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4)$. 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 losing a single packet increases linearly.

Most existing duty-cycling protocols achieve energy savings through scheduling or low power listening (LPL) [6]. However, both scheduling and LPL are not feasible solutions for network coding, since overhearing, the fundamental building block of

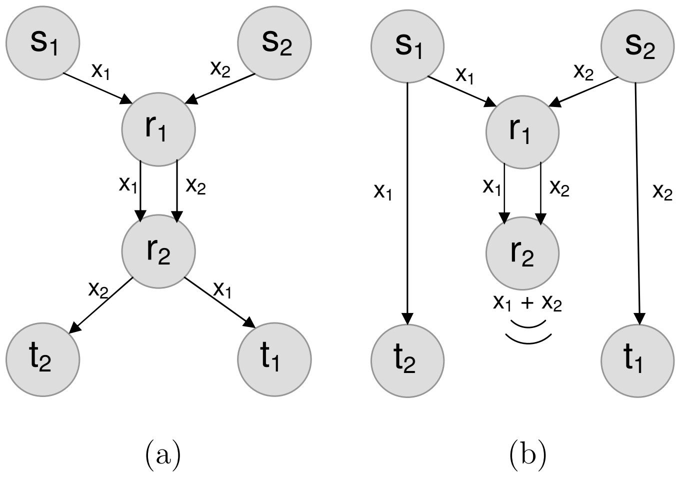

Fig. 1. a) Transmissions without network coding; b) with network coding.

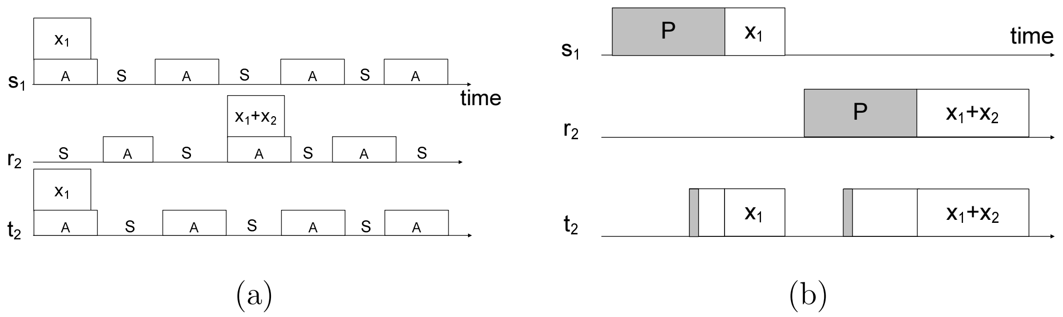

Fig. 2. Network coding integrated with two major categories of duty-cycling protocols: (a) Scheduling based ("A" and "S" represent active and sleep states respectively), and (b) Low power listening based on long preambles (P represents a Preamble).

network coding, is difficult to achieve when those techniques are employed. If network coding is used with such duty cycling protocols, the probability of losing useful packets will increase by failing to overhear packets. Consequently, system performance such as energy consumption and code dissemination time will be impacted. Figure 2(a) illustrates the scenario where scheduling is employed in the network coding example shown in Figure 1(b), and Figure 2(b) shows the scenario when LPL is applied. In Figure 2(a), t_2 misses the coded packet x_1+x_2 due to scheduled sleep time. Figure 2(b)

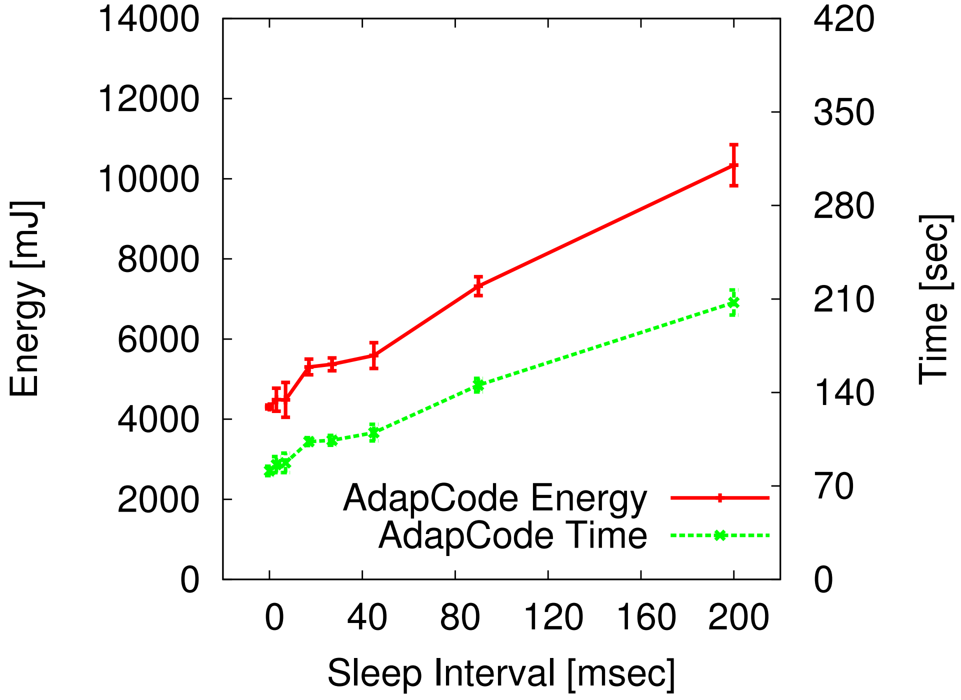

Fig. 3. Execution of X-MAC-enabled AdapCode on a 42 epic mote testbed.

shows an increase in total execution time caused by long preambles used for LPL.

To validate our intuition, we performed experiments on a testbed of 42 Epic motes. We integrated AdapCode [28], a code dissemination application that uses network coding, with X-MAC [7], 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 delay 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.

During these experiments, a significant number of redundant packet transmissions was detected. This redundancy in AdapCode could be attributed to its simplified assumption that the network is uniform, where the number of neighbors is assumed to be the same for all nodes. However, uniform network is not a practical assumption 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 possible packet transmissions.

When code download is not taking place, LPL mode is a better solution in terms of energy saving, since when network traffic is low, there are more opportunities of making nodes to sleep. Thus, a technique ensuring adaptive transitions between LPL mode and RLPL mode based on the traffic is required. Such technique also needs to provide smooth and timely transition to avoid packet loss during the transitions.

CHAPTER IV

DUTY CYCLING OF NETWORK CODED 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 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. Each packet of the stream has an additional control field indicating the number of remaining packets of the stream. A node can compute the duration of stream as the transmission time per packet times the remaining packets of stream.

4.1 Preliminaries

In this section we introduce a typical application, called NetCode, that uses network coding for energy efficiency. NetCode is a generic representation of flooding applications that use network coding. AdapCode [28] is an instance of NetCode. 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, it broadcasts the data as pages. Each page consists of a number of

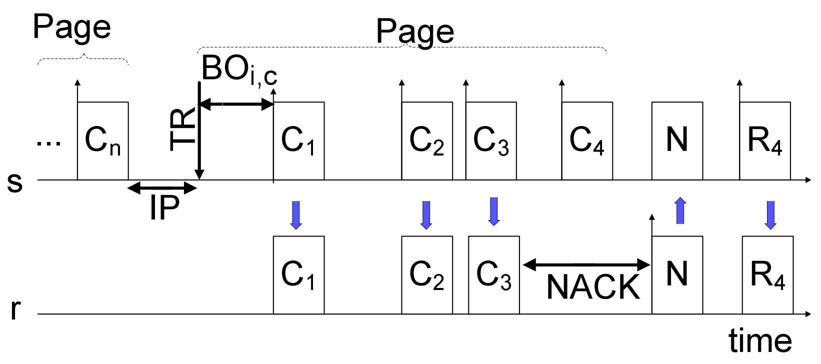

Fig. 4. The NetCode protocol: C_i are coded packets, N is a NACK packet, due to missing packet C_4, R_4 is the ReNACK packet - the non coded packet missing, IP is the Inter-page interval, $BO_{i,c}$ is the backoff interval - initial and congestion, NACK is a timer.

packets. After transmitting a page, the source waits for a short period of time for code propagation, and subsequently sends the next page. The time interval that separates two pages is called "*Inter-page interval*" (IP in Figure 4). The source maintains a constant small delay between any two packets of a page. The "*Transmission Request*" (TR in Figure 4), is the transmission request for a packet transmission. In Figure 4, the TR only for the coded packet C_1 is shown to keep the figure simple. 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 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 4). As with existing CSMA

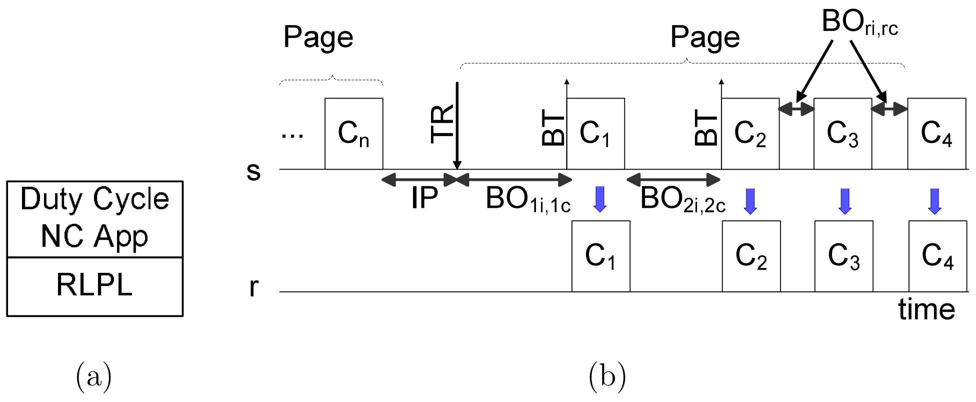

Fig. 5. (a) DutyCode architecture: (b) Streaming in DutyCode: after the backoff intervals, $\mathrm{BO}_{1i,1c}$ and $\mathrm{BO}_{2i,2c}$ coded packets in a page are streamed.

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 in Figure 4) from BO_i and BO_c, respectively.

4.2 Proposed Solution: DutyCode

Our solution, called DutyCode, is shown in Figure 5(a). DutyCode is an integrated scheme (MAC and Network coding application) in which the MAC layer facilitates streaming, random sleeping and synchronization, while the application layer 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.

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 "Sleep Interval" as the duration per packet, for which a node sleeps upon receiving a packet from a useless stream.

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 ($\mathrm{BO}_{1i,1c}$ and $\mathrm{BO}_{2i,2c}$) chosen from different ranges and the rest of the stream is sent with very small and fixed backoff interval $\mathrm{BO}_{ri,rc}$. 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}). The application can set these values according to the reliability of the network.

Upon receiving a stream packet, a node yields if it has no unfinished stream transmission and no packet awaiting transmission. As shown in Figure 6 upon receiving a stream from node s, nodes r_1 and r_2 decides to yield to the stream from s (red color arrows) and they 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 of the stream. After r_1 wakes up, it

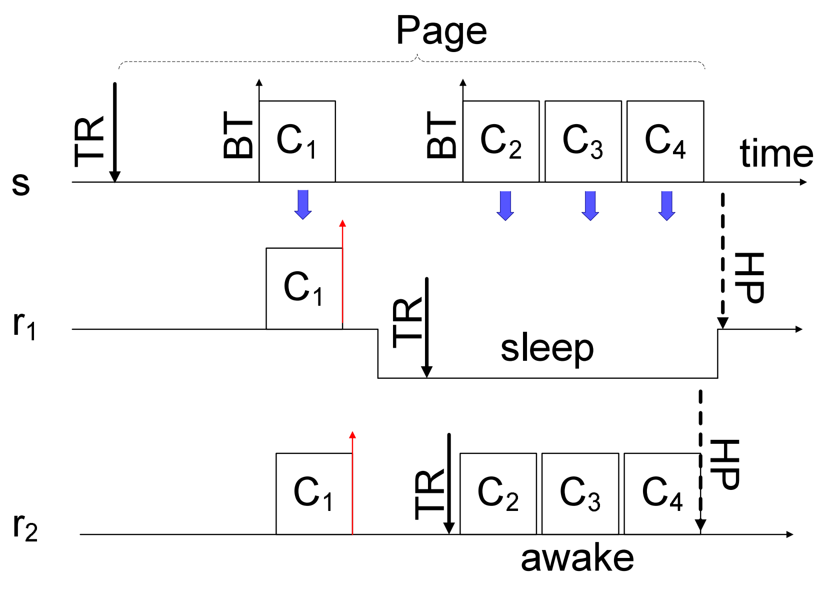

Fig. 6. Streaming with no packets awaiting transmission and no unfinished streams at nodes r_1 and r_2.

tries to transmit any pending packet. Because node r_2 does not have the page being transmitted, it 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 (HP in Figure 6). In Figures 6 through 10, TR indicates a transmission request from the NC application; BT indicates the backoff timer fire event; HD indicates the handling of deferred packet and HP indicates the handling of pending transmissions.

Transmission Defer. A transmission defer is a decision made by MAC layer to postpone a packet transmission for a future time, if feasible. A node defers its transmission when it decides to yield to an existing stream from another node. In Figure 7, nodes r_1 and r_2 yield when they have no unfinished stream, 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 resume the deferred packet transmission when they

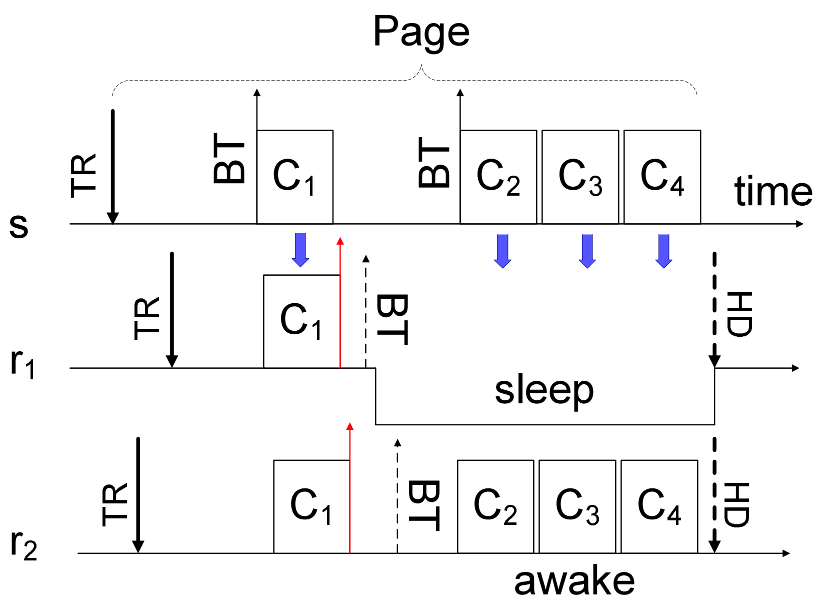

Fig. 7. Streaming with transmission defer (red color arrows indicate the defer of the current transmission) with no unfinished streams at nodes r_1 and r_2.

realize that the stream transmission by node s is over. A deferred packet is handled as a new transmission request i.e., the node backs off for the duration selected randomly from the 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.

Transmission Arbitration. Transmission arbitration happens when two nodes attempt to transmit a packet from an unfinished stream at the same time. Unique node ID is used to determine who will transmit first. Specifically, a node with larger node ID will have a higher priority. However, constantly giving a priority to nodes with larger IDs might result in starvation of nodes with smaller IDs. To compensate for this unfairness, $\mathrm{BO}_{ri,rc}$ is determined based on node ID, and nodes with smaller IDs are given smaller backoff interval, allowing more chances to transmit. Figures 8, 9, and 10 illustrate different transmission arbitration scenarios where nodes s and r_1 ($s > r_1$) compete for the channel. In Figure 8, r_1 learns about the stream from s, and

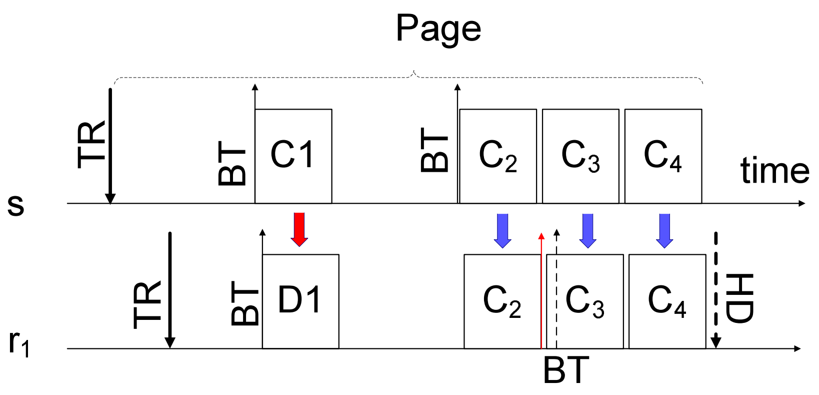

Fig. 8. Transmit arbitration: $s > r_1$ and r_1 learns about the stream from s and successfully defers its transmission.

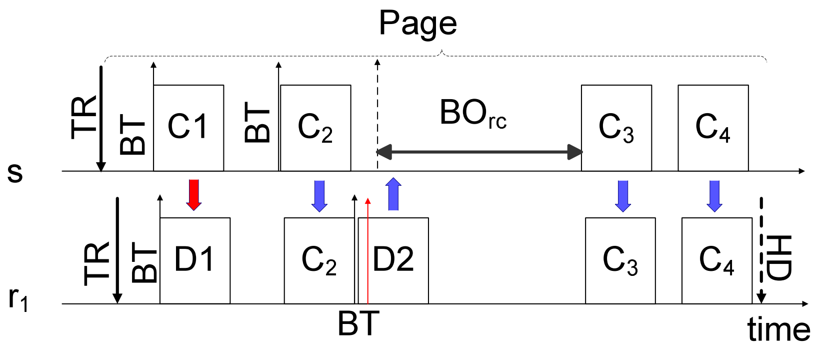

Fig. 9. Transmit arbitration: $s > r_1$ and r_1 learns about the stream from s and fails to defer its transmission.

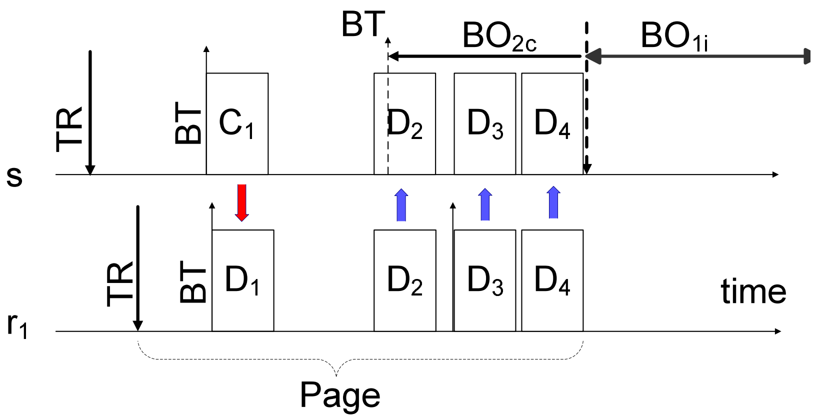

Fig. 10. Transmit arbitration: $s > r_1$ and s learns about the stream from r_1 first.

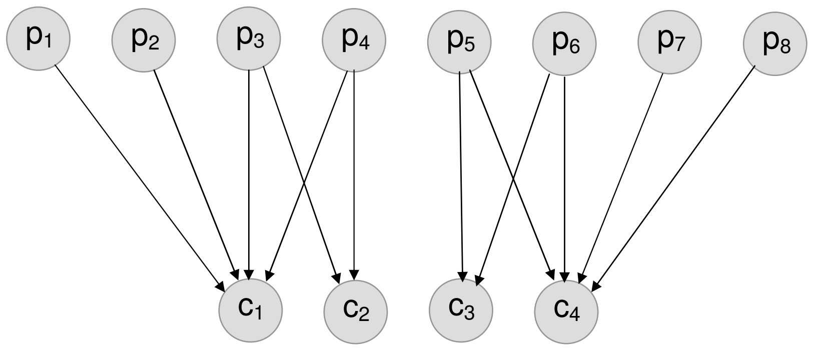

Fig. 11. An example of a network topology that causes redundant packet transmissions.

yields to s by successfully deferring its transmission. Figure 9 presents the case where r_1 learns about the stream from s but fails to defer. Thus, s sees the channel is busy. In this case, s waits for BO_{rc}, after which s resumes its stream. Figure 10 shows a different scenario, where s learns about the stream from r_1. In this case s waits for BO_{2c}, and after receiving the last packet from the stream of r_1, s tries to transmit the remaining stream as a new stream.

4.3 Coding Enhancement

While the solution proposed in the previous section saves a considerable amount of energy, there is still an opportunity to save more energy, i.e., by minimizing the number of unnecessary packet transmissions. To illustrate such redundant packet transmissions, Figure 11 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. We first easily note that leaf nodes ($c_1 - c_4$) do not need to send coded packets. In order to analyze the unnecessary packet transmissions, we define PCS (Preferred Coding Scheme) for each node. PCS is the maximum coding scheme that a node's parents can have such that all the children of the parent can successfully decode all the encoded packets. As an example, consider the PCS value for c_1. Since c_1 receives 4 coded packets, each from its 4 parents, $p_1 - p_4$, at most 4 packets can be encoded into a single packet, yielding a PCS value of 4. In a similar way, the PCS values for c_2, c_3, and c_4 are 2, 2, and 4 respectively. A critical observation is that any coded packet transmissions from p_1, p_2, p_7, and p_8 are useless, because the coding scheme of $p_3 - p_6$ must be 2, and all children $c_1 - c_4$ can decode all the packets from the coded packet transmission of $p_3 - p_6$.

In order to avoid such extraneous transmissions and save extra energy, we propose ECSDT, a new efficient coding scheme decision mechanism for any static network topology. The static network topology is represented 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 is associated with a link quality (LQ) which represents the successful packet delivery ratio. Only the edges with LQ greater than a pre-defined link quality threshold (LQT) are considered. A different LQT results in different topologies, affecting the performance of ECSDT. We let $P_i = \{n_j : (n_j, n_i) \in E\}$ be the set of all one-hop parents of node n_i and let $C_i = \{n_j : (n_i, n_j) \in E\}$ be the set of all one-hop children of node n_i. We denote by $n_{i,j}^{PCS}$ the PCS value of n_i for its parent $n_j \in P_i$, and by n_i^{CS} the coding scheme of n_i.

The ECSDT algorithm is depicted in Algorithm 1. The ECSDT runs in 2 phases. In the first phase, each node n_i 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 determined to be the $min\{n_{j,i}^{PCS} : n_j \in C_i\}$, i.e., the minimum PCS value among all the PCS values of its children (Line 4-6). If $|C_i| = 0$, "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, each node n_i checks for any possible redundant transmissions by examining the initial coding schemes of its parents. If

Algorithm 1 ECSDT: Assigning Coding Schemes

```
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:     $n_i^{CS} \leftarrow min\{n_{i,j}^{PCS} : n_j \in C_i\}$
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 all 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:        $n_i^{CS} \leftarrow min\{n_{j,i}^{PCS} : n_j \in C_i\}$
22:    end if
23: end for
```

node n_i finds a possible redundant transmission from its parent n_j, it updates the PCS value for the parent, $n_{i,j}^{PCS}$ to null coding scheme (Line 7-16). As the last step, each node checks the PCS value of its children. If all children suggest a null coding scheme, the parent sets its final coding scheme to the null coding scheme; otherwise to the minimum PCS value among all the PCS values of its children (Line 17-23).

We present a proof-of-concept simulation result that proves the correctness of the proposed algorithm. A simple simulator written in JAVA generates a random topology with 16 to 100 nodes, in which one node is chosen as a source and starts a code update. We measured the total number of transmitted packets during the code update for both AdapCode and ECDST. In AdapCode, a node determines the coding scheme based on the number of neighbors from which it received useful packets. Thus, one-hop parents and peer nodes (the nodes that are in the same hops away from the source) can be its neighbors. In this simulation, both cases, when the peer nodes are counted as neighbors and when not, are considered. Figure 12 depicts the results. Compared with the AdapCode result with peer nodes being counted as neighbors (The line "Adapcode+Peer nodes" in Figure 12), ECSDT shows more than a few magnitude less packet transmissions, and compared with the AdapCode without the peer nodes being counted as neighbors, ECSDT shows 50% less packet transmissions.

Although ECSDT is a centralized algorithm where coding schemes are decided at a central entity, it can be modified to run in a distributed manner through some additional message exchanges between nodes. Also, it can be adapted to dynamic environments to take link failures into account by piggy-backing feedback related to coding schemes in NACKS.

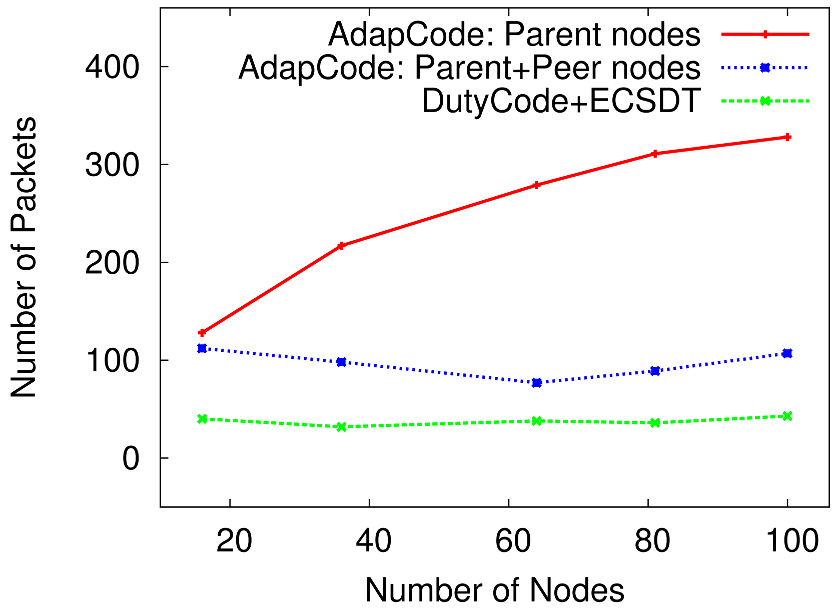

Fig. 12. The proof-of-concept result showing the effectiveness of the ECSDT algorithm.

4.4 LPL/RPLP Mode Transitioning

Leveraging data streaming, aggressive energy saving can be achieved using RLPL when a code update is underway. However, this is not the case when the network traffic is low, typically after a code update is finished. In that case, LPL becomes more efficient than RLPL. Thus, there is a need for an efficient technique to switch between the two modes. Such switching technique needs to be carefully designed to ensure a smooth timely transition with no packet loss.

In our solution, each node starts with LPL mode, and when it receives the first packet of code update, it attempts to switch modes (this is illustrated in Figure 13) To minimize packet loss during mode transition from LPL to RLPL, we use a transient mode, called NoSleepLPL, instead of directly switching mode from LPL to RLPL. In NoSleepLPL mode, after receiving the first packet of code update, a node does not sleep trying not to miss any packets, and received packets are relayed utilizing

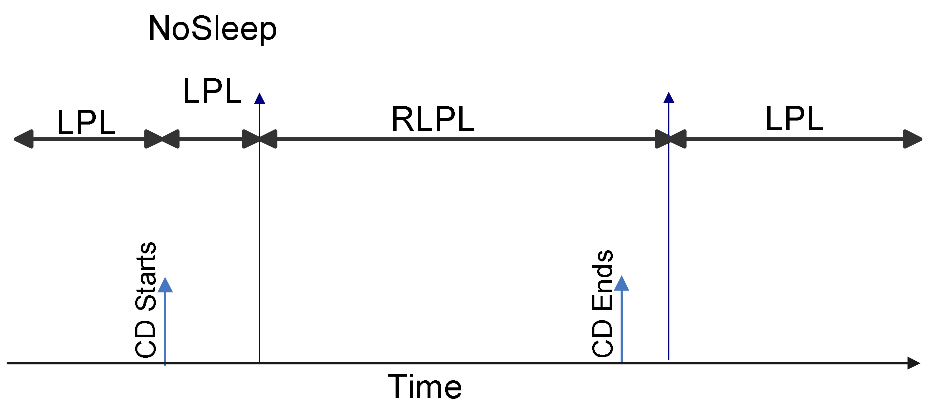

Fig. 13. Protocol transition; "CD starts" indicates the beginning of code download and "CD Ends" indicates when a node receives all expected code update packets.

long preambles to ensure that its children, in turn, do not miss the relayed packets and go into NoSleepLPL successfully. At a fixed time interval after receiving the first packet of code download, a node switches its mode from NoSleepLPL to RLPL. RLPL mode is switched back to LPL mode when a node has received all the expected coded packets. Switching back to LPL mode is relatively easier due to low traffic, without incurring any issues of packet loss.

CHAPTER V

DUTYCODE PROTOCOL ANALYSIS

In this chapter we analyze the operation of the proposed DutyCode scheme. The aim of our analysis is as follows:

- 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. The two metrics we investigate are the total number of packets per page, and the total execution time.
- To show that the coding schemes assigned by ECSDT are optimal.
- To analytically evaluate the DutyCode protocol and compute an upper bound on the energy savings.

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 chapter), 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):

$$SI \cdot pp/cs \leq (BO_{1i}/2 + pp/cs \cdot t_{tr}) \tag{5.1}$$

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.

5.1 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: a) 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:*

$$C_p^a = C_p^d \tag{5.2}$$

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* $k_1 = k_2$.

$$\frac{cp_1}{cp} = \frac{BO_c}{BO_{1i}} \tag{5.3}$$

If backoff intervals are chosen such that $BO_{1i} \geq BO_c$, *then, from Equation 5.3:*

$$cp_1 \leq cp \tag{5.4}$$

From Equation 5.4, when $pp/cs \geq (1 + cp_2)$ *and* $BO_{1i} \geq BO_c$ *then:*

$$N_p^d \leq N_p^a \tag{5.5}$$

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 regular packets with no coding.*

In NetCode, if there is a collision while transmitting a coded packet, the application may need to send all the packets that are coded into the message, individually and not coded. 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:* $R^a_p = cs \cdot pp/cs \cdot (cp + 2cp^2 + ...)$.

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:*

$$\begin{aligned} R_p^d &= cp_1 \cdot pp + 2cp_1^2 \cdot pp + 3cp_1^3 \cdot pp + ... \\ &= pp \cdot (cp_1 + 2cp_1^2 + ...) \end{aligned}$$

From Equation 5.4, when $BO_{1i} \geq BO_c$ then:

$$R_p^d \leq R_p^a \tag{5.6}$$

From Equations 5.2, 5.5 and 5.6, and since $P_p^{a/d} = C_p^{a/d} + N_p^{a/d} + R_p^{a/d}$, then $P_p^d \leq P_p^a$.

5.2 Total Execution Time

Theorem 2 *If the backoff intervals satisfy $pp/cs \cdot BO_c \geq BO_{1i}$ 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:*

$$P^d = s \cdot pp/cs + N^d \tag{5.7}$$

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

$$P^a = s \cdot pp/cs + N^a \tag{5.8}$$

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 5.8, we obtain:

$$T_n^a = s \cdot pp/cs \cdot BO_c/2 + N^a \cdot BO_c/2 + P^a \cdot D_t^a + P^a \cdot t_{tr} \tag{5.9}$$

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 $BO_{1c}/2$. D_t^d is the average time delay in DutyCode, similar to D_t^a ($D_t^d << D_t^a$). In DutyCode, based on Equation 5.1, the total time for code download is the total time required for all packet transmissions. Consequently the total time per node is:

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

since $BO_{1c} = BO_c$.

The above equation 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.

From Equation 5.5 and Theorem 4.1 and when $pp/cs \cdot BO_c \geq BO_{1i}$ is satisfied then:

$$T_n^d \leq T_n^a \tag{5.10}$$

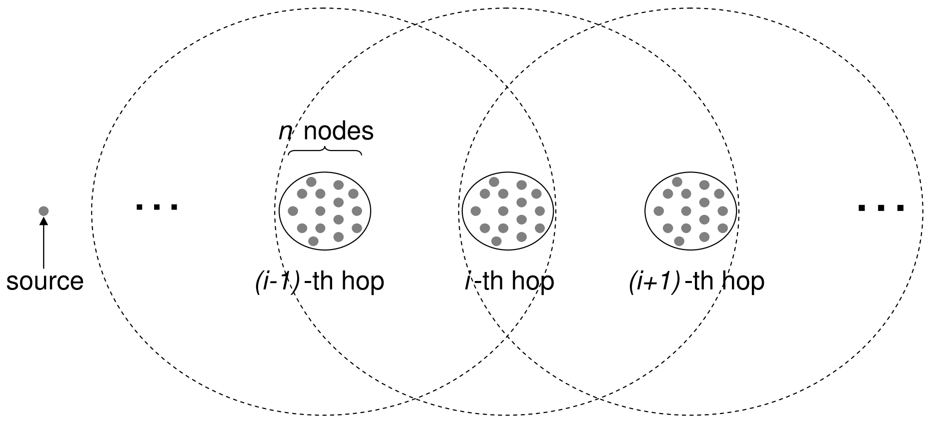

Fig. 14. A multihop network topology for a flood-based wireless sensor network application.

5.3 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 14, where any nodes in i-th hop are within the interference range of the nodes in $(i+1)$-th hop and $(i-1)$-th hop. We assume that the Inter-page interval (IP) is fixed to minimal, i.e., the source sends the next page after the previous page has been successfully downloaded from all the nodes in 3 hops from the source, 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:

$$E_{saving} = T_{sleep}/T_{total} \tag{5.11}$$

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

Since the code download is a pipelined process, given the minimal IP interval, T_{total} can be measured as the total time taken for any nodes in S_{i-1}, S_i, and S_{i+1}

to finish the code download. Consider a node $n_i \in S_i$. In this topology, n_i cannot transmit a packet if any nodes in S_{i-1} (parents), S_i (peers), or S_{i+1} (children) are transmitting packets. Thus, the expected total code download time $E(T_{total})$ is given as,

$$E(T_{total}) = \left(\frac{|S_{i-1}| + |S_i - 1| + |S_{i+1}|}{2} \right) \cdot T_{page} \cdot P$$

$$= \frac{3n}{2} \cdot T_{page} \cdot P \tag{5.12}$$

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.

If the coding scheme is implemented properly, only packets transmitted by the nodes in S_{i-1} are useful. So, 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 given as follows:

$$E(T_{sleep}) = \left(\frac{|S_i - 1| + |S_{i+1}|}{2} \right) \cdot T'_{page} \cdot P$$

$$= n \cdot T'_{page} \cdot P \tag{5.13}$$

where T'_{page} is the sleep time for a page, i.e., $T'_{page} = SI \cdot pp/cs$.

From Equation 5.1,

$$2n \cdot SI \cdot pp/cs \leq 2n(BO_{1i}/2 + pp/cs \cdot t_{tr}). \tag{5.14}$$

The maximum energy saving is achieved when

$$SI \cdot pp/cs = (BO_{1i}/2 + pp/cs \cdot t_{tr}) \tag{5.15}$$

Thus,

$$E(E_{saving}) = E\left(\frac{T_{sleep}}{T_{total}}\right)$$
$$= \frac{n \cdot (SI \cdot pp/cs)}{\frac{3n}{2} \cdot (BO_{1i}/2 + pp/cs \cdot t_{tr})} \leq \frac{2}{3} \approx 66\%. \tag{5.16}$$

5.4 Coding Scheme Enhancement

This section proves that the coding schemes obtained from ECSDT are optimal.

Theorem 3 *ECSDT outputs the optimal coding schemes for all nodes to decode all the packets successfully.*

Proof 3 *Assume in contradiction that there exists a better coding scheme A. This implies that in A, there exists a node that transmits fewer packets than that of ECSDT. Let such node be n_i. Note that, in ECSDT, each parent chooses the minimum of the coding schemes 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. And so, A does not exist.*

CHAPTER VI

IMPLEMENTATION

We implemented the DutyCode protocol in nesC for TinyOS 2.1.0. The implementation was done in the CC2420ReceiveC (Receive), CC2420TransmitC (Transmit) and CC2420CsmaC (Csma) modules. New modules RandomLPL (RLPL), RPowerCycleC(Power) were created, which differ from the existing DefaultLPL and PowerCycle respectively, as presented in Chapter IV. The implementation changes can be broadly classified based on the two basic aspects of the DutyCode protocol: i) Packet Streaming ii) Elastic and Random Sleeping (ERS).

6.1 Packet Streaming

Transmit, Receive and RLPL modules are modified to achieve packet streaming. When streaming is achieved, the application sends packets one after the other without any significant delay (i.e., as soon as sendDone() is signaled). Each packet header contains the number of remaining packets in the stream, computed based on the coding scheme used.

Streaming Packet Received. The Receive module notifies the Transmit module when it receives a stream packet from other nodes. Upon receiving a stream 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 Chapter IV (Line 2). If not, no action is taken. Otherwise, if it is in the middle of transmission, the node tries to defer the packet transmission and informs

Algorithm 2 Transmit: Streaming Packet Received (From Receive Module)

```
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
```

RLPL about: i) the details of the stream transmission; and ii) the defer status, if there was a need for packet defer (Line 4-5). RLPL then sets a NoSend timer (Line 7) and keeps future transmit requests 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).

We implemented the packet defer in the Transmit module because it is the only module that maintains the transmission internal state, and because Transmit has to be informed at the earliest time so that it can defer the transmission if possible.Earlier our solution included extended backoff intervals but if there is any pending packet transmission, RLPL does not turn off the radio. (This check is validated even in the DefaultLPL, and this check is needed to ensure that the radio would not be turned off in middle of transmission.)

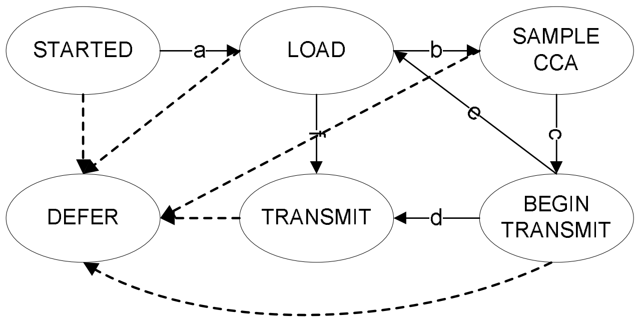

Fig. 15. State transition diagram for the transmit module, with changes (dotted arrows) for the "packet transmission defer". Other transitions: (a) transmission request; (b) copy packet on radio stack; (c) perform CCA; (d) channel clear, transmit; (e) congestion backoff; (f) no CCA requested, transmit.

Packet Ready for Transmission. 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 and waits for the backoff interval corresponding to the CCA result (transitions a-b-c-d in Figure 15). Transmit can defer a packet transmission until the actual transmission has been started (dotted arrows in Figure 15). The application would not be aware about this packet defer and RLPL handles the deferred packet as soon as possible.

6.2 Elastic and Random Sleeping (ERS)

This section presents the implementation changes done for ERS. RLPL and Power modules are modified such that sleep requests are no longer handled periodically (as done for LPL). Instead, they are treated as one time requests. The Power module is also modified to not perform clear channel assessment (CCA) before turning off the radio. Csma is modified to turn off the radio only if Transmit decides to defer the transmission, when the sleep request arrives while it is in the middle of packet transmission.

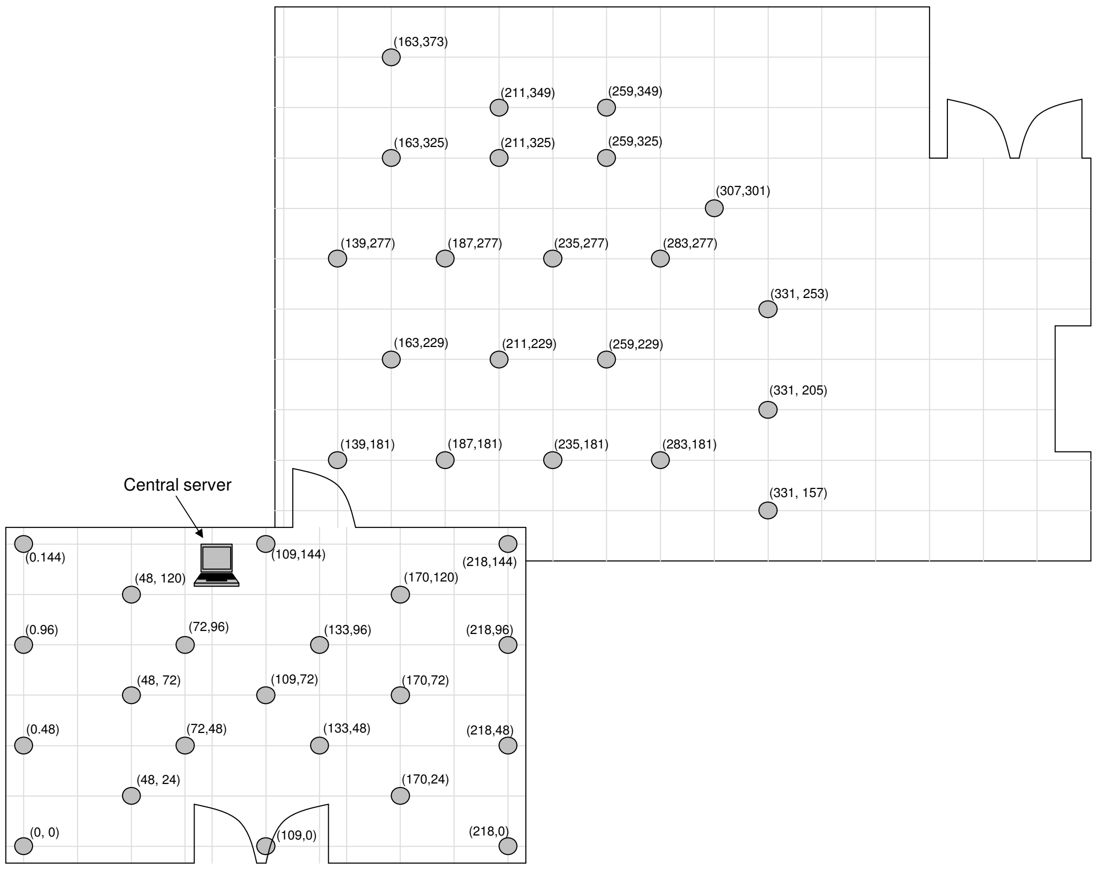

Fig. 16. The map of our testbed. (x, y) represents the relative coordinate of a node in inches with the node at the lower left corner as origin.

6.3 The ECSDT Algorithm

The ECSDT algorithm is implemented in JAVA and deployed on a central server which is shown in Figure 16. It communicates with all 42 motes of our testbed via serial ports. When the algorithm starts, the network topology is constructed by receiving neighbor tables from the motes. Using the network topology, the ECSDT then computes an appropriate coding scheme for each node as described in Section 4.3. The new coding schemes are transmitted back to the nodes via serial ports. When the code download starts, each mote uses the new coding scheme.

6.4 LPL to RLPL Mode Transition

In order to achieve a smooth transition between LPL mode and RLPL mode (as described in Section 4.4) two independent MAC protocol modules, one with LPL and the other with RLPL, are built, and a new wrapper module is created to serve as a common interface to the underlying MAC protocols. The wrapper forwards the function calls to the relevant MAC function based on the current mode selected by the application.

The wrapper handles the mode transition smoothly: if the current MAC protocol is in the middle of transmission, the transition happens after receiving the SendDone signal. 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 4.4. The noSleepLPL is implemented such that the DefaultLPL and Power-Cycle modules are modified not to turn off the radio when requested by an application through Wrapper. In order to fit the two MAC layers in the 10KB RAM of the epic motes, the LPL and RLPL are designed to share modules that are common to both protocols.

CHAPTER VII

PERFORMANCE EVALUATION

We evaluate the performance of DutyCode in an indoor testbed consisting of 42 Epic motes [38] deployed in an approximately 500ft^2 area. Out of the 42 nodes, 14 are instrumented for power consumption measurements. Different TX power is obtained by changing the TXCTRL.PA_LEVEL register of the CC2420 transceiver [39]. Experiments are performed in a 5 hop network, obtained by setting the TX power of each node to 4. Each experimental point represents the mean of 5 executions of the protocol. Standard deviation is depicted in all performance evaluation results.

For comparison with state of art we chose AdapCode [28], a flooding protocol that uses network coding and is representative of our NetCode model. The metrics used for performance evaluation are per node energy consumption and total code dissemination time. While we are interested in energy consumption, we also aim to not increase the total download time. The parameters that we vary are sleep interval (SI), node density (ND), the size of packet (SP), the number of packets (NP), NACK delay ($NACKD$), and inter-page interval (II). From Theorems 4.1 and 4.2, the BO$_{1i}$ value 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. Default values for the parameters are: SI=17msec, ND=4, SP=28bytes, NP=256, $NACKD$=640msec, II=300msec (from here on units of measure will be omitted). The effects of these parameters are investigated in the remaining part of this Chapter.

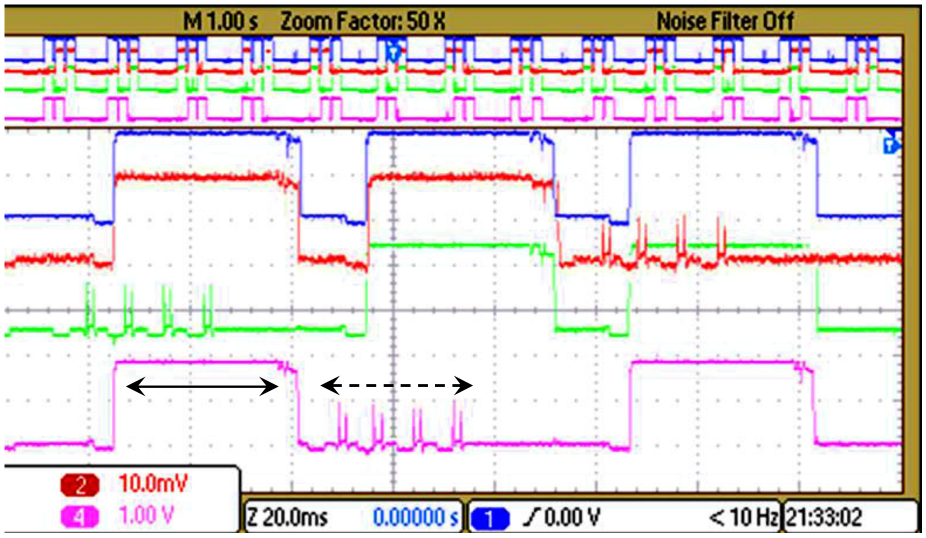

Fig. 17. Oscilloscope view of streaming. Sleeping is indicated by solid arrows and packet streams are denoted by dotted arrows.

7.1 Preliminary Evaluation

The DutyCode framework was initially verified using three nodes and a source forming a single hop network. An oscilloscope was used to measure actual power consumption and sleep intervals. Figure 17 depicts the oscilloscope view of coded packet transmissions of the 3 nodes after receiving a page from the source. In Figure 17, two small spikes under the dotted arrow indicate a packet transmission, and the cluster of such spikes represents a "stream". The solid arrow indicates the sleep duration of a node. As the figure shows, during a stream transmission, other nodes are in sleep state. As soon as the transmission is finished, one of the other nodes starts its stream transmission.

7.2 Sleep Interval

In this experiment we investigate how sleep interval SI affects energy consumption and total dissemination time. It should be noted that AdapCode and DutyCode con-

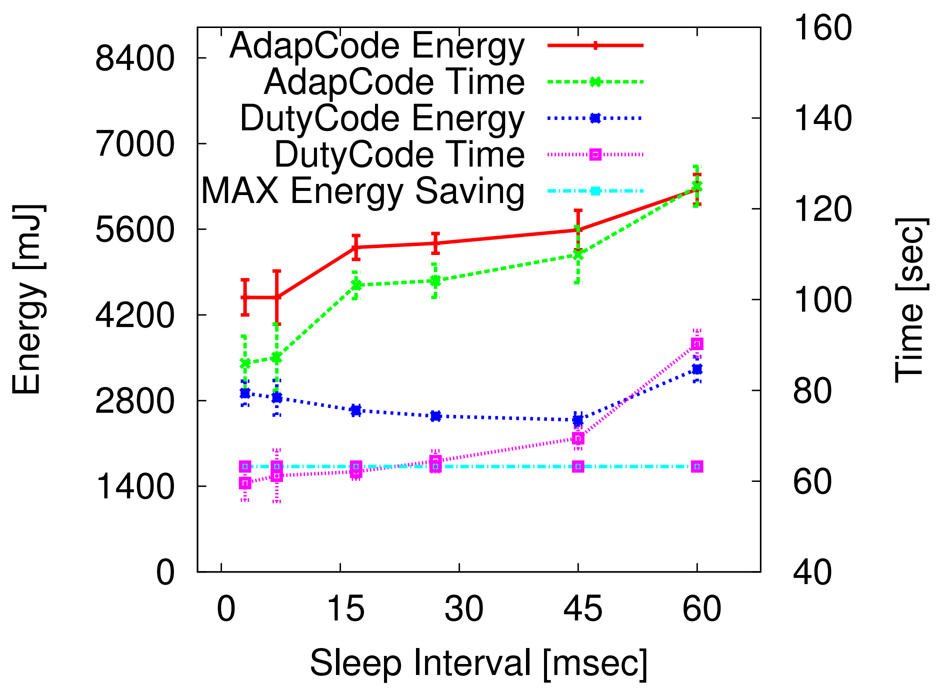

Fig. 18. The effect sleep interval has on energy consumption and time.

sume the same amount of energy when Sleep Interval is set to zero. Our intuition is that energy saving would incur with non-zero sleep interval; but if sleep interval reaches a certain point, energy consumption would increase. We measured energy consumption and total dissemination time by varying SI in the [4, 60] range, while keeping other parameters constant. Figure 18 depicts the results, which confirm our expectation. As SI increased, the energy consumption of DutyCode gradually decreased until SI was 45, after which it started to increase. According to our analysis in Chapter V, maximum energy saving is achieved when sleep duration matches stream duration. The experimental results follow the analytical result, as the maximum energy saving for our testbed was achieved when $SI = 45$. The theoretical upper bound of energy saving was also drawn in the figure for comparison. The maximum energy savings achieved for our testbed was 42%, while the theoretical bound is 66%. To complete the evaluation, we also compare the DutyCode with NoCode application,

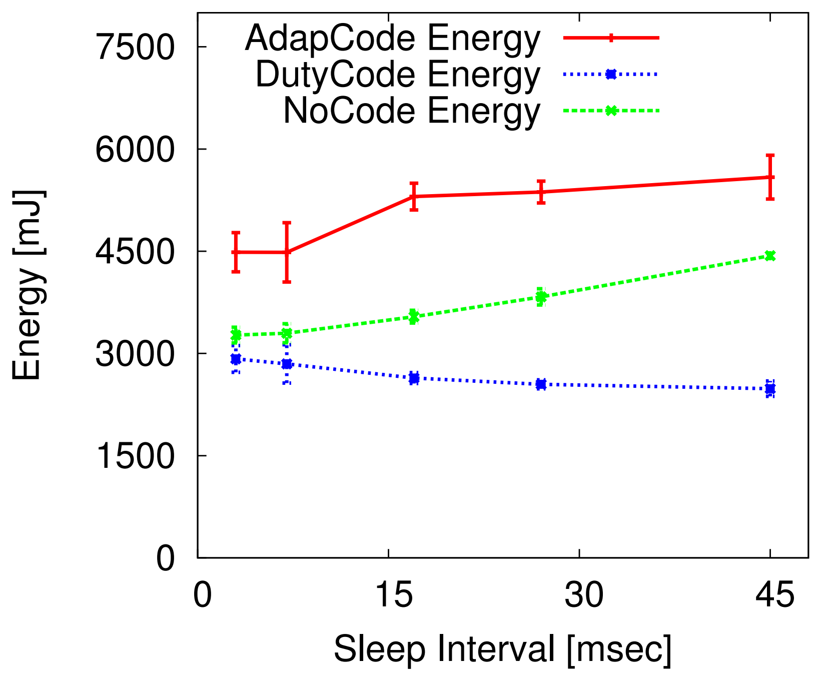

Fig. 19. Energy consumption of DutyCode, AdapCode and NoCoding.

a modified version of AdapCode which does not do network coding. The results are given in Figure 19. DutyCode outperforms the NoCode application thus revealing that DutyCode is useful for any flood-based message intense application.

7.3 Node Density

In this experiment, we explore the impact of network node density by varying TX Power. All other parameters are kept constant. A higher node density causes higher collisions, increasing energy consumption and total dissemination time. However, at the same time, it might also decrease maximum hop count from the source, given that the network size is constant like our testbed, resulting in lower energy consumption and time. This conflicting effect is depicted in Figure 20. As TX Power increased from 3 to 4 and from 5 to 6, energy consumption and dissemination time decreased due to decreased hop count, despite the higher number of collisions. However, for the

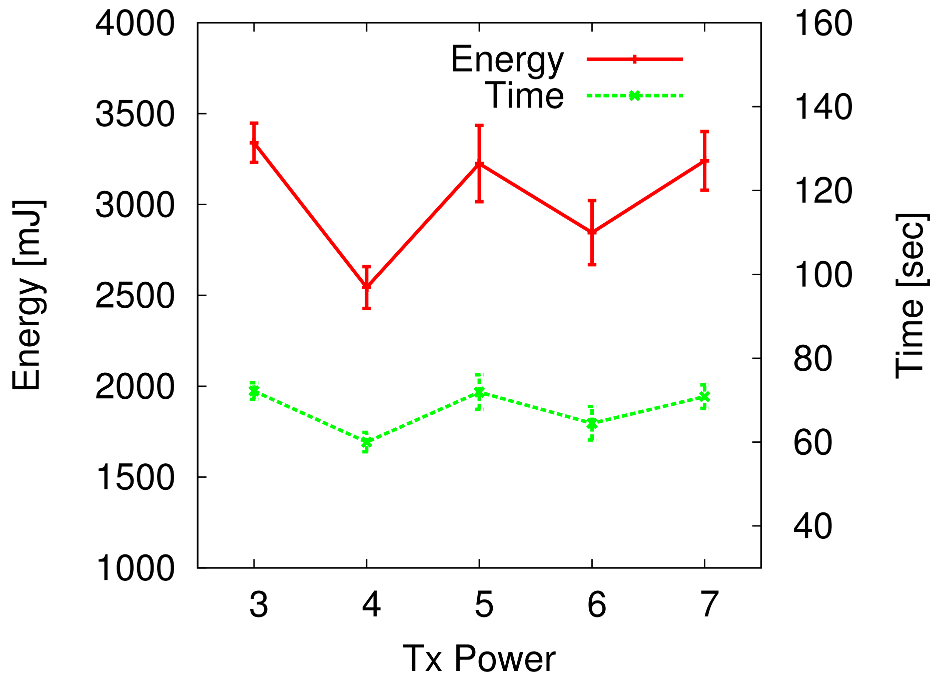

Fig. 20. The effect node density has on energy consumption and total time dissemination time.

increases of TX Power from 4 to 5 and from 6 to 7, hop count was not changed. In this case, only the higher number of collisions affected the performance, increasing energy consumption and time.

7.4 Number of Packets

We evaluated the impact of total number of packets on energy savings and code dissemination time. The results are depicted in Figure 21. As the total number of packets increased, both energy consumption and code dissemination time increased. This is because larger number of packets increases the probability of collisions and the total transmission time. To be more specific, 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 the most

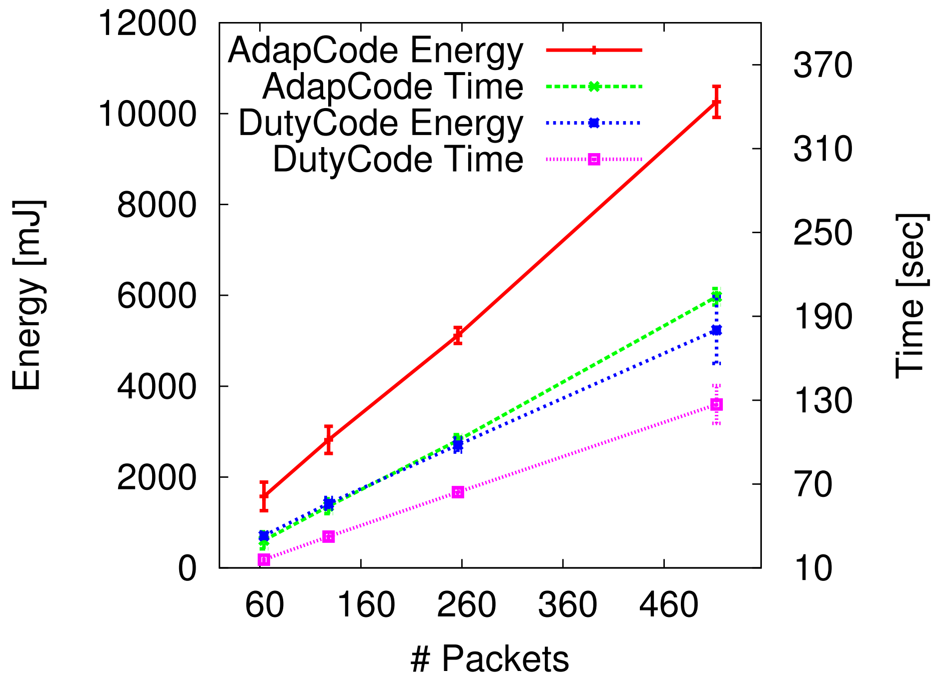

Fig. 21. The effects total number of packets has on energy consumption and total time for code dissemination

appropriate coding scheme and useless transmissions are decreased. Although energy savings increase from 866mJ to 5,021mJ, when compared to AdapCode, our solution's savings reduced from 55% to 48%. This reduction in power savings can be attributed to reduced redundant transmissions.

7.5 Packet Size

In this experiment, we investigate the effect of packet size on energy efficiency and total time. We measured energy consumption and total time by varying packet size from 28 to 108 keeping other parameters constant. The results are depicted in Figure 22. As packet size increased, both energy consumption and code dissemination time gradually decreased. With an increase in the packet size, we also increased the time gap between successive transmissions from 4msec to 9msec. This is because

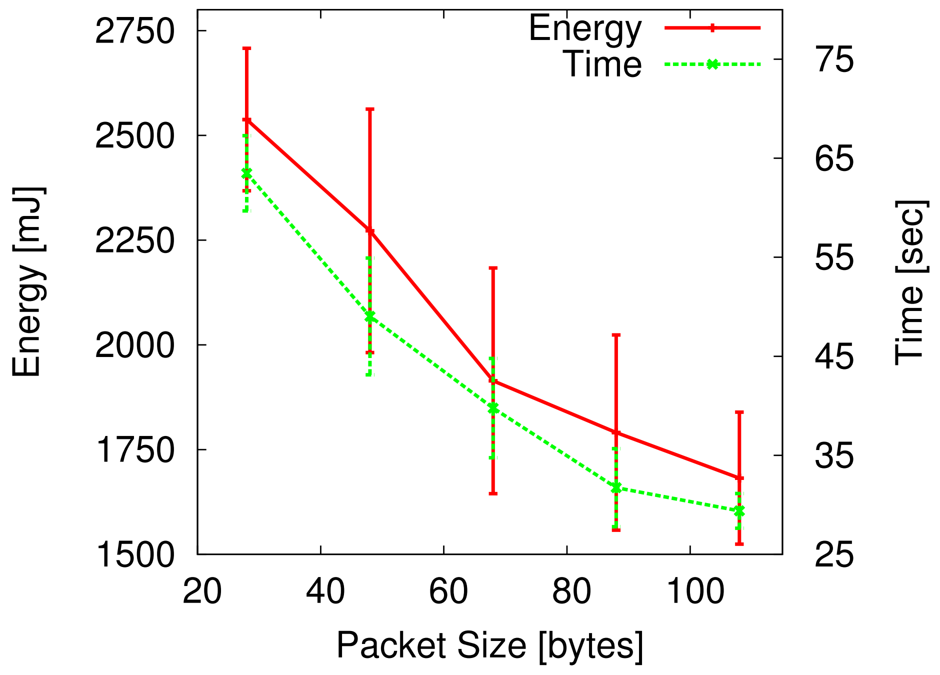

Fig. 22. Effects total number of packets has on energy consumption and total time.

as packet size increases, required computation time increases, and nodes need larger time gap between successive packet transmissions to allow more time to process a larger packet. When packet size increases 3-fold, energy consumption decreased by 33%. This emphasizes that in a packet transmission, the backoff intervals are a lot larger than the time taken for the actual packet transmission. From our experiments, it appears that we save energy using few large packets, compared to many small packets. A possible explanation is the good link quality in our testbed.

7.6 NACK Interval

In this experiment we investigate 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 23. The NACK intervals are chosen from the range [340, 740]. An

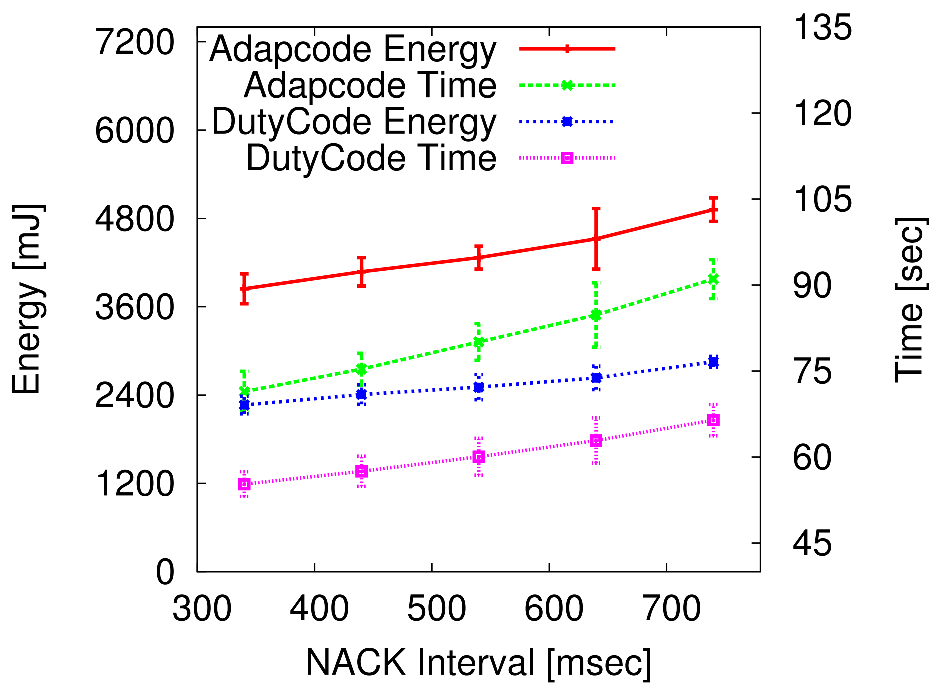

Fig. 23. Effects of NACK interval on energy consumption and total time.

increase in NACK time is expected to generate additional delays. As the NACK interval increases from 340 to 740, the increase in the energy consumption of DutyCode followed the same pattern as that of AdapCode.

7.7 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 AdapCode with different inter-page intervals in the range [300, 700] while keeping all other parameters constant, including $NACKD$. The results of our evaluation of inter-page intervals are depicted in Figure 24. 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 inter-page interval does not affect the total time taken and power consumption

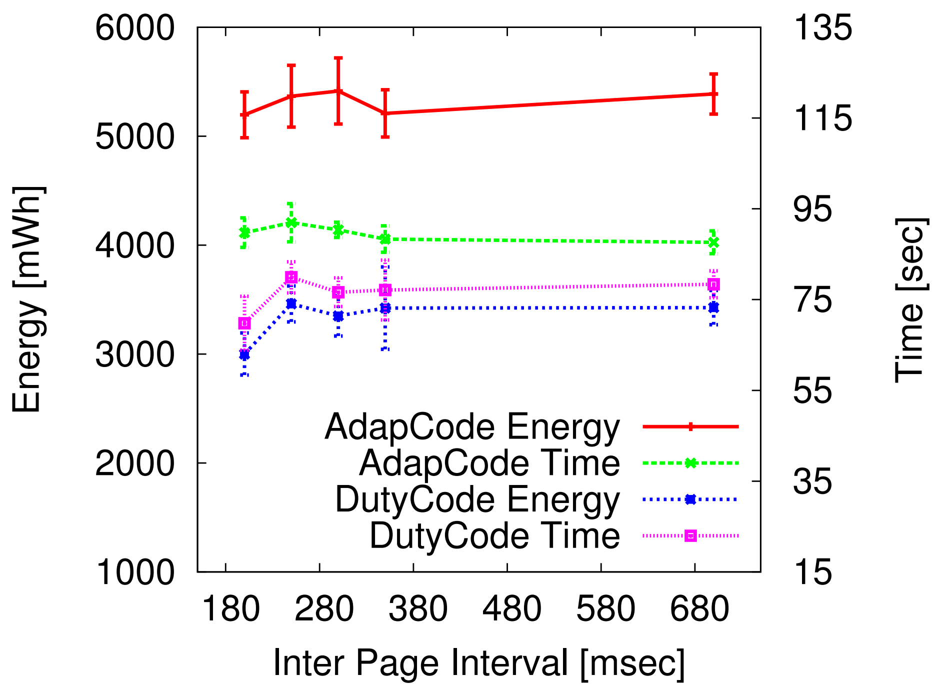

Fig. 24. Effects page interval has on energy consumption and total time.

of both AdapCode and DutyCode.

7.8 DutyCode + ECSDT

In this section, we investigate the performance gain of integrating ECSDT with DutyCode. Also, the impact of link quality threshold (LQT) is examined. LQT is a design parameter of ECSDT. Different LQT values result in different topologies, affecting the performance of ECSDT.

Performance gain of DutyCode with ECSDT. we measured energy consumption and total dissemination time for both DutyCode with ECSDT and DutyCode (without ECSDT) by varying sleep interval. For these experiments, the LQT was set to .95 to obtain accurate coding schemes and ensure sufficient redundancy for packet decoding. The results are depicted in Figure 26. The patterns of energy consumption and code dissemination time of DutyCode with ECSDT was similar to that

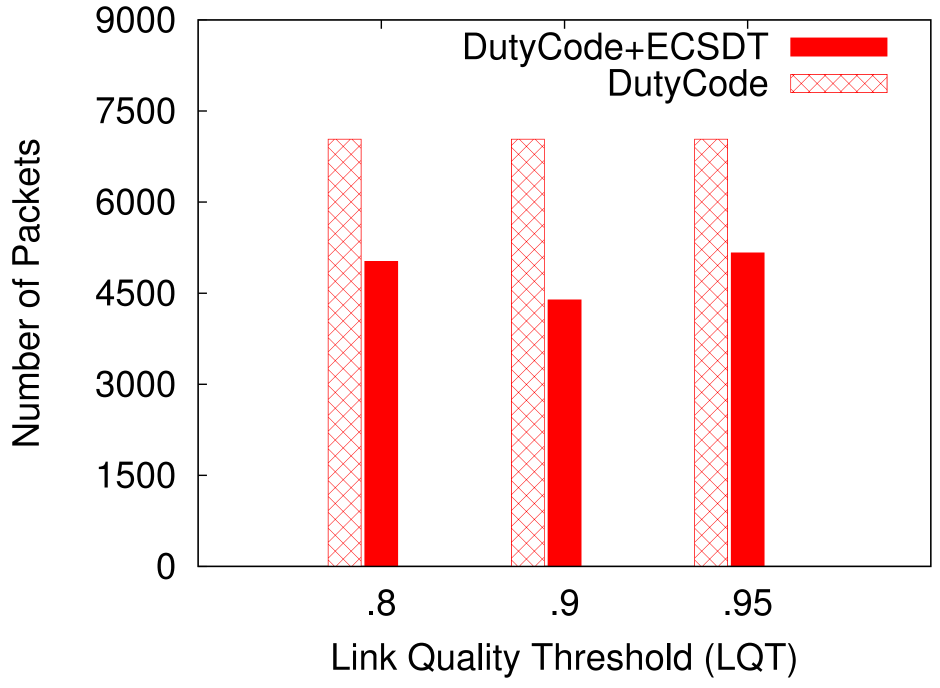

Fig. 25. Effects link quality threshold has on total number of transmitted packets.

of DutyCode. The graph also shows that DutyCode with ECSDT outperforms DutyCode: the maximum energy saving was more than 10% compared with DutyCode which is about 46% enhancement compared with AdapCode.

LQT. the total number of packet transmissions was measured for both DutyCode with ECSDT and DutyCode by varying LQT. The results are shown in Figure 25. As expected, DutyCode with ECSDT outperforms DutyCode in terms of the total packet transmissions, regardless of LQT. As LQT increased from .8 to .9, the total number of packet transmissions for DutyCode with ECSDT decreased. This is because more accurate coding schemes are assigned with higher LQT. Interestingly, however, the increase of LQT from .9 to .95 actually increased the total number of transmissions. This is because the total number of valid links tend to decrease with extremely high LQT, thereby allowing only few additional redundant transmissions.

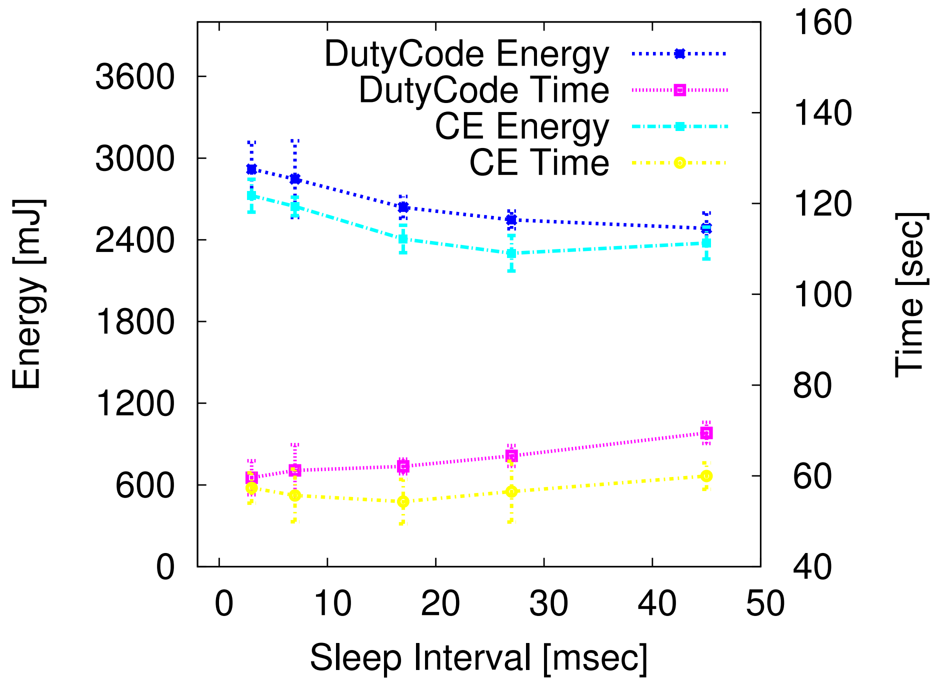

Fig. 26. Effects sleep interval has on energy consumption and total time of DutyCode with ECSDT.

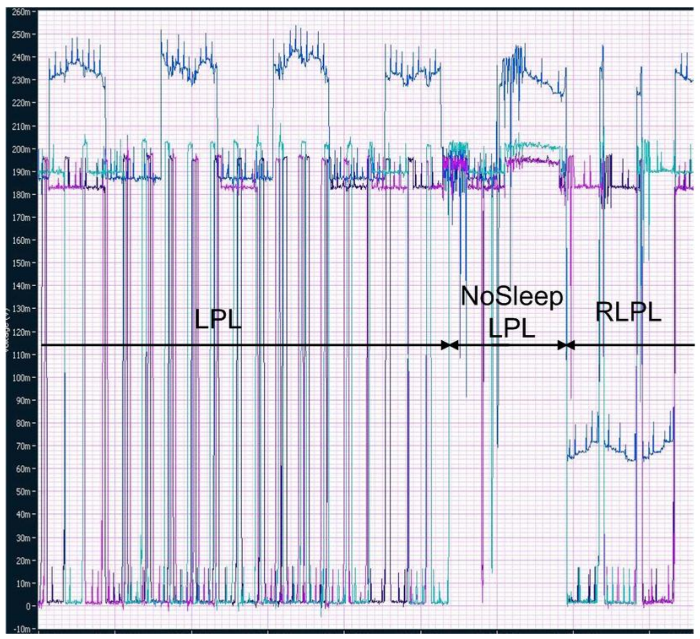

Fig. 27. A snapshot of voltage level changes showing the LPL to RLPL mode transitions.

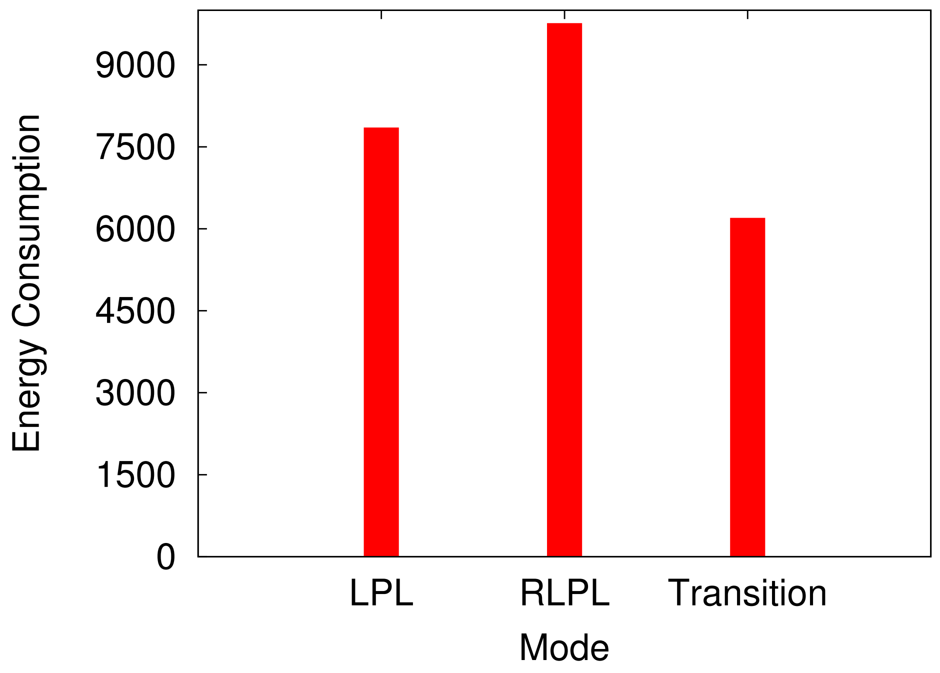

Fig. 28. Effect of MAC protocol on the energy consumption of DutyCode.

7.9 LPL/RLPL Mode Transition

The transition from LPL to RLPL as explained in section 4.4, is illustrated in Figure 27(where mode switches are denoted as changes in voltage level). The nodes are initially in LPL mode and when the source starts the code download all nodes stop duty-cycling and stays in NoSleepLPL mode for a while and switch to RLPL mode. While in NoSleepLPL mode, the nodes do not sleep but transmit the packets as in LPL mode.

To assess the effects of protocol switching, the DutyCode is tested in LPL mode, RLPL mode and in the "protocol transition" mode. The results are shown in Figure 28. The Sleep Interval value is chosen as 45msec for these experiments. This is because energy savings of DutyCode in RLPL mode achieves more energy savings compared to the DutyCode in LPL mode when the sleep interval is 45msec. The energy consumption for a 150sec time interval is shown in the figure. As expected, the energy consumption of RLPL mode is high as the nodes were awake most of the time when the code download was not happening. The “protocol transition” mode is 20% more energy efficient than the LPL mode.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

Network coding and duty-cycling are two popular techniques for saving energy in wireless adhoc and sensor networks. In this thesis, 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. ECSDT is proposed to enhance the coding scheme selection technique for further energy savings. The ECSDT, solves the coding scheme assignment problem as a graph problem with an objective to minimize the redundant transmissions. A complete solution is provided by facilitating MAC protocol transition, which can be configurable in the application based on the changes in the traffic.

Through analysis and real system implementation we demonstrate that DutyCode does not incur higher overhead, and that it achieves up to 46% more energy savings when compared with network coding-based solutions that do not use duty-cycling. Our analytical model predicts the upper bound on the energy savings to be 66%. Our experiments reveal that the proposed solution is beneficial not only for flood-based network coding applications but also for any message intense flooding applications. The proposed scheme requires minimal changes to existing network coding applications. Currently this solution targets only flood-based network coding application and we expect to extend this solution to other network coding applications. And also making this solution more secure is left for future work.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Trans. Inf. Theory*, vol. 46, pp. 1204–1216, 2000.

[2] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, “Highly-resilient, energy-efficient multipath routing in wireless sensor networks,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, no. 4, pp. 11–25, 2001.

[3] R. Ahlswede and H. Aydinian, “On error control codes for random network coding,” in *Network Coding, Theory, and Applications (NetCod).* Lausanne, Switzerland: IEEE, 2009.

[4] C. K. Ngai and R. Yeung, “Secure error-correcting (SEC) network codes,” in *Network Coding, Theory, and Applications (NetCod).* Lausanne, Switzerland: IEEE, 2009.

[5] K. Klues, V. Handziski, C. Lu, A. Wolisz, D. Culler, D. Gay, and P. Levis, “Integrating concurrency control and energy management in device drivers,” in *Proceedings of ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2007.

[6] J. Polastre, J. Hill, and D. Culler, “Versatile low power media access for wireless sensor networks,” in *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[7] M. Buettner, G. V. Yee, E. Anderson, and R. Han, “X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks,” in *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[8] 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 International Symposium on Computers and Communications (ISCC)*, 2004.

[9] 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.

[10] W. Ye, J. Heidemann, and D. Estrin, “An energy-efficient MAC protocol for wireless sensor networks,” in *Proceedings of Conference on Computer Communications (INFOCOM)*, 2002.

[11] W. Ye, F. Silva, and J. Heidemann, “Ultra-low duty cycle MAC with scheduled channel polling,” in *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.

[12] T. van Dam and K. Langendoen, “An adaptive energy-efficient MAC protocol for wireless sensor networks,” in *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[13] S. S. Ray, I. Demirkol, and W. Heinzelman, “Adv-mac: Advertisement-based mac protocol for wireless sensor networks,” in *Proceedings of International Conference on Mobile Ad-hoc and Sensor Networks*, 2009.

[14] Y. Sun, S. Du, O. Gurewitz, and D. B. Johnson, “DW-MAC: A low latency, energy efficient demand-wakeup MAC protocol for wireless sensor networks,” in *Proceedings of International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, 2008.

[15] B. Jang, J. B. Lim, and M. Sichitiu, “AS-MAC: An asynchronous scheduled

MAC protocol for wireless sensor networks," in *Proceedings of Mobile Ad Hoc and Sensor Systems (MASS) 2008*, 2008.

[16] Y. Sun, O. Gurewitz, and D. B. Johnson, "RI-MAC: A receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks," in *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[17] S. C. Ergen, C. Fischione, D. Marandin, and A. L. Sangiovanni-Vincentelli, "Duty-cycle optimization in unslotted 802.15.4 wireless sensor networks," in *Proceedings of Global Communications Conference Exhibition & Industry Forum (GLOBECOM)*, 2008.

[18] S. Guo, Y. Gu, B. Jiang, and T. He, "Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links," in *Proceedings of International Conference on Mobile Computing and Networking (Mobicom)*, 2009.

[19] J. Hong, J. Cao, W. Li, S. Lu, and D. Chen, "Sleeping schedule-aware minimum latency broadcast in wireless ad hoc networks," in *Proceedings of International Conference on Communications (ICC)*, 2009.

[20] X. Jiao, W. Lou, J. Ma, J. Cao, X. Wang, and X. Zhou, "Duty-cycle-aware minimum latency broadcast scheduling in multi-hop wireless networks," in *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 2010.

[21] Y. Sun, O. Gurewitz, S. Du, L. Tang, and D. B. Johnson, "ADB: An efficient multihop broadcast protocol based on asynchronous duty-cycling in wireless sensor networks," in *Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.

[22] T. Cui, L. Chen, and T. Ho, "Energy efficient opportunistic network coding for wireless networks," in *Proceedings of Conference on Computer Communications (Infocom)*, 2008.

[23] J. Zhang, Y. Chen, and I. Marsic, "Network coding via opportunistic forwarding in wireless mesh networks," in *Proceedings of Wireless Communications and Networking Conference (WCNS)*, 2008.

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

[25] N. Thomos and P. Frossard, "Raptor network video coding," in *Proceedings of International Workshop on Workshop on Mobile Video (MV)*, 2007.

[26] Y. Xiao, L. Ma, K. Khorasani, and A. Ikuta, "A new robust narrowband active noise control system in the presence of frequency mismatch," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 14, no. 6, pp. 2189–2200, 2006.

[27] X. Tao, C. Zhang, and J. Lu, "Network coding for energy efficient wireless multimedia transmission in ad hoc network," in *Proceedings of International Conference on Communication Technology (ICCT)*, 2006.

[28] 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 Conference on Computer Communications (INFOCOM)*, 2008.

[29] J. Widmer and J.-Y. Le Boudec, "Network coding for efficient communication in extreme networks," in *Proceedings of ACM SIGCOMM Workshop on Delay-tolerant Networking (WDTN)*, 2005.

[30] L. Li, R. Ramjee, M. Buddhikot, and S. Miller, "Network coding-based broadcast in mobile ad hoc networks," in *Proceedings of Conference on Computer Communications (Infocom)*, 2007.

[31] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[32] T.-G. Li, C.-C. Hsu, and C.-F. Chou, "On reliable transmission by adaptive network coding in wireless sensor networks," in *Proceedings of International Conference on Communications (ICC)*, 2009.

[33] H. Gong, J. Cao, M. Liu, L. Chen, and L. Xie, "A traffic aware, energy efficient MAC protocol for wireless sensor networks," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 4, no. 3/4, pp. 148–156, 2009.

[34] P. Hurni and T. Braun, "MaxMAC: A maximally traffic-adaptive MAC protocol for wireless sensor networks," in *Proceedings of European Conference on Wireless Sensor Networks (EWSN)*, 2010.

[35] M. Anwander, G. Wagenknecht, T. Braun, and K. Dolfus, "Beam: A burst-aware energy-efficient adaptive mac protocol for wireless sensor networks," in *Proceedings of International Conference on Networked Sensing Systems (INSS)*, 2010.

[36] T. Zheng, S. Radhakrishnan, and V. Sarangan, "PMAC: An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE, 2005.

[37] R. Chandanala and R. Stoleru, “Network coding in duty cycled sensor networks,” in *Proceedings of International Conference on Networked Sensing Systems (INSS)*, 2010.

[38] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler, “A building block approach to sensornet systems,” in *Proceedings of ACM Conference on Embedded Network Sensor Systems (SenSys)*, 2008.

[39] “Cc2420 data sheet,” http://www.chipcon.com, accessed on 12/2009.

VITA

Roja Ramani Chandanala was born in Andhra Pradesh, India. After completing schooling and college at the SSSM, Navodaya and the Nalanda Junior College, she attended university at the National Institute of Technology Warangal, Andhra Pradesh, India from 2002 to 2006. She received the degree of Bachelor of Technology in May, 2006. She worked with Amdocs DVCI for 2 years. She entered graduate school at Texas A&M University, College Station in August, 2008 and graduated with her Master of Science in computer science in December, 2010. She may be reached at:

Address:

Department of Computer Science and Engineering

Texas A&M University

TAMU 3112

College Station

TX 77843-3112

Email: roja.chandanala at gmail.com

The typist for this thesis was Roja Ramani Chandanala.