Dynamic Duty Cycle Control for End-to-End Delay Guarantees in Wireless Sensor Networks

Xiaodong Wang†, Xiaorui Wang†, Guoliang Xing‡, and Yanjun Yao†
†*Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996*
‡*Department of Computer Science and Engineering, Michigan State University, MI 48824*
{*xwang33, xwang, yyao9*}*@utk.edu glxing@cse.msu.edu*

***Abstract*— It is well known that periodically putting nodes into sleep can effectively save energy in wireless sensor networks, at the cost of increased communication delays. However, most existing work mainly focuses on static sleep scheduling, which cannot guarantee the desired delay when the network conditions change dynamically. In many applications with user-specified end-to-end delay requirements, the duty cycle of every node should be tuned individually at runtime based on the network conditions to achieve the desired end-to-end delay guarantees and energy efficiency. In this paper, we propose DutyCon, a control theory-based dynamic duty cycle control approach. DutyCon decomposes the end-to-end delay guarantee problem into a set of single-hop delay guarantee problems along each data flow in the network. We then formulate the single-hop delay guarantee problem as a dynamic feedback control problem and design the controller rigorously, based on feedback control theory, for analytic assurance of control accuracy and system stability. DutyCon also features a queuing delay adaptation scheme that adapts the duty cycle of each node to unpredictable packet rates, as well as a novel energy balancing approach that extends the network lifetime by dynamically adjusting the delay requirement allocated to each hop. Our empirical results on a hardware testbed demonstrate that DutyCon can effectively achieve the desired tradeoff between end-to-end delay and energy conservation. Extensive simulation results also show that DutyCon outperforms two baseline sleep scheduling protocols by having more energy savings while meeting the end-to-end delay requirements.**

I. Introduction

Many wireless sensor network (WSN) applications require information to be transmitted to the destination in a timely manner. For example, in military surveillance applications [1], authorities should be notified promptly upon the detection of intruders. On the other hand, many nodes in a WSN use batteries as their energy source. It is therefore critical to reduce the energy consumption of those nodes in order to achieve a longer network lifetime. Periodically putting the radios of WSN devices into sleep has been widely recognized as the most effective way of saving energy in WSNs. Taking the commonly used TelosB mote [2] as an example, the idle listening power consumed by the radio is more than 1,000 times the power consumed when it is sleeping [3]. However, a problem of putting radio into sleep is that it may incur an additional communication delay, known as *sleeping delay* [4], since the sender of a communication pair must wait for the receiver to wake up and receive a packet. Therefore, it is important to balance the tradeoff between energy savings and the delay incured by using the periodic sleeping scheme.

Power management with node sleeping has been extensively studied in WSNs. The existing power management schemes can be categorized into three classes. The first class includes various TDMA protocols, such as TRAMA [5] and DRAND [6]. However, a node in TDMA networks has to wait for its time slot to transmit, which is inefficient for applications with tight and varying delay requirements. The second class includes synchronous duty cycling protocols, such as S-MAC [4] and T-MAC [7]. The major issue with these protocols is that the sleep schedules of nodes need to be frequently synchronized, which may lead to energy waste and additional communication delays. The third class of power management schemes consists of asynchronous channel polling protocols, such as B-MAC [8] and X-MAC [9]. Nodes in these protocols wake up periodically to poll the channel for activities. If the channel is busy, they stay awake and prepare to get data. However, nodes using the channel polling approach are usually configured with static duty cycles, which may lead to unpredictable delay performance when network conditions (*e.g.*, interference, traffic load) change dynamically.

In this paper, we propose DutyCon, a dynamic duty cycle control scheme that provides an end-to-end communication delay guarantee while taking advantage of periodic sleeping to achieve energy efficiency. DutyCon is significantly different from the existing work on delay and duty cycle management in WSNs in three aspects. First, we control the end-to-end delay of each data flow in a WSN to a user-specific bound while achieving energy conservation. Second, we dynamically adjust the duty cycle of each node individually to adapt to the network condition changes in different areas in the WSN. Network conditions in a WSN can vary both spatially and temporally [10]. Third, DutyCon can also adapt to the unpredictable incoming packet rate changes, which are common in many WSN-based monitoring applications, *e.g.*, packets can be generated at a higher rate when an emergency event occurs.

In DutyCon, to meet the end-to-end delay requirement, we decompose the end-to-end delay control problem into a set of single-hop delay control subproblems by assigning a local delay requirement to each single hop along a communication flow. A delay model is established to characterize the relationship between the single-hop delay and the sleep interval of the receiver's sleep schedule. To solve the single-hop delay control problem, we design a feedback controller based on the well-established feedback control theory [11]. The advantage of having control theory as a theoretical foundation is that we can utilize standard approaches to choosing the right control parameters so that exhaustive iterations of tuning and testing are avoided. In addition, control theory provides us analytical assurance of delay control accuracy and system stability. This rigorous design methodology is in sharp contrast to heuristic-based adaptive solutions that rely on extensive empirical evaluation and manual tuning. When assigning the

single-hop delay requirement to each hop, DutyCon considers energy balancing on different nodes along an end-to-end flow. DutyCon also features a queuing delay adaptation strategy to dynamically change the duty cycle and coordinate with the feedback control approach to adapt to the unpredictable packet rate. We validate and evaluate the design of DutyCon both on a real hardware testbed and in the NS-2 simulator. Specifically, the contributions of this paper are four-fold.

- We propose DutyCon, a new approach to meeting end-to-end delay requirements in a WSN. In DutyCon, the end-to-end delay guarantee problem is decomposed into a set of single-hop delay guarantee problems along each data flow. We then formulate the single-hop delay guarantee problem as a feedback control problem.
- Based on the single-hop delay model, we design a delay controller, based on well-established feedback control theory, to control each single-hop delay in a distributed manner by dynamically changing the sleep interval in each receiver's sleep schedule.
- We design a strategy to adapt the sleep interval of each node to the queuing delay incurred by the unpredictable incoming packet rate. We also propose an energy balancing scheme that balances the energy consumptions of nodes at different portions of each end-to-end communication flow when assigning the single-hop delay requirement to each hop.
- We evaluate DutyCon both on a real hardware testbed composed of Tmote Sky motes and in the NS-2 simulator, and compare DutyCon with two baselines, STS and DTS, recently proposed in [12].

The remainder of this paper is organized as follows. Section II highlights the distinction of our work by discussing the related work. Section III introduces the formulations of our end-to-end delay and single-hop delay control problems. Section IV presents the controller design for the single-hop delay control problem. Section V elaborates on the design of the queuing delay adaptation strategy. In Section VI, we propose two assignment schemes for the single-hop delay requirement. In Section VII, we evaluate DutyCon using both testbed experiments and simulations. Section VIII concludes the paper.

II. Related Work

Several protocols have been proposed to provide delay guarantees for wireless sensor and ad hoc networks. Implicit EDF [13] is a collision-free scheduling scheme which provides delay guarantees by exploiting the periodicity of WSN traffic. RAP [14] uses a velocity monotonic scheduling scheme to prioritize real-time traffic based on a packet's deadline and its distance to the destination. SPEED [15] achieves end-to-end communication delay guarantees by enforcing a uniform communication speed throughout the network. Karenos et al. [16] have also presented a flow-based traffic management mechanism to provide delay guarantees. Our work is different from the aforementioned research. By dynamically manipulating the sleep interval, we provide delay guarantees for end-to-end communications, while the delay incurred by sleeping nodes is not considered in the aforementioned protocols.

Periodic sleeping is a widely adopted approach to saving energy for WSNs. The existing periodic sleeping approaches can be categorized into two classes: *static sleep scheduling* and *dynamic sleep scheduling*. In the static sleeping approach category, S-MAC [4] proposes a synchronous periodic sleeping MAC with fixed duty cycles for energy savings. D-MAC [17] is developed especially for a tree topology network. It aims to reduce sleep latency while decreasing energy consumption. Several other static sleep scheduling protocols (*e.g.*, [18][19]) are also proposed. However, none of the above studies provides delay guarantees when utilizing periodic sleeping to save energy. A static sleep scheduling approach with delay guarantee has been recently proposed [20]. However, it cannot adapt to network condition changes such as interference incurred by additional workload at runtime. The second class of periodic sleeping schemes is dynamic sleeping scheduling. In those approaches, nodes are allowed to have different sleep schedules and change their schedules dynamically at runtime. Min et al. [21] propose to choose different nodes to go to sleep at different time based on the Analytic Hierarchy Process (AHP). Ning et al. [22] propose to use the dynamic programing approach to control sleep time of nodes for energy minimization. Different from the existing dynamic sleep scheduling approaches, our work dynamically adjusts the sleep interval of each node based on the delay constraint and the network condition changes.

The control-theoretic approach has been applied to various computing and networking systems. A survey of feedback performance control for software services is presented in [23]. However, only a few recent studies in wireless sensor networks start to utilize feedback control theory to provide performance guarantees. ATPC [24] employs a feedback-based transmission power control algorithm to dynamically maintain individual link quality over time in WSNs. Merlin et al. [25] propose to control the duty cycle of each node for the desired throughput in a WSN. A control-theoretical approach is also designed in [26] to achieve the maximum network throughput in multi-channel WSN. To our best knowledge, this work is the *first* one that takes advantage of feedback control theory to dynamically control the sleep interval of every individual node for end-to-end delay guarantees in WSNs.

III. Problem Formulation

In this section, we introduce the formulation of our problem. We assume that the network is composed of m sources, some relay nodes and multiple sinks. A data flow in the network includes a source to generate data packets based on a certain distribution, multiple nodes to relay the packets hop by hop, and a corresponding sink node. There are m flows in the network, each of which is generated by one of the m sources. All flows are assumed to be disjoint with each other because disjoint flows are widely used in multi-path routing to enhance the system's fault-tolerance [27][28]. However, our approach can be easily extended to the case where multiple flows share the same nodes by allowing the shared nodes to wake up at all the wake-up time instants decided by different flows.

We assume that nodes operate in a periodic sleep schedule where each sleep period consists of a *sleep interval* and a *wake-up interval*. The *sleep interval* is the time duration when

the node's radio is off in each sleep period. The *wake-up interval* is the time duration that a node has its radio on to transmit packet. Our primary goal is to design a dynamic duty cycle control policy to dynamically tune the sleep interval of each node so that the communication on each data flow can achieve an end-to-end delay guarantee while taking advantage of periodic sleeping to save energy. We first introduce the following notation:

- $G = (E, V)$, a WSN where V is the node set and E is the communication link set of the network.
- $f(u_0, u_n)$, a node set which forms a single data flow with source u_0 and destination u_n. The node index in a flow is enumerated from 0 to n in the hop sequence. Specifically, $f(u_0, u_n)$=$\{u_i|(u_{i-1}, u_i) \in E, 1 \leq i \leq n\}$.
- F, the set of all m flows. Specifically, $F = \{f_j|\forall j : 1 \leq j \leq m\}$
- d_i, the single-hop communication delay on link (u_{i-1}, u_i), which is formally defined in Section IV.
- $D_{ref}^{f_j}$, the end-to-end delay requirement of flow f_j.
- c_i, the time length of the sleep interval in one sleep period of node u_i.

Our goal is to control the end-to-end communication delay according to a given end-to-end delay requirement for each data flow. We formulate this problem as follows:

$$\min_{\forall f_j \in F} \left| \sum_{u_i \in f_j} d_i - D_{ref}^{f_j} \right| \tag{1}$$

However, to solve the problem defined above, one must know the global information of a flow in the WSN, such as the communication delay of each hop. This is inefficient, especially when flows have high hop counts. As our goal is to achieve the end-to-end delay control, we decompose our end-to-end delay control problem into a set of single-hop delay control subproblems. By achieving the single-hop delay control goal, we can control the multi-hop end-to-end delay. Specifically, for each flow f_j, our objective is:

$$\min_{\forall u_i \in f_j} \left| d_i - D_{ref}^{i} \right| \tag{2}$$

subject to the constraints

$$\sum_{u_i \in f_j} D_{ref}^{i} = D_{ref}^{f_j} \tag{3}$$

$$d_i \geq D_{min} \tag{4}$$

$$c_i \geq 0 \tag{5}$$

where D_{ref}^{i} is the single-hop delay requirement for link (u_{i-1}, u_i) and D_{min} is the minimum transmission time for a packet to be successfully received by the receiver. Constraint (3) enforces the single-hop delay requirement based on the end-to-end delay requirement. Constraint (4) means that the single-hop transmission delay has a lower bound, which is decided by the transmission rate of the radio and the packet size. Constraint (5) enforces that sleep interval of any node must be non-negative.

IV. Single-hop Delay Control

In this section, we first present a single-hop delay model. Our model characterizes the expected one-hop communication delay by taking into account several realistic factors, such as network conditions and retransmission delays due to lossy links. Based on the model, we introduce the design of the single-hop delay controller.

A. Single-hop Delay Model

We assume that after packets are received by a node, they are immediately ready for being transmitted to the next hop without a queuing delay. This assumption is relaxed in Section V. The sender is assumed to know the sleep schedule of receiver. We will discuss in Section IV-D how this can be achieved. We also assume that the sender will try to send the packet only once every time the receiver wakes up. If the packet is not successfully received by the receiver during the wake-up time of the current sleep period, the sender will go to sleep and try to send the packet at the receiver's wake-up time in the next period. This prevents a sender from keeping using the channel for a long time, so that other nodes cannot get the channel during their wake-up times, especially when the link quality is low. Thus, the time delay, $d(k)$, for the kth packet to transmit from the sender to the receiver can be modeled as:

$$d(k) = \frac{c(k-1) + t_{data}}{PRR(k)} \tag{6}$$

where $c(k-1)$ is the sleep interval to be used at the receiver after the $(k-1)$th packet is received. $PRR(k)$ is the average packet reception ratio estimated when the kth packet is ready for transmission. It is a metric widely used to quantify the quality of links [29]. t_{data} is the time needed to transmit one packet after getting the channel, which includes processing time and the time to transmit the packet on radio. It can be approximated as a constant because the packet size and transmission rate usually do not change and it is significantly smaller than the sleep period. Since we do not use CSMA, t_{data} does not include a backoff time. Contention interference is captured in $PRR(k)$.

Figure 1 illustrates the single-hop delay model in Equation (6). The total single-hop delay is the number of transmissions it takes to successfully transmit the packet multiplied by the time needed for each sleep period, which is the summation of the sleep interval and the wake-up time to transmit a packet. In the case that the packets arrive aperiodically, we can simply add a time difference quantity to Equation (6), which is defined as the time difference between the packet arrival time and the closest wake up time of the sender.

We assume that the average network condition of a link does not change frequently, compared with the wake-up frequency of the node on that link. In this case, PRR remains constant between the arrivals of two back-to-back packets. We assume that the PRR values of the links in our network are higher than a threshold in order to reach the communication link requirement [29]. Zhao et al. [30] show that the PRR value is temporally stable when it is relatively high. Therefore, we simplify the PRR value to be a constant during the transmission of two consecutive packets. Using Equation (6),

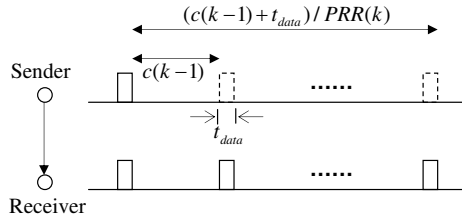

Fig. 1. Single-hop delay is the time length of total sleep periods used to successfully transmit a packet.

we can derive the dynamic model of our single-hop communication delay as a difference equation in Equation (7), where $\Delta c(k) = c(k+1) - c(k)$.

$$d(k+1) = d(k) + \frac{\Delta c(k)}{PRR} \tag{7}$$

In the above model, PRR is the estimated average success rate of the packet transmission on a single link. In different areas of the network, we may have different values of PRR due to network condition variations. In order to capture the uncertainty of the network condition, we add an uncertainty ratio to our model as:

$$d(k+1) = d(k) + g\frac{\Delta c(k)}{PRR} \tag{8}$$

where g represents the ratio between the estimation of PRR and the actual PRR under the current network condition due to the uncertainty of the network environment. Note that the exact value of g is unknown at design time due to the unpredictable network condition. We explain how we handle this uncertainty in the next subsections.

B. Feedback Controller Design

The core of any feedback control loop is the controller. In each control period, the controller monitors and controls a *controlled variable* by adjusting a system parameter, called *manipulated variable*, in order to meet a system requirement, usually called *performance reference*. In our problem, we try to control the single-hop delay of each packet to meet the delay requirement by dynamically adjusting the receiver's sleep interval. Therefore, the *controlled variable* in our problem is the single-hop communication delay of the next packet. The *manipulated variable* is the sleep interval time that the receiver sets for the next packet and the performance reference is the single-hop delay requirement, denoted as D_{ref}.

Note that the model in Equation (8) cannot be directly used to design the controller because the g is used to model the uncertainties in the network conditions and thus, is unknown at design time. Therefore, we design the controller based on an approximate system model, which is model (8) with $g = 1$. In a real network where the packet reception ratio is different from the estimation, the actual value of g may be different than 1. As a result, the closed-loop system may behave differently. However, in next subsection, we show that a single-hop delay controlled by the controller designed with $g = 1$ can remain stable as long as the variation of g is within a certain range. This range is established using a stability analysis of the closed-loop system by considering model variations.

Following standard control theory [11], we design a Proportional (P) controller to achieve the desired control performance, such as stability. We can derive the receiver's desired sleep interval for the kth packet as shown in Equation 9.

$$c(k) = (D_{ref} - d(k)) \times PRR + c(k-1) \tag{9}$$

It is easy to prove that the controlled system is stable and has zero steady state errors when $g = 1$. The detailed proofs and design procedures can be found in a standard control textbook [11] and are skipped due to space limitations. As shown in Equation 9, the computational overhead of the P controller is just two additions/subtractions and one multiplication and is thus small enough to be implemented in a real sensor mote.

C. Stability Analysis for PRR Variation

In this section, we analyze the system stability when the designed P controller is used in an area where $g \neq 1$. A fundamental advantage of the control-theoretic approach is that it provides confidence for system stability, even when the packet reception ratio deviates from the estimation.

The closed-loop transfer function for the real-system is:

$$G(z) = \frac{g}{z - (1-g)} \tag{10}$$

The closed-loop system pole in Equation (10) is $1 - g$. In order for the controller to be stable, the pole must be within the unit circle. Hence, the system will remain stable as long as $0 < g < 2$. The result means that despite every link may have a different g, in order to achieve stability, the estimated PRR of a link should be less than twice its actual PRR. In WSN applications, we usually have a lower threshold of PRR, which is the lowest PRR that can provide an acceptable communication quality [29]. If the actual PRR of a link is less than the threshold, we simply cannot use this link for communication. As our goal is to dynamically control the sleep interval to achieve the delay guarantee, we assume that the links in the control problem are communication links. Thus, with a lower threshold of PRR, we can have the stability guarantee. For example, if the threshold of PRR is 0.5 which indicates that a packet is retransmitted twice on average before it is successfully received, we can use any estimated PRR value in the controller for stability, since the estimated PRR is always less than 1. Detailed empirical studies on link quality and PRR can be found in [29].

D. Implementation

A periodic sleeping scheme requires the sender to be aware of the receiver's sleep schedule, so that the sender can also wake up to transmit packet when the receiver wakes up. This means that the information of the sleep interval change of the receiver needs to be shared by the sender. Therefore, we implement the controller on the receiver side and take advantage of the ACK packet to feed the updated sleep interval back to the sender side.

On the sender side, the sender first timestamps the packet when the packet is received (or generated if the sender is source). Then the sender will add the transmission times of the current packet in the packet header. This information is updated every time the packet is being transmitted (or

retransmitted). Upon successfully receiving the packet, the receiver calculates the time delay based on the time stamp on the sender side and runs the controller to compute the new sleep interval for the receiver's sleep scheduling, starting from the next packet. The receiver then inserts the newly updated sleep interval value in the ACK packet and sends the ACK back to the sender. The sender updates the receiver's sleep scheduling information on its own side for future transmissions. To handle the loss of ACK packets, we further adopt a localized synchronization scheme. Whenever the sleep schedule is successfully updated at both the sender and receiver through an ACK, we set up a timer with an interval several times longer than the duty cycle. If the sleep schedule is not updated during this interval, the sender and receiver wake up themselves and synchronize their schedules.

V. Queuing Delay Adaptation

In the last section, we assume that the incoming packet rate on the sender side is low, such that the packet is immediately available for transmission without a queuing delay. However, the packet receiving time (or generating time) at the sender is usually aperiodic and unpredictable, especially in event-driven WSN applications. If the packet generation rate is high in such applications, the transmission of the packet may suffer additional time latency because of the queuing delay at the sender. With the queuing delay, the model of our transmission delay in Equation (6) is subject to changes. In this section, we consider the queuing delay caused by the unpredictable packet rate when adjusting the sleep interval for the next packet.

A. Impact of Queuing Delay

As introduced previously, when the incoming packet rate is high, the packet may suffer additional delay from queuing on the sender side because of the busy MAC layer. Therefore, we need to consider the queuing delay in our single-hop delay model. The model from Equation (6) is changed as follows

$$D(k) = t^q(k) + \frac{c(k-1) + t_{data}}{PRR(k)} \tag{11}$$

where $t^q(k)$ is the queuing delay of the kth packet at the sender and $D(k)$ is the total delay when the queuing delay is counted. Compared with the queuing delay, the delay incurred by the upper layer for computation purpose is small and negligible. Therefore, the queuing delay can be calculated as the time difference between the moment the packet is received and the moment the packet is ready to be sent in the MAC layer. In essence, the queuing delay of the jth packet in the queue is the time needed to transmit all the packets that are ahead of packet j in the queue. With the assumption we made in Section IV-A that the network condition does not change frequently, the queuing delay for the jth packet in the queue can be calculated as:

$$t^q(j) = \sum_{n=1}^{j-1} d(n) = \sum_{n=1}^{j-1} \frac{c(n-1) + t_{data}}{PRR} \tag{12}$$

From Equation (12), we know that the queuing delay of the jth packet in the queue depends highly on the delay of all the packets ahead of it in the queue. The goal of our design is to choose a sleep interval at the receiver such that the single-hop delay for the next packet is controlled to the reference D_{ref}. Therefore, when calculating the sleep interval at the receiver for the next packet, we need to consider its impact on the queuing delays of all the packets in the queue on the sender side.

Suppose that the new sleep interval $c(k)$ will be used until the queue on the sender side empties. We can calculate the queuing delay of any packet $(k+n)$ (the nth packet currently in the queue) as:

$$t^q(k+n) = \frac{n(c(k) + t_{data})}{PRR} \quad \forall n : 1 \le n \le j \tag{13}$$

In order to achieve energy efficiency, we need to choose a maximum sleep interval $c(k)$ under the constraint that the new sleep interval will not cause any packet in the queue to violate the delay requirement. We denote the slack time left for the $(k+n)$th packet until it expires, based on the single-hop reference D_{ref}, as $T_{slack}(k+n)$. The formulation of this sleep interval calculation is as follows:

$$\max \ c(k) \tag{14}$$

subject to the constraint:

$$\begin{aligned} T_{slack}(k+n) \ge & t^q(k+n) + d(k+n) \\ & \forall n : 1 \le n \le j \end{aligned} \tag{15}$$

where j is the total packet number in the queue after the kth packet.

B. Implementation and Coordination with Single-Hop Delay Controller

We now discuss the details of the implementation for the sleep interval calculation when we need to adapt to the queuing delay. Similar to the single-hop delay controller designed in Section IV, we implement the new sleep interval computation on the receiver side. The difference is that the sender will first compute a minimal *nominal sleep interval* only based on the slack time of each packet in the queue, without considering the network condition. The *nominal sleep interval* is added to the first packet in the queue, which is the next packet to be transmitted. Upon receiving the packet, the receiver will calculate the real desired sleep interval for the next packet, based on the current network conditions. After the calculation of the new sleep interval, the receiver uses the ACK packet to feed the sleep interval change back to the sender.

Since we have two approaches that both dynamically adjust the sleep interval at the receiver: feedback delay control and queuing delay adaptation, a coordination scheme must be designed in order to avoid any conflicts. In our design, we use queuing delay adaptation when the queue at the sender is not empty. The reason is that the delay controller controls the delay of every packet. If we use the delay controller to perform feedback control for every packet when there are packets waiting in the queue, the packets in the queue after the current packet are likely to miss their delay requirements. This is because that the delay controller does not consider the queuing delay incurred by the unpredictable incoming packet rate. If there is no packet in the queue when the current packet

is sent at the sender, we then use the delay controller to perform the packet-level delay control.

VI. Single-Hop Delay Requirement in End-to-end Delay Guarantee

In the previous sections, we have introduced how to control the single-hop delay based on feedback control and queuing delay adaptation. As our final goal is to control the end-to-end delay of a flow f based on the requirement D^f_{ref}, in this section, we introduce how to set the single-hop delay requirement D^i_{ref}, which is the function of Constraint (3) in Section III. As long as each hop can meet its single-hop delay requirement, the desired end-to-end delay is guaranteed.

A. Worst-case Assignment

The first assignment scheme for the single-hop delay requirement is to use the worst-case analysis approach from [27]. Specifically, we perform the assignment for the single-hop delay requirement by calculating the worst-case PRR of each hop and assign the single-hop delay requirement proportionally. The worst-case PRR calculation is explained in detail in [27]. After obtaining the worst-case PRR for each link along the flow, we break the end-to-end delay requirement into a single-hop delay requirement on each hop as:

$$D^i_{ref} = \frac{1/PRR^w_i}{\sum_{u_i \in f}(1/PRR^w_i)} \times D^f_{ref} \tag{16}$$

where PRR^w_i is the worst-case PRR for link (u_{i-1}, u_i) on flow f.

The worst-case assignment considers the worst-case scenario in the network such that a hop with a worse PRR^w is assigned a longer single-hop delay requirement. Those nodes with shorter single-hop delay requirements will wake up more often than those with longer delay requirements. This is a pessimistic and static assignment, because the worst-case scenario does not happen frequently in real networks. By assigning an unnecessarily longer delay requirement to the worse links in the worst-case scenario, the nodes in the data flow may have unbalanced energy consumption.

B. Assignment for Energy Balancing

In this section, we introduce the second assignment scheme to determine the single-hop delay requirements. We propose to periodically update the delay requirement at each hop in the flow. During the transmission of the packet, each node adds the actual average packet reception ratio information of its own receiving link into the packet. The sink will periodically calculate the desired single-hop delay requirement based on the average packet reception ratio at each hop. The sink then sends out the updated delay assignment after each calculation. All the nodes update their single-hop delay requirement information upon receiving this requirement assignment packet. Since the delay requirements are periodically updated based on the current network conditions, each hop can have a fair delay requirement, such that the duty cycle of each receiving node is tuned to be approximately the same, thus leading to the energy consumption balancing. This procedure incurs a small overhead, as the calculation is performed periodically. The period can be set to relatively long if the network condition does not change dramatically.

VII. Performance Evaluation

In this section, we evaluate our dynamic duty cycle control (DutyCon) scheme based on the experiments both on a hardware testbed and in the NS-2 simulator.

A. Testbed Results

We now present the results from two sets of experiments conducted on real hardware testbeds. In the first experiment, we validate the single-hop controller. The second experiment evaluates the end-to-end delay control with local interference on a single multi-hop flow.

In the first experiment, we set the delay reference of the single-hop transmission to three different values during three different periods, which are 1s, 2s and 1.5s, respectively. The results are shown in Figure 2. We can see that the single-hop delay is approximately controlled to all the different reference times in different periods.

We now test DutyCon in a scenario of end-to-end communications. The testbed we use consists of 7 Tmote Sky motes, five of which construct a 4-hop end-to-end communication flow. The remaining two motes form a single communication link, periodically transmitting packets and introducing interference to the 4-hop flow. The distance between each two adjacent motes in the 4-hop flow is 5m. The interference link is placed 5m apart from the center node of the flow. The source of the 4-hop end-to-end flow sends out packets, following the uniform distribution at a rate of averagely 1 packet every 3 seconds. The interference link transmits packets at the rate of 1 packet per second. 200 packets are sent by the source in each run of the experiments. We configure the transmitting power of the interference node to be small, so that they will only interfere with the center node of the end-to-end flow.

We evaluate the end-to-end delay control performance of DutyCon on the single flow with different end-to-end delay requirements. We also compare DutyCon with a *Uniform Fixed Duty Cycle* baseline. In this baseline, we set the same fixed duty cycle for every node. The duty cycle we use is the highest duty cycle among all the links from the experiment using DutyCon scheme. The reason we choose this duty cycle is that the local interference is usually unknown a priori. With fixed duty cycle, we want to prepare for the worst-case scenario. Therefore, we use the highest possible duty cycle. From the results in Figure 3, we can see that DutyCon can control the average end-to-end delay very close to the delay requirement at all different values. However, using the worst-case duty cycle leads to unnecessarily low end-to-end delays in the *Uniform Fixed Duty Cycle* scheme. Figure 4 shows the average duty cycle of all motes in the end-to-end flow except the source node. We can see that the *Uniform Fixed Duty Cycle* scheme has much higher duty cycles (and thus more energy consumption) than DutyCon at all different delay requirements. The reason is that the baseline statically set the duty cycles of all links to the same worst-case value, which leads to an unnecessary energy waste.

B. Baselines and Simulation Settings

In the following sections, we present the simulation results in the NS-2 simulator. In order to show the efficiency of our

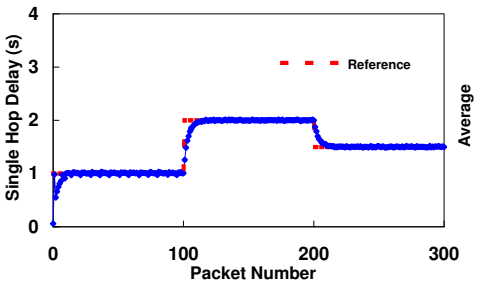

Fig. 2. Single-hop delay under control when delay requirement changes at runtime.

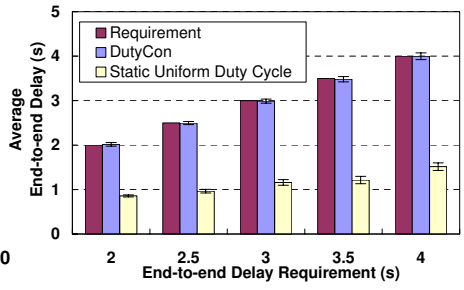

Fig. 3. End-to-end delay on the testbed under different delay requirements.

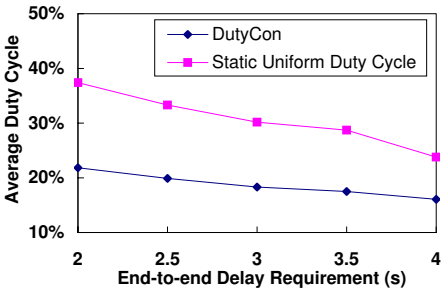

Fig. 4. Average duty cycle on the testbed under different end-to-end delay requirements.

design, we compare DutyCon with two recently published baselines: Static Traffic Shaper (STS) and Dynamic Traffic Shaper (DTS) [12]. Both STS and DTS use traffic shaper to regulate the packet transmitting time at every node such that the end-to-end communication delay can be regulated. The reason we choose these baselines is that STS tries to control the end-to-end communication delay by regulating traffic and DTS tries to regulate packets mainly based on the packet rate, which makes them comparable to DutyCon. Note that it has been demonstrated in [12] that STS and DTS outperform SYNC [4], which features a fixed duty cycle and PSM [31], which adapts a node's duty cycle in response to the network load observed at the MAC layer. Therefore, by outperforming STS and DTS, DutyCon also outperforms the baseline protocols used by them.

The difference between STS and DTS is that STS enforces a static traffic shaping algorithm according to an end-to-end delay requirement and source data rate, while DTS dynamically shapes the traffic only based on the source data rate. STS decomposes the end-to-end delay requirement and assigns the same delay requirement to each single hop. It also assigns a level to each node based on the distance from the destination. It then regulates the sending time at each node based on the local delay requirements, the node level, and source packet rate. Different from STS, DTS does not enforce the end-to-end delay requirement. It always sets the next packet sending time as the current packet sending time plus the inter-packet time at the source.

Every flow in the following experiments consists of 5 nodes with 100m distance between each hop in a single flow. Flows are randomly deployed in the topology, where each flow has intersection with some other flows. The source of each flow generates packets in a uniform distribution. The average packet rate is one packet every 3 seconds.

C. Different Delay Requirements

In this set of experiments, we use 3 end-to-end communication flows. We want to test the average end-to-end delays of DutyCon and baselines under different end-to-end delay requirements. Because only the baseline STS enforces an end-to-end delay requirement, we compare the performance of our protocol only to STS.

Figure 5 shows the average end-to-end delay performance under different delay requirements. We can see that when the delay requirement is loose (from 3s to 6s), DutyCon can control the average end-to-end delay very close to the desired value. The baseline STS does not have effective control of the average end-to-end delay . When the end-to-end delay requirement is tight (1s and 2s), the average end-to-end delays of both DutyCon and STS are longer than the delay requirement. However, DutyCon performs better than STS in these two cases.

The reason for this result is when the end-to-end delay requirement is loose, DutyCon controls the average delay of each hop to converge to the single-hop delay requirement, such that the average end-to-end delay can be controlled. When the desired end-to-end delay is tight, the delay requirements of each single-hop are too tight such that the controllers of some hops become saturated. When saturation happens, nodes cannot wake up more often. This is because the minimum sleep period of each hop is bounded by the minimum one-hop transmission delay. The saturation of the controller leads to undesired long end-to-end delay. The baseline STS statically estimates the sending time of each packet, based on the source packet rate and the single-hop delay requirement. The randomness of packet generation leads to the inaccurate estimation of the sending time at each hop. Therefore, the end-to-end delay is not well controlled to the desired value. Moreover, for tight end-to-end delay requirements, STS cannot give a good end-to-end delay guarantee. This is because all the nodes of the same level in STS wake up at the same time and try to send packet, which leads to a higher interference degree in the network.

Figure 6 is the result of the average duty cycle under different end-to-end delay requirements. When the desired delay is tight (1s and 2s), the wakeup time of DutyCon is only half of that of STS. This is because with a tight single-hop delay requirement, the estimation of the sending time by STS is always earlier than the actual packet ready time at the sender, resulting in a long wakeup time on the receiver side. When the delay requirement is 4s and 5s, STS has less duty cycle than DutyCon. However, the two corresponding points in Figure 5 show a longer delay than the requirements. When the delay requirement is 6s, STS stays awake for a longer time than DutyCon, which leads to a shorter end-to-end delay in Figure 5. The results indicate that when the delay requirements is loose, STS can violate the end-to-end delay requirement; while in other time it consumes unnecessarily high energy for an unnecessarily short delay. Different from STS, DutyCon achieves a good trade off between energy and end-to-end delay by controlling the delay close to the requirement.

D. Different Flow Numbers

In this section, we evaluate the performance of DutyCon and the two baseline protocols by varying the number of flows in

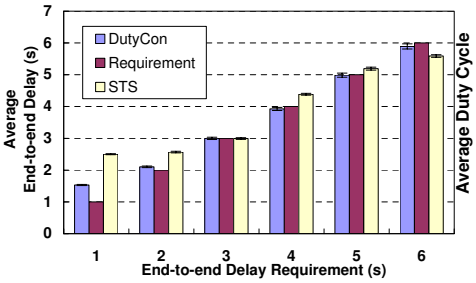

Fig. 5. End-to-end delay under different delay requirements.

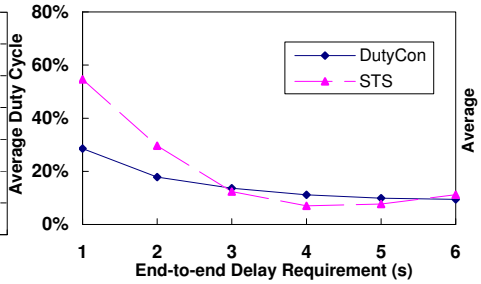

Fig. 6. Average duty cycle under different end-to-end delay requirements.

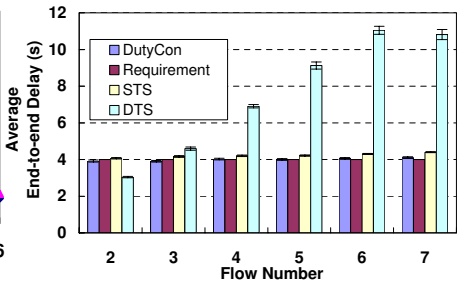

Fig. 7. End-to-end delay under different number of flows.

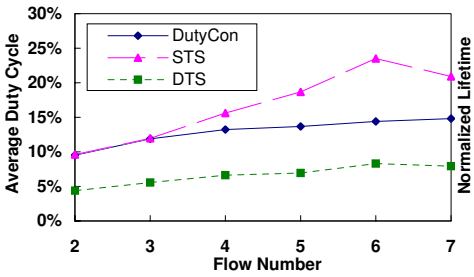

Fig. 8. Average duty cycle under different number of flows.

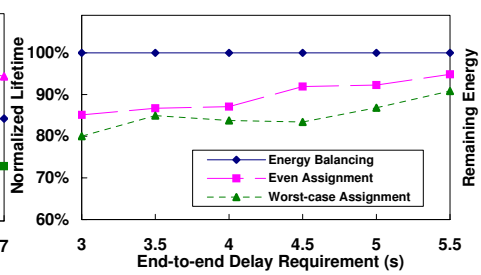

Fig. 9. Average network lifetime normalized to the maximum under the three single-hop delay requirement assignment schemes.

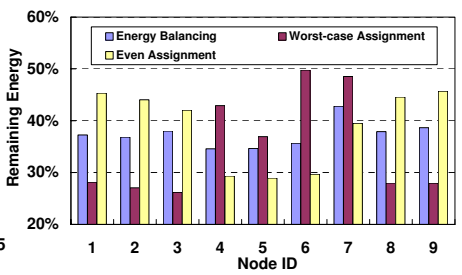

Fig. 10. Remaining energy of each node under the three single-hop delay requirement assignment schemes.

the network. With more flows, there will be additional interference in the network. The end-to-end delay requirement of each flow is set to 4 seconds.

Figure 7 shows the average end-to-end delay performance. DutyCon can always control the average end-to-end delay close to the requirements. The delay of STS is always higher than the requirement, especially when there are more flows in the network. This is because when the number of flows increases, the number of nodes at the same level increases, leading to a higher degree of interference when they wake up together and try to send packets. DTS has a shorter delay only when the flow number is 2. When the flow number is more than 2, DTS shows a significantly higher delay when compared with DutyCon and STS. The reason is that DTS always estimates the next packet sending time as the previous packet time plus the packet interval at the source. This inaccurate estimation causes the packets to be stacked in the queue, such that all the packets suffer significant queuing delays. Among these three schemes, DutyCon shows the best end-to-end delay as it utilizes the feedback control scheme to dynamically adapt to network environment changes. Furthermore, it features the queuing delay adaptation scheme to adapt the duty cycle to the queuing delay, especially when the number of flow is large. More flows incur more interference in the network, which causes more packets stacked in the queue. However, with the queuing delay adaptation scheme, DutyCon can handle this problem significantly better than the two baselines.

Figure 8 shows the average duty cycles of the three protocols with different numbers of flows in the network. DTS has a lower duty cycle in all the situations. This is because the estimation of the next packet sending time is always later than the packet ready time in most cases. Most packets are stacked in the queue, so that when the receiver wakes up, there is always a packet ready for sending at the sender. This leads to a short wakeup time at the receiver. STS shows a higher duty cycle when there are more flows in the network. The higher degree of interference caused by additional flows leads to the longer waking up time in STS. DutyCon has a moderate duty cycle compared with the two baselines.

E. Benefit of Energy Balancing

In this section, we evaluate the network lifetime and energy balancing on each node under three single-hop delay assignment schemes. The first one is *Worst-case Assignment*, which estimates the worst-case PRR, as introduced in Section VI. The second one is the *Energy Balancing* scheme proposed in Section VI. In this scheme, we apply the energy consumption balancing scheme to update the delay requirements of each hop periodically, at a rate of every 500 seconds. The third one is called *Even Assignment*, where we assign the delay requirements evenly to each hop of the flow.

Eleven nodes are used to construct a 10-hop single flow in this experiment. Two additional pairs of nodes, forming two single-hop communication links, are placed close to node 5 to introduce interferences to the single flow. The interference signal can be received by nodes 4, 5, and 6. The average packet interval at the source of the end-to-end flow is set to 5 seconds and the packet interval at the interference pairs is set to 1 second. We repeat this end-to-end transmission experiment with different end-to-end delay requirements. The experiment runs until one of the nodes in the flow exhausts all of its initial energy. The entire simulation time of each experiment is considered as the life time of the single-flow network.

Figure 9 shows the average network lifetime under each scheme. The values are normalized to the longest lifetime of all the three schemes. The experiment is conducted repeatedly with different delay requirements. The *Energy Balancing* scheme always has the longest network lifetime for all the different delay requirements. When the end-to-end delay requirement is 3 seconds, the *Worst-case Assignment* only

achieves an 80% lifetime, normalized to the lifetime of the *Energy Balancing* scheme. The results demonstrate that the single-hop delay assignment scheme is critical to the lifetime of the network when employing DutyCon.

We then examine the distribution of the remaining energy among all the nodes. In this experiment, the total number of packets that the end-to-end flow needs to transmit is fixed. Figure 10 shows the remaining energy of each node in the single flow after the completion of all the transmissions. When using *Even Assignment*, the remaining energy of nodes 4, 5, and 6 is significantly lower than that of the other nodes. This is because the *Even Assignment* scheme does not consider the different network conditions in different areas of the network. With the same single-hop delay requirement, the interference around nodes 4, 5, and 6 makes the three nodes wake up more often, leading to additional energy consumption. There is a 15% difference in the remaining energy between the most and the least remaining energy from all the nodes. The second scheme, *Worst-case Assignment*, tends to pessimistically assign an unnecessarily long delay requirement to the the hops that are estimated offline to have more interference. Therefore, nodes 4, 5, 6, and 7 have a lower duty cycle than all other nodes, which leads to their lower energy consumption. The difference between the most and least remaining energy under *Worst-case Assignment* is approximately 25% in this experiment. The *Energy Balancing* scheme has the best energy balancing by periodically updating the delay requirements for each hop, based on the current network conditions. As a result, the difference between the most and least remaining energy is only approximately 8% under *Energy Balancing*.

VIII. Conclusion

In many WSN applications with user-specified delay requirements, the duty cycle of every node should be individually tuned dynamically based on the network conditions to achieve the desired end-to-end delay guarantees. In this paper, we have proposed DutyCon, a dynamic duty cycle control approach that decomposes the end-to-end delay guarantee problem into a set of single-hop delay guarantee problems along each data flow in the network. We then formulate the single-hop delay guarantee problem as a dynamic feedback control problem. DutyCon is designed rigorously based on well-established feedback control theory for analytic assurance of delay control accuracy and system stability. DutyCon also features a queuing delay adaptation scheme that adapts the node duty cycle to unpredictable packet rates, as well as a novel energy balancing approach that extends the network lifetime by dynamically adjusting the delay requirements allocated to each hop in a data flow. Both empirical results on a hardware testbed and extensive simulations demonstrate that DutyCon can effectively achieve the desired tradeoff between end-to-end delay and energy conservation. Our simulation results also show that DutyCon outperforms two baseline sleep scheduling protocols by having more energy savings while meeting the end-to-end delay requirements.

References

[1] T. He, P. Vicaire, T. Yan, Q. Cao, G. Zhou, L. Gu, L. Luo, R. Stoleru, J. A. Stankovic, and T. F. Abdelzaher, "Achieving long-term surveillance in vigilnet," in *IEEE INFOCOM*, April 2006.

[2] "Crossbow technology, telosb mote." [Online]. Available: http://www.xbow.com/Products/productdetails.aspx?sid=252

[3] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *IPSN*, April 2005.

[4] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *IEEE Infocom*, 2002.

[5] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks." in *SenSys*, 2003.

[6] I. Rhee, A. Warrier, J. Min, and L. Xu., "Drand: distributed randomized tdma scheduling for wireless ad-hoc networks," in *MobiHoc*, 2006.

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

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

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

[10] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin, "Temporal properties of low power wireless links: modeling and implications on multi-hop routing," in *MobiHoc*, 2005.

[11] G. F. Franklin, M. L. Workman, and D. Powell, *Digital Control of Dynamic Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[12] O. Chipara, C. Lu, and G.-C. Roman, "Efficient power management based on application timing semantics for wireless sensor networks," in *ICDCS*, 2005.

[13] M. Caccamo, L. Y. Zhang, and L. Sha, "An implicit prioritized access protocol for wireless sensor networks," in *RTSS*, 2002.

[14] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He, "RAP: A real-time communication architecture for large-scale wireless sensor networks," in *RTAS*, 2002.

[15] T. He, J. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A stateless protocol for real-time communication in sensor networks," in *ICDCS*, 2003.

[16] K. Karenos and V. Kalogeraki, "Real-time traffic management in sensor networks," in *RTSS*, 2006.

[17] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks," in *IPDPS*, 2004.

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

[19] R. W. Ha, P.-H. Ho, and X. S. Shen, "Cross-layer application-specific wireless sensor network design with single-channel csma mac over sense-sleep trees," *Computer Communications*, vol. 29, pp. 3425–3444, 2006.

[20] Y. Gu, T. He, M. Lin, and J. Xu, "Spatiotemporal delay control for low-duty-cycle sensor networks," in *RTSS*, 2009.

[21] Y. Min, L. T. Yang, F. Wang, and W. Wang, "Dynamic sleeping algorithm based on ahp for wireless sensor networks," *Future Generation Communication and Networking*, vol. 2, pp. 387–392, 2008.

[22] X. Ning and C. Cassandras, "Optimal dynamic sleep time control in wireless sensor networks," in *CDC*, 2008.

[23] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *IEEE Control Systems Magazine*, vol. 23, p. 2003, 2003.

[24] S. Lin, J. Zhang, G. Zhou, L. Gu, J. A. Stankovic, and T. He, "Atpc: adaptive transmission power control for wireless sensor networks," in *SenSys*, 2006.

[25] C. Merlin and W. Heinzelman, "Duty cycle control for low-power-listening mac protocols," in *MASS*, 2008.

[26] H. K. Le, D. Henriksson, and T. Abdelzaher, "A control theory approach to throughput optimization in multi-channel collection sensor networks," in *IPSN*, 2007.

[27] X. Wang, X. Wang, F. Xing, G. Xing, and N. Jha, "Flow-based real-time communication in multi-channel wireless sensor networks," in *EWSN*, 2009.

[28] M. Maimour, "Maximally radio-disjoint multipath routing for wireless multimedia sensor networks," in *WMuNep*, 2008.

[29] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *SenSys*, 2003.

[30] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *SenSys*, 2003.

[31] "Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Standard 802.11*.