# Delay-Bounded MAC with Minimal Idle Listening for Sensor Networks

Yang Peng, Zi Li, Daji Qiao and Wensheng Zhang
Iowa State University, Ames, IA 50011
{yangpeng, zili, daji, wzhang}@istate.edu

***Abstract*—This paper presents a new receiver-initiated sensor network MAC protocol, called CyMAC, which has the following unique features. It reduces the idle listening time of sensor nodes via establishing rendezvous times between neighbors, provides the desired relative delay bound guarantee for data delivery services via planning the rendezvous schedules carefully, and adjusts the sensor nodes' duty cycles dynamically to the varying traffic condition. More importantly, CyMAC achieves the above goals without requiring time synchrony between sensor nodes. We have implemented and evaluated CyMAC in both TinyOS and the ns-2 simulator. Experimental and simulation results show that, comparing with RI-MAC – a state-of-the-art sensor network MAC protocol, CyMAC can always guarantee the desired delay bound for data delivery services and yields a lower duty cycle under reasonable delay requirements.**

## I. Introduction

Wireless sensor networks should be energy efficient in order to operate for a long time. When a sensor node has its radio turned on, it operates at a similar power consumption level regardless whether it is transmitting, receiving or idle listening [1]. Hence, numerous MAC protocols have been proposed to reduce the idle listening time of a sensor node, which has been found to contribute substantially to a sensor node's total energy consumption [2], [3].

### A. Related Work

Most of the existing MAC protocols are either synchronous or asynchronous. Representative synchronous protocols such as S-MAC [1], T-MAC [4], RMAC [5] and DW-MAC [6] require neighbor nodes to be time-synchronized. They align the active and sleep intervals of neighbor nodes, which wake up only during the common active time intervals to exchange packets. Since the active intervals usually are short, substantial energy can be saved. However, strictly synchronizing the clocks of neighbor nodes imposes high overhead, and the aligned and short active intervals can cause congestion when multiple flows cross the same node.

Asynchronous protocols such as B-MAC [7], WiseMAC [8], X-MAC [9] and RI-MAC [10] decouple the duty cycle schedules of different nodes and thus eliminate the overhead for synchronization. B-MAC, WiseMAC and X-MAC are sender-initiated preamble-based protocols which employ the low power listening technique. Particularly, B-MAC requires a sender to transmit a preamble longer than the sleep interval of its receiver to signal the receiver. WiseMAC shortens the preamble length by requiring a sender to learn the duty cycle schedule of its receiver and start a preamble shortly before the receiver wakes up. X-MAC improves B-MAC by replacing the long preamble with a sequence of short, strobed preambles. Nevertheless, these protocols are optimized mainly for light traffic conditions. In the scenarios of bursty or high traffic load, which can be caused by convergecast [11], correlated events [12] and data aggregation [13], the preambles may congest the channel and block data transmissions. Hybrid protocol such as SCP [14] combines a synchronous protocol with asynchronous low power listening but suffers the same clock synchronization overhead as synchronous protocols.

To work under a wider range of traffic conditions, RI-MAC [10] adopts a receiver-initiated beacon-based strategy. Each node periodically wakes up and sends out a short beacon to explicitly notify its neighbors that it is ready to receive data. When a node has data to transmit, it wakes up and waits for a beacon from its receiver. Once such a beacon is received, it starts sending the data. Compared to the sender-initiated preamble-based protocols, RI-MAC uses shorter and less frequent beacons which consume less bandwidth, and its receiver-initiated nature allows more efficient collision resolution. However, RI-MAC has the following limitations. A sender needs to remain awake after a data packet arrives, till the receiver wakes up to receive the packet, potentially wastes a lot of time on idle listening. Also, a receiver sends out beacons at a fixed time interval on average and does not adapt to changes of traffic pattern.

### B. Motivations and Contributions

To further reduce idle listening and improve the energy efficiency of sensor networks, we propose in this paper a new MAC protocol called *CyMAC*. Similar to RI-MAC, CyMAC is a receiver-initiated beacon-based protocol. The difference is that CyMAC reduces the idle listening time significantly through establishing rendezvous times between sender and receiver. In addition, rendezvous schedules are adaptive to the changes of traffic condition so that sender and receiver can operate with minimal duty cycles while a certain desired delay bound for data delivery services can still be guaranteed. More importantly, CyMAC achieves the above goals without requiring clock synchrony between sensors. It functions properly as long as the desired delay bound is less stringent than the degree of clock asynchrony.

CyMAC targets to provide *relative delay bound* [15] guarantee for sensor data delivery services, which is defined as the ratio of the data delivery delay to the average data arrival interval. For example, if data packets arrive every 100 seconds and the delivery delay of a data packet is 10 seconds, the relative delay is 10%. This is in contrast to the absolute delay bound that usually is provided with a fixed beacon interval (e.g., in RI-MAC) so that the delivery delay of a data packet can be guaranteed less than the beacon interval. For sensor network applications, a relative delay bound could be more meaningful and important than an absolute delay bound. For example, the same delivery delay of one second may have different effects on two different sensor network applications: one with a data arrival interval of one second and the other with a data arrival interval of 100 seconds. The former situation could be far worse than the latter, since by the time when a data packet is delivered, it has become obsolete because a newer data packet has arrived. Relative delay bound may help sensor nodes conserve energy too. For example, if a 10% relative delay bound is acceptable, when the data arrival interval increases from 10 to 100 seconds, the number of beacons sent by the receiver and hence the energy consumed by the receiver can be reduced by an order of magnitude.

The contributions of this work are summarized as follows:

- We propose a new receiver-initiated MAC protocol, called CyMAC, for sensor networks. CyMAC attempts to minimize idle listening and hence duty cycles of sensor nodes via establishing rendezvous times between neighbors. It is adaptive to the changes in traffic condition, and can guarantee desired relative delay bound for sensor data delivery services under various traffic conditions. Different from existing synchronous MAC protocols, CyMAC does not require clock synchrony between sensor nodes.
- We have implemented CyMAC in TinyOS and evaluated it with small-scale experiments. We have also implemented it in the ns-2 simulator for evaluation in large-scale networks.
- Extensive experiments and simulations have demonstrated that CyMAC can always guarantee the desired delay bound, and has a lower duty cycle than RI-MAC in most cases except when the required delay bound is very tight. In this case, CyMAC can still provide the delay bound guarantee at the cost of having a slightly higher duty cycle than RI-MAC.

### C. Organization

In the rest of the paper, Section II presents the design details of CyMAC, which is followed by the description of CyMAC implementation in TinyOS. Experiment and simulation results are presented in Section III. Section IV discusses the future work and concludes the paper.

## II. CyMAC Design

In the following, we give an overview of the proposed CyMAC protocol for sensor networks.

*1) CyMAC is a receiver-initiated MAC protocol but with minimal idle listening time at the sender side.* Similar to RI-MAC, the data exchange between CyMAC sender and receiver is initiated by the receiver with a beacon. However, different from RI-MAC which requires the sender to remain awake (upon a data packet arrival) and listen idly till the beacon arrives, CyMAC only requires the sender to wake up at pre-scheduled rendezvous times to communicate with the receiver, thus reducing the idle listening time significantly.

*2) CyMAC provides delay-bounded data delivery services.* A unique feature of CyMAC is its ability to adjust the duty cycles and rendezvous schedules of sensor nodes to provide the desired relative delay bound to data delivery services.

*3) CyMAC adjusts the sensor nodes' duty cycles dynamically to the varying traffic condition.* Another unique feature of CyMAC is dynamic duty cycling. When the traffic is light, CyMAC nodes sleep more and send fewer beacons to conserve more energy, while when the traffic is heavy, CyMAC nodes wake up more often to interact with each other so as to provide the desired delay bound.

*4) CyMAC does not require clock synchrony between sensor nodes:* Different from existing synchronous MAC protocols, CyMAC does not require clock synchrony between sensor nodes nor synchronization protocols executed on sensor nodes. CyMAC functions properly as long as the desired delay bound is less stringent than the degree of clock asynchrony between neighbor nodes. Section II-C discusses in detail how CyMAC handles clock asynchrony issues.

Next, we describe the design of CyMAC in detail. Table I lists the variables maintained at each CyMAC node.

TABLE I
Variables Maintained at Each CyMAC Node

| Variable | | Meaning |
|---|---|---|
| For each sender $i$ | $T_{\text{LAST},i}$ | arrival time of the last received data packet from sender $i$ |
| | $T_{\text{BEACON},i}$ | latest time to serve sender $i$ (by sending a beacon) in order to satisfy the delay bound |
| $T_{\text{BEACON}} = \min_i\,(T_{\text{BEACON},i})$ | | scheduled next beacon time |
| For each receiver $j$ | $T_{\text{LISTEN},j}$ | scheduled next listen time for receiver $j$ |
| | $\text{DONE}_j$ | the set of packets that (i) have failed all transmission attempts (ii) arrive after the last successfully-delivered packet; the last successfully-delivered packet is also included in set $\text{DONE}_j$ |
| | $\text{WAIT}_j$ | the set of packets waiting to be transmitted |
| For each packet $x$ in $\text{WAIT}_j$ or $\text{DONE}_j$ | $T_{\text{arrv}}(x)$ | arrival time of packet $x$ |
| | $D(x)$ | delay between $T_{\text{arrv}}(x)$ and when $x$ is transmitted |
| | $\theta(x)$ | updated estimate of mean of packet arrival interval |
| | $\delta(x)$ | updated estimate of variance of packet arrival interval |

### A. Receiver Behavior

The operation flowchart of a CyMAC receiver is shown in Fig. 1. In CyMAC, the receiver wakes up at the scheduled beacon time $T_{\text{BEACON},i}$ to interact with sender $i$ by sending a beacon and then waiting for a short dwell time[1]. As shown in the flowchart, if a new packet is received successfully, the receiver records the packet information, updates its estimates of the data traffic, and schedules the next beacon time using the $I_{\text{allow}}$ information piggybacked in the packet by the sender (which tells the receiver when the next beacon should be sent); otherwise, it schedules the next beacon time for sender $i$ based on (i) $T_{\text{BEACON},i}$; (ii) $T_{\text{LAST},i}$ – the arrival time of the last received data packet from sender $i$; and (iii) $\mu$ – the desired relative delay bound over a single hop. Note that, since a receiver may serve multiple senders, it performs the above routine for all senders and informs everyone of its very next scheduled beacon time: $T_{\text{BEACON}} = \min_i(T_{\text{BEACON},i})$. This

[1]This short dwell time is platform dependent. In our implementation of CyMAC on MicaZ motes, it is set to 17.5ms.

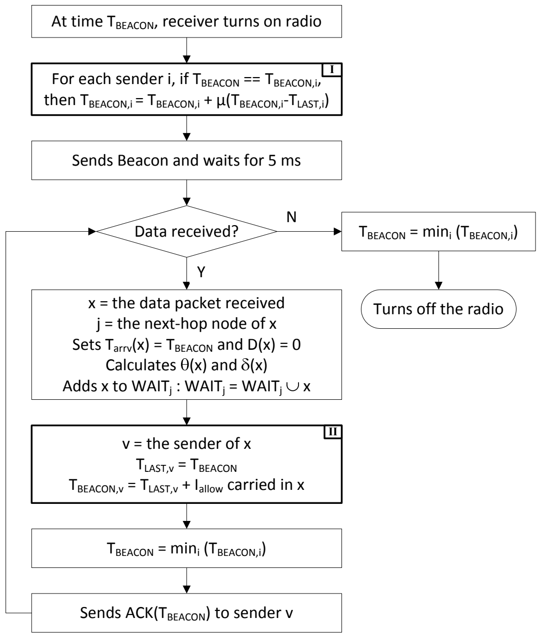

At time $T_{BEACON}$, receiver turns on radio
For each sender i, if $T_{BEACON} == T_{BEACON,i}$, then $T_{BEACON,i} = T_{BEACON,i} + \mu(T_{BEACON,i}-T_{LAST,i})$
I
Sends Beacon and waits for 5 ms
Data received?
N
Y
$T_{BEACON} = \min_i (T_{BEACON,i})$
Turns off the radio
x = the data packet received
j = the next-hop node of x
Sets $T_{arrv}(x) = T_{BEACON}$ and $D(x) = 0$
Calculates $\theta(x)$ and $\delta(x)$
Adds x to $WAIT_j$ : $WAIT_j = WAIT_j \cup x$
II
v = the sender of x
$T_{LAST,v} = T_{BEACON}$
$T_{BEACON,v} = T_{LAST,v} + I_{allow}$ carried in x
$T_{BEACON} = \min_i (T_{BEACON,i})$
Sends ACK($T_{BEACON}$) to sender v

Fig. 1. Operation flowchart of a CyMAC receiver.

way, a sender may be able to forward a packet that arrives earlier than expected to the receiver opportunistically at an earlier beacon time that was scheduled for other senders, thus reducing the delivery delay further.

*1) Online Traffic Estimation:* Upon arrival of a data packet $x$, the receiver updates its estimate of the mean of data arrival interval as:

$$\theta(x) = \alpha(x)\theta(x') + (1 - \alpha(x))\theta_{\text{new}}(x), \tag{1}$$

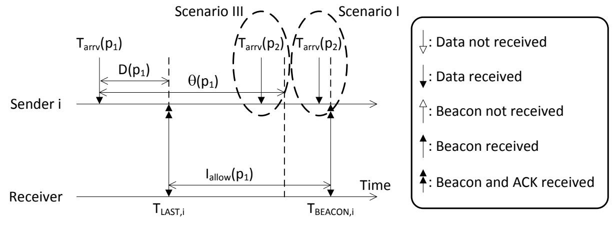
where $x'$ is the last successfully-received data packet prior to $x$ and has the same next-hop node as packet $x$. $\theta_{\text{new}}(x) = T_{\text{arrv}}(x) - T_{\text{arrv}}(x')$ is the new sample mean and $\alpha(x)$ is the smoothing factor: $\alpha(x) = 2^{\frac{-\theta_{\text{new}}(x)}{10\cdot\theta(x')}} \cdot 0.9$. The reason for choosing such a smoothing factor for estimating the mean of data arrival interval is that, a larger $\theta_{\text{new}}(x)$ value implies that the previously estimated mean ($\theta(x')$) has become more obsolete, and hence a larger weight should be given to the new sample. For example, if $\theta_{\text{new}}(x) = 10 \cdot \theta(x')$, meaning that packet $x$ arrives much later after the previous packet $x'$ (10 times the mean arrival interval), then a larger weight ($0.55 = 1 - \alpha(x)$) is given to the new sample.

The receiver also updates its estimate of the variance of data arrival interval, but with a fixed smoothing factor:

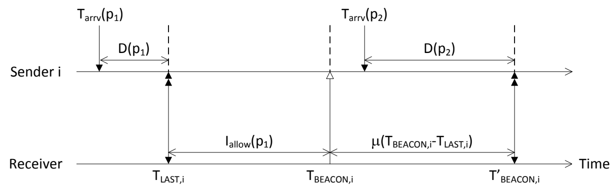
$$\delta(x) = \beta\delta(x') + (1 - \beta)\delta_{\text{new}}(x), \tag{2}$$

where $\delta_{\text{new}}(x) = |\theta_{\text{new}}(x) - \theta(x')|$ is the new sample variance and $\beta = 0.9$. This is because a late arriving packet (i.e., a larger $\theta_{\text{new}}(x)$ value) may skew the calculation of $\delta_{\text{new}}(x)$; hence we opt to not give a larger weight to the new sample in the estimation to avoid undesired complication.

*2) Relative Delay Bound Guarantee:* One of the key design goals of CyMAC is to provide delay-bounded data delivery services, meaning that if all packets (beacon, data and ACK)



(a) Scenario I: packet $p_2$ arrives between $T_{\text{arrv}}(p_1) + \theta(p_1)$ and $T_{\text{BEACON},i}$. Scenario III: packet $p_2$ arrives before $T_{\text{arrv}}(p_1) + \theta(p_1)$.



(b) Scenario II: packet $p_2$ arrives after $T_{\text{BEACON},i}$.

Fig. 2. Example scenarios to illustrate how the desired delay bound is satisfied with CyMAC.

are transmitted successfully, the delivery delay of a data packet $x$ over a single hop is

$$D(x) \leqslant \mu \max\{\theta(x), T_{\text{arrv}}(x) - T_{\text{arrv}}(x')\}, \tag{3}$$

where $x'$ is the last successfully-received data packet prior to $x$ and has the same next-hop node as packet $x$. $\mu$ is the desired relative delay bound over a single hop. In practice, a sensor network application often specifies its desired delay bound in terms of end-to-end delay ($\mu_{\text{e2e}}$). Let $\xi$ denote the hop-count diameter of the sensor network, we conservatively translate the application-specified end-to-end delay bound $\mu_{\text{e2e}}$ to the hop-by-hop relative delay bound $\mu$ as follows:

$$\mu = (1 + \mu_{\text{e2e}})^{1/\xi} - 1. \tag{4}$$

To illustrate how CyMAC guarantees Eq. (3), we present a few example scenarios in Fig. 2. Here, we assume that a CyMAC receiver only has one sender (sender $i$) to receive data packets from. As shown in the figure, after packet $p_1$ is delivered successfully from sender $i$ to the receiver at time $T_{\text{LAST},i}$, the receiver schedules its next beacon time to

$$T_{\text{BEACON},i} = T_{\text{LAST},i} + I_{\text{allow}}(p_1), \tag{5}$$

where $I_{\text{allow}}(p_1)$ is the information piggybacked in packet $p_1$ and set by sender $i$. For a relative delay bound of $\mu$, let us set $I_{\text{allow}}(p_1)$ to

$$I_{\text{allow}}(p_1) = (1 + \mu)\theta(p_1) - D(p_1). \tag{6}$$

Then, depending on the arrival time of the next data packet $p_2$, there are three possible scenarios:

- Scenario I: $T_{\text{arrv}}(p_1) + \theta(p_1) \leqslant T_{\text{arrv}}(p_2) \leqslant T_{\text{BEACON},i}$. In this case, packet $p_2$ arrives before the scheduled beacon time $T_{\text{BEACON},i}$ but after $T_{\text{arrv}}(p_1) + \theta(p_1)$, as shown in Fig. 2(a). The delivery delay for packet $p_2$ is then:

$$\begin{aligned} D(p_2) &= T_{\text{BEACON},i} - T_{\text{arrv}}(p_2) \\ &= T_{\text{LAST},i} + (1 + \mu)\theta(p_1) - D(p_1) - T_{\text{arrv}}(p_2) \\ &= T_{\text{arrv}}(p_1) + (1 + \mu)\theta(p_1) - T_{\text{arrv}}(p_2) \\ &\leqslant T_{\text{arrv}}(p_1) + (1 + \mu)\theta(p_1) - (T_{\text{arrv}}(p_1) + \theta(p_1)) \\ &= \mu\theta(p_1) \leqslant \mu \max\{\theta(p_2), T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1)\}. \end{aligned} \tag{7}$$

Therefore, the desired delay bound is guaranteed.

- Scenario II: $T_{\text{arrv}}(p_2) > T_{\text{BEACON},i}$. In this case, packet $p_2$ arrives after the scheduled beacon time, as shown in Fig. 2(b). As a result, the receiver schedules the next beacon time to

$$T'_{\text{BEACON},i} = T_{\text{BEACON},i} + \mu(T_{\text{BEACON},i} - T_{\text{LAST},i}). \quad (8)$$

If packet $p_2$ arrives before $T'_{\text{BEACON},i}$, its delivery delay is bounded under the limit:

$$\begin{aligned} D(p_2) &= T'_{\text{BEACON},i} - T_{\text{arrv}}(p_2) < T'_{\text{BEACON},i} - T_{\text{BEACON},i} \\ &= T_{\text{BEACON},i} + \mu(T_{\text{BEACON},i} - T_{\text{LAST},i}) - T_{\text{BEACON},i} \\ &= \mu(T_{\text{BEACON},i} - T_{\text{LAST},i}) < \mu(T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1)) \\ &= \mu \max\{\theta(p_2), T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1)\}. \end{aligned} \quad (9)$$

If packet $p_2$ arrives after $T'_{\text{BEACON},i}$, a similar analysis can be applied to show that the desired delay bound is always satisfied.

- Scenario III: $T_{\text{arrv}}(p_2) < T_{\text{arrv}}(p_1) + \theta(p_1)$. In this case, since packet $p_2$ arrives before $T_{\text{arrv}}(p_1) + \theta(p_1)$, as shown in Fig. 2(a), its delivery delay would be

$$\begin{aligned} D(p_2) &= T_{\text{BEACON},i} - T_{\text{arrv}}(p_2) \\ &= T_{\text{arrv}}(p_1) + (1+\mu)\theta(p_1) - T_{\text{arrv}}(p_2) \\ &> T_{\text{arrv}}(p_1) + (1+\mu)\theta(p_1) - (T_{\text{arrv}}(p_1) + \theta(p_1)) \\ &= \mu\theta(p_1) > \mu \max\{\theta(p_2), T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1)\}. \end{aligned} \quad (10)$$

This means that, for any packet that arrives within the estimated mean packet arrival interval, the delivery delay cannot be bounded under the desired limit. As a result, we may not be able to bound the average delivery delay (over all packets) under certain packet arrival distributions. One way to deal with this potential issue is to employ a more conservative approach by replacing $\theta$ with $(\theta - m\delta)$ in Eq. (6):

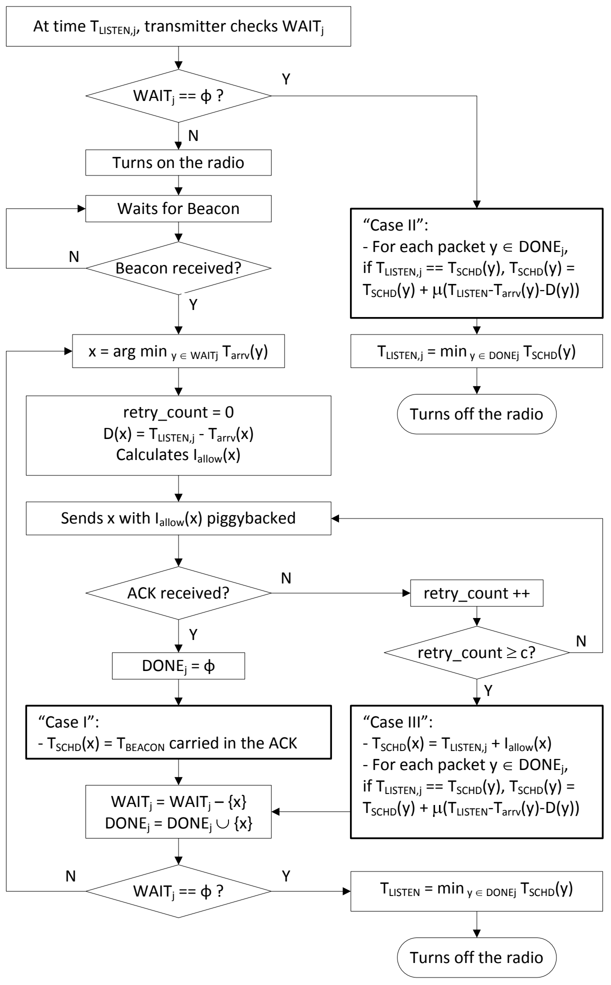
$$I_{\text{allow}}(p_1) = (1+\mu)(\theta(p_1) - m\delta(p_1)) - D(p_1), \quad (11)$$

where $m \geqslant 1$ and larger $m$ values may be used for more stringent delay requirements. This way, fewer packets would experience higher delay, and thus the average delivery delay may be bounded under the limit.

## B. Sender Behavior

The operation flowchart of a CyMAC sender is shown in Fig. 3. In CyMAC, the sender acts in a leading role. It schedules the rendezvous times with each receiver by calculating $I_{\text{allow}}$ and piggybacks such information in the packet transmissions to the receiver. For receiver $j$, the sender maintains two sets of packets (as listed in Table I): (i) DONE$_j$ – the set of packets that have failed all transmission attempts and arrive after the last successfully-delivered packet, which itself is also included in the set; and (ii) WAIT$_j$ – the set of packets waiting to be transmitted. It also maintains $T_{\text{LISTEN},j}$ – the next listen time for beacons from receiver $j$. At $T_{\text{LISTEN},j}$, the sender forwards all the packets in WAIT$_j$ to receiver $j$ with $I_{\text{allow}}$ information piggybacked in each packet.

*1) Rendezvous between Sender and Receiver:* As shown in Fig. 3, there are three different cases when the sender schedules its next listen time differently. CyMAC is able to guarantee rendezvous between sender and receiver in all three cases, which will be explained with the help of example scenarios given in Fig. 4.



At time T_LISTEN,j, transmitter checks WAIT_j
WAIT_j == ϕ ?
Y
N
Turns on the radio
Waits for Beacon
N
Beacon received?
Y
"Case II":
- For each packet y ∈ DONE_j,
if T_LISTEN,j == T_SCHD(y), T_SCHD(y) = T_SCHD(y) + μ(T_LISTEN-T_arrv(y)-D(y))
T_LISTEN,j = min y ∈ DONEj T_SCHD(y)
Turns off the radio
x = arg min y ∈ WAITj T_arrv(y)
retry_count = 0
D(x) = T_LISTEN,j - T_arrv(x)
Calculates I_allow(x)
Sends x with I_allow(x) piggybacked
ACK received?
N
retry_count ++
Y
retry_count ≥ c?
N
DONE_j = ϕ
Y
"Case I":
- T_SCHD(x) = T_BEACON carried in the ACK
"Case III":
- T_SCHD(x) = T_LISTEN,j + I_allow(x)
- For each packet y ∈ DONE_j,
if T_LISTEN,j == T_SCHD(y), T_SCHD(y) = T_SCHD(y) + μ(T_LISTEN-T_arrv(y)-D(y))
WAIT_j = WAIT_j – {x}
DONE_j = DONE_j ∪ {x}
N
WAIT_j == ϕ ?
Y
T_LISTEN = min y ∈ DONEj T_SCHD(y)
Turns off the radio

Fig. 3. Operation flowchart of a CyMAC sender with respect to receiver $j$.

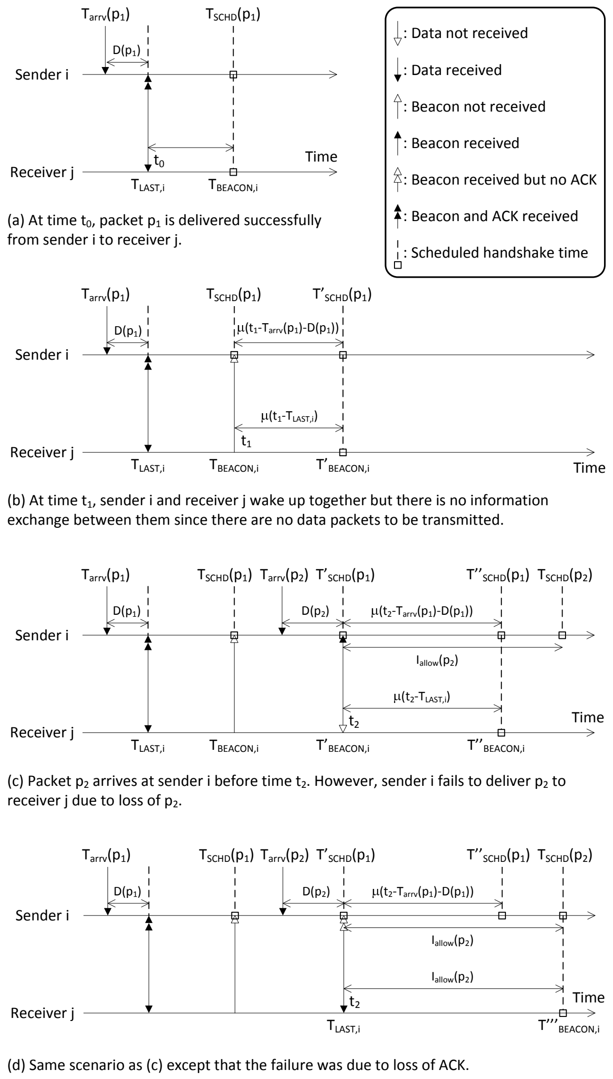
- *Case I: after a successful data packet delivery.* In this case, the sender sets the next listen time to $T_{\text{BEACON}}$ that is carried in the ACK. This case is illustrated in Fig. 4(a) where we assume that there is only one sender (sender $i$) and one receiver (receiver $j$). We can see that, after packet $p_1$ is delivered successfully at time $t_0$, both sender and receiver schedule to wake up together at $T_{\text{SCHD}}(p_1) = T_{\text{BEACON},i} \triangleq t_1$.
- *Case II: when there are no data packets to be transmitted.* Despite that there is no information exchange between sender and receiver in this case, CyMAC can still guarantee that sender and receiver wake up together at future time instances. Fig. 4(b) shows an example scenario when there are no data packets to be transmitted at time $t_1$. Sender $i$ schedules the next listen time to (according to Case II in Fig. 3 Flowchart)

$$T'_{\text{SCHD}}(p_1) = t_1 + \mu(t_1 - T_{\text{arrv}}(p_1) - D(p_1)), \quad (12)$$

and receiver $j$ schedules the next beacon time to (according to Box I in Fig. 1 Flowchart)

$$T'_{\text{BEACON},i} = t_1 + \mu(t_1 - T_{\text{LAST},i}). \quad (13)$$

These two time instances are indeed the same, mean-



Data not received
Data received
Beacon not received
Beacon received
Beacon received but no ACK
Beacon and ACK received
Scheduled handshake time
Sender i
Receiver j
Time
(a) At time t0, packet p1 is delivered successfully from sender i to receiver j.
(b) At time t1, sender i and receiver j wake up together but there is no information exchange between them since there are no data packets to be transmitted.
(c) Packet p2 arrives at sender i before time t2. However, sender i fails to deliver p2 to receiver j due to loss of p2.
(d) Same scenario as (c) except that the failure was due to loss of ACK.

Fig. 4. Example scenarios to illustrate how CyMAC guarantees rendezvous between sender and receiver.

ing that sender and receiver will wake up together at $T'_{\text{SCHD}}(p_1) = T'_{\text{BEACON},i} \triangleq t_2$.

- *Case III: after a failed data packet delivery.* In the design, the sender assumes the data packet delivery is failed after retrying for $c$ times ($c$ is a configurable system parameter as the retry count threshold) without receiving an ACK from the receiver. This is the most complicated case as the sender is unsure whether the failure was due to loss of data packet or loss of ACK, when the receiver behaves differently. These two scenarios are illustrated in Figs. 4(c) and (d), where at time $t_2$ the receiver schedules the next beacon time to (loss of data packet; Box I in Fig. 1 Flowchart)

$$T''_{\text{BEACON},i} = t_2 + \mu(t_2 - T_{\text{LAST},i}), \tag{14}$$

and (loss of ACK; Box II in Fig. 1 Flowchart)

$$T'''_{\text{BEACON},i} = t_2 + I_{\text{allow}}(p_2), \tag{15}$$

respectively. In order to guarantee rendezvous between sender and receiver, CyMAC requires the sender to wake up at both time instances. To do so, the sender updates $T_{\text{SCHD}}$ for all packets in set DONE and listen at all the updated $T_{\text{SCHD}}$ time instances. In the example scenarios shown in Figs. 4(c) and (d), since sender $i$ now has DONE$_j$ = $\{p_1, p_2\}$, it will listen at both

$$T''_{\text{SCHD}}(p_1) = t_2 + \mu(t_2 - T_{\text{arrv}}(p_1) - D(p_1)) \tag{16}$$

and

$$T'_{\text{SCHD}}(p_2) = t_2 + I_{\text{allow}}(p_2), \tag{17}$$

which match $T''_{\text{BEACON},i}$ and $T'''_{\text{BEACON},i}$, respectively.

*2) Minimal Idle Listening Time:* A major difference between CyMAC and RI-MAC is how a sender behaves upon a data packet arrival. In RI-MAC, a sender turns on the radio immediately after a data packet arrives, idly listening till it receives a beacon from the receiver. In comparison, a CyMAC sender only turns on the radio at scheduled listen times for possible interactions with receivers. So if a data packet arrives before the next scheduled listen time, the packet will be inserted into set WAIT but the radio won't be turned on till the scheduled listen time. This way, the idle listening time is reduced drastically.

*3) Dynamic Duty Cycling:* Another unique feature of CyMAC is that sensor nodes adjust their duty cycles dynamically to the varying traffic condition. When the traffic is light, sensor nodes sleep more and send less beacons to conserve more energy, while when the traffic is heavy, sensor nodes wake up more often to interact with each other so as to provide the desired delay bound.

Fig. 5 shows the behavior of CyMAC nodes when the network turns idle (i.e., no more new data packets) after a packet is delivered successfully at $T_{\text{LAST}}$. As shown in the figure, the $k$-th ($k \geqslant 1$) rendezvous time after $T_{\text{LAST}}$ will be scheduled at $T_{\text{LAST}} + (1+\mu)^{i-1}\phi$, according to Case II in the sender flowchart and Box I in the receiver flowchart. For example, if $T_{\text{LAST}}$ = 0 second, $\phi$ = 1 second and $\mu$ = 50%, the future rendezvous times will be at approximately $\{1, 1.5, 2.3, 3.4, 5.1, 7.6, 11.4, 17.1, \cdots\}$ seconds. This procedure goes on till new data packets arrive which will direct CyMAC nodes to reset their duty cycles based on their updated estimates of the data traffic. This shows that CyMAC nodes are able to adjust quickly to the varying traffic condition and operate in ultra low duty cycles when the traffic is light.

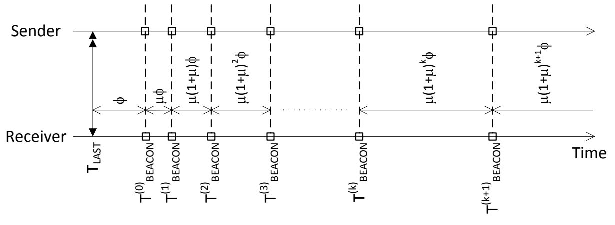


Fig. 5. Dynamic duty cycling with CyMAC.

## C. Effects of Time Asynchrony

In a practical sensor network, sender and receiver nodes are inevitably asynchronous. Typically, clocks of sensor nodes differ for two reasons: *clock skew* that is simply the initial difference between clocks, and *clock drift* that refers to different clocks counting time at slightly different rates, which results in varying clock skews over time. In general, clock asynchrony between sender and receiver nodes can be described with the following equation:

$$t_r = a \times t_s + b, \tag{18}$$

where $t_s$ is the time instance at the sender, $t_r$ is the corresponding time instance at the receiver, and $a$ and $b$ represent the clock drift and the clock skew, respectively. In this section, we analyze the effects of clock asynchrony on CyMAC performance, and discuss how we enhance CyMAC to deal with these issues.

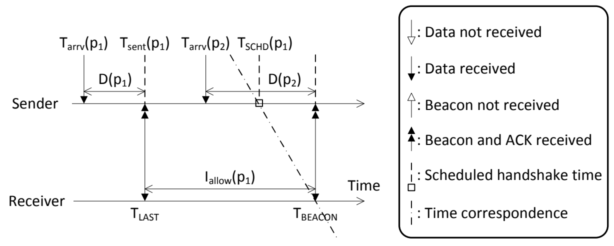
*1) $a < 1$:* In this case, the sender clock counts time at a faster rate than the receiver clock, as shown in Fig. 6(a). After the sender delivers a packet $p_1$ successfully to the receiver, both sender and receiver know that $T_{\text{sent}}(p_1)$ on the sender clock corresponds to $T_{\text{LAST}}$ on the receiver clock, and schedule the next rendezvous time to $I_{\text{allow}}(p_1)$ time later. Since the sender clock counts faster, when the sender wakes up at $T_{\text{SCHD}}(p_1)$ to listen for beacon from the receiver, the receiver won't wake up till $I_{\text{allow}}(p_1)(\frac{1}{a} - 1)$ time later. As a result, an extra delay is introduced to the delivery of packet $p_2$:

$$D(p_2) = I_{\text{allow}}(p_1)\frac{1}{a} + D(p_1) - (T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1)). \tag{19}$$

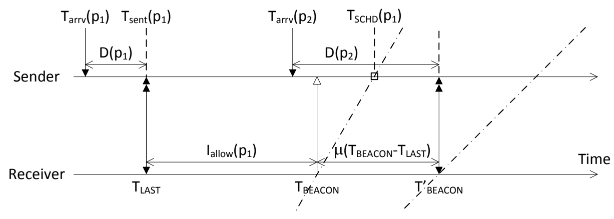
When the system stabilizes, $D(p_1) = D(p_2) \triangleq D$ and $T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1) = \theta(p_1) \triangleq \theta$. Plugging in Eq. (6), we have

$$\begin{aligned} D &= ((1+\mu)\theta - D)\frac{1}{a} + D - \theta \\ \Longrightarrow \quad D &= (\mu + 1 - a)\theta. \end{aligned} \tag{20}$$

This means that an extra delay of $(1 - a)\theta$ has been added to the packet delivery delay.



(a) When the sender clock counts time faster than the receiver clock (i.e., $a < 1$).



(b) When the sender clock counts time slower than the receiver clock (i.e., $a > 1$).

Fig. 6. Effects of time asynchrony on CyMAC performance.

*2) $a > 1$:* In this case, the sender clock counts time at a slower rate than the receiver clock, as shown in Fig. 6(b). After the sender delivers a packet $p_1$ successfully to the receiver, both sender and receiver schedule the next rendezvous time to $I_{\text{allow}}(p_1)$ time later. Since the sender clock counts slower, when the sender wakes up at $T_{\text{SCHD}}(p_1)$ to listen for a beacon from the receiver, the receiver has already finished its beacon transmission. As a result, the sender has to remain awake to wait for the next beacon. We have:

$$D(p_2) = (1+\mu)I_{\text{allow}}(p_1)\frac{1}{a} + D(p_1) - (T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1)). \tag{21}$$

When the system stabilizes, $D(p_1) = D(p_2) \triangleq D$ and $T_{\text{arrv}}(p_2) - T_{\text{arrv}}(p_1) = \theta(p_1) \triangleq \theta$. Plugging in Eq. (6), we have

$$\begin{aligned} D &= (1+\mu)((1+\mu)\theta - D)\frac{1}{a} + D - \theta \\ \Longrightarrow \quad D &= \left(\mu + 1 - \frac{a}{1+\mu}\right)\theta. \end{aligned} \tag{22}$$

This means that an extra delay of $(1 - \frac{a}{1+\mu})\theta$ has been added to the packet delivery delay.

To ameliorate the effects of time asynchrony, we have employed the following schemes in CyMAC:

- To guarantee a relative delay bound of $\mu$, CyMAC does it more conservatively by replacing $\mu$ with $\mu^* = \mu - |1 - \hat{a}|$ as the target delay bound in sensor nodes' operations, where $\hat{a}$ is the upper limit of clock drift between sensor nodes. When $|1-\hat{a}| < \mu$, CyMAC works fine. However, if $\mu \leqslant |1-\hat{a}|$, CyMAC won't be able to provide the desired delay bound. Fortunately, this situation rarely occurs in practice as it makes little sense to ask a sensor network to provide a delay bound that is even tighter than the degree of clock asynchrony between sensor nodes.
- In CyMAC, the sender wakes up a bit earlier prior to the scheduled listen time to wait for beacons. Specifically, if the time between the previous listen time and the next listen time is $\psi$ seconds, the sender will wake up at $\left(\frac{\mu\psi}{2+2\mu}\right)$ prior to the next listen time.

With these two enhancements, we have proved that time asynchrony can be dealt with effectively. Please refer to [16] for proofs which are omitted due to space limitation.

### D. CyMAC Implementation in TinyOS

We implement CyMAC within the UPMA framework [17] in TinyOS, as illustrated in Fig. 7. The generic UPMA framework includes the *Radio Core* as its lower layer to handle packet transmission, reception, backoff control, etc., and the *MacC* component as its upper layer to contain the modules implementing any particular MAC protocol.

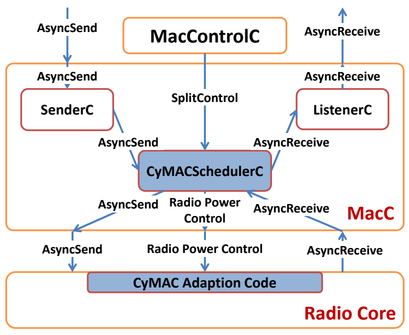


Fig. 7. Implementation of CyMAC within the UPMA framework.

To implement CyMAC, we develop *CyMACSchedulerC* and *CyMAC Adaption Code*, which are parts of the MacC and

Radio Core, respectively. CyMACSchedulerC performs the main functionalities of CyMAC, including receiver beacon generation, sender wakeup, computation of $I_{\text{allow}}$ and radio power control. Similar to the Adaption Code in the implementation of RI-MAC, the CyMAC Adaption Code is mainly for checking CCA and controlling backoff. The difference lies in that, the CyMAC Adaption Code does not support preloading data packets into the CC2420 TX buffer for the following reason. A data packet in CyMAC contains the $I_{\text{allow}}$ value which is computed immediately before the packet is ready to send, and thus the packet cannot be loaded into the buffer earlier. Because of no packet preloading, the processing time for packet transmission becomes longer in CyMAC than in RI-MAC, but the fresh $I_{\text{allow}}$ value carried by the packets helps to reduce both energy consumption and delay in practice. In our implementation, $I_{\text{allow}}$ is calculated according to Eq. (11) with $m = 1$ and has a minimum value of 10ms.

$I_{\text{allow}}$ is added to the radio message header of each data packet and $T_{\text{BEACON}}$ is added to each ACK packet, to enable senders and receivers exchange rendezvous information. Meanwhile, the *hardware ACK* and the *address recognition* function are disabled to allow the CyMAC modules to process ACK packets, just like in RI-MAC.

CyMAC requires a node to use some memory space to maintain state information about its senders and receivers. Particularly, 9 bytes RAM space is needed for each sender and 25 bytes RAM space is needed for each receiver. A sample TinyOS program is developed to use the UPMA framework which contains CyMAC. The program consumes about $24K$ bytes ROM and 800 bytes RAM space on each MICAz mote, where each mote maintains the state information for 10 senders and 5 receivers. Thus, the memory consumption of CyMAC is comparable to RI-MAC and other existing MAC protocols.

## III. Performance Evaluation

Testbed-based experiment and ns-2 based simulation are conducted to evaluate the performance of CyMAC and compare it with RI-MAC, in terms of relative end-to-end delay and duty cycle.

### A. Testbed Evaluation

We set up a testbed system composed of 9 MicaZ motes, forming a line or a star topology as illustrated in Fig. 8. For each topology, CyMAC or RI-MAC is run respectively in the experiment. The average beacon interval in RI-MAC is set to one second. The only parameter for CyMAC is $\mu$, the desired relative delay bound for a single hop. Depending on the desired end-to-end relative delay bound $\mu_{\text{e2e}}$, $\mu$ is set to $(1+\mu_{\text{e2e}})^{1/\xi} - 1$ where $\xi$ is the hop-count diameter of the network, following the definition in Section II-A2.

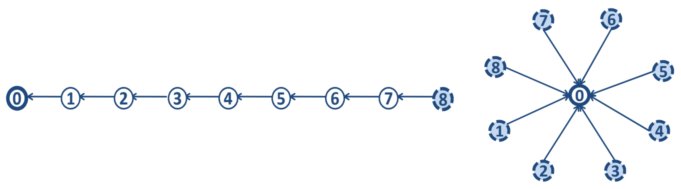


Fig. 8. The line and star topologies of the testbed system.

*1) Line Topology:* In each experiment, there is a single data packet flow starting from node 1, 2, 4 or 8 to sink node 0 with flow length of 1, 2, 4 or 8 hops, respectively. The performances of CyMAC and RI-MAC are compared with varying flow length, data packet generation interval $\tau$ and $\mu_{\text{e2e}}$.

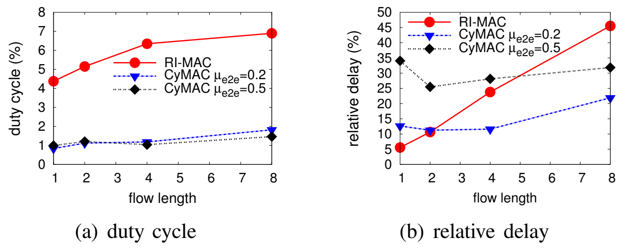


Fig. 9. Comparison of CyMAC and RI-MAC with the line topology as the flow length and the end-to-end relative delay bound $\mu_{\text{e2e}}$ vary. For each flow, data packets are generated at the source node at an average interval of $\tau = 10s$ with 10% variance. $\mu_{\text{e2e}}$ is 0.2 or 0.5.

With varying flow length and $\mu_{e2e}$, the duty cycles of CyMAC and RI-MAC are compared in Fig. 9(a). In RI-MAC, as each node sends a beacon every one second regardless of the traffic condition, and each sender needs to idly listen for 0.5 seconds (on average) to send a packet, a lot of energy is consumed. In contrast, CyMAC establishes rendezvous times between neighbors adaptively to the packet arrival interval; hence, it saves much idle listening and has significantly lower duty cycle than RI-MAC. Fig. 9(b) shows the relative delay with CyMAC and RI-MAC. CyMAC provides the desired delay bound as expected, while the end-to-end delay in RI-MAC increases linearly with the flow length. When the flow length is large, RI-MAC cannot provide the desired delay bound even with a higher duty cycle than CyMAC.

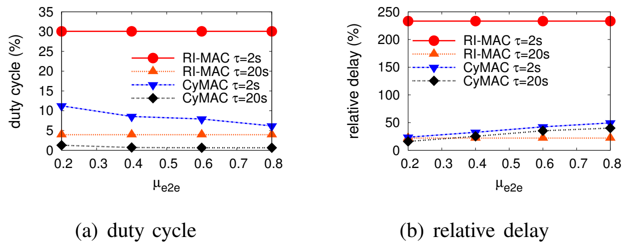


Fig. 10. Comparison of CyMAC and RI-MAC with the line topology as the desired end-to-end relative delay bound $\mu_{\text{e2e}}$ and the average packet generation interval $\tau$ vary. The flow length is fixed at 8 hops.

CyMAC and RI-MAC are compared in Fig. 10 with varying $\mu_{\text{e2e}}$ and $\tau$. As $\mu_{\text{e2e}}$ increases, the relative delay achieved by CyMAC increases accordingly and the average duty cycle of sensor nodes decreases. This is because CyMAC attempts to schedule the rendezvous times between neighbor nodes in the way that the duty cycle of the nodes is as low as possible provided that the desired delay bound is guaranteed. However, RI-MAC does not change its beacon interval as $\mu_{\text{e2e}}$ changes, and therefore keeps the same duty cycle and relative delay. Similar to the reasons explained for Fig. 9, RI-MAC has higher duty cycle and relative delay than CyMAC.

Fig. 11 demonstrates a trace of instantaneous changes in duty cycle and relative delay as $\tau$ varies over time. Each delay

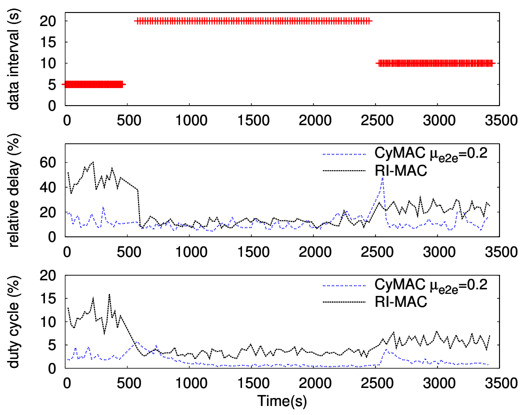


Fig. 11. A trace demonstrates the instantaneous changes in duty cycle and relative delay as the packet generation interval $\tau$ varies over time. The flow length is fixed at 4 hops. $\mu_{e2e}$ is fixed to 0.2.

or duty cycle point in the figure represents the measurement during a 20s period ending at the corresponding time instance. As we can see, CyMAC always guarantees the desired end-to-end delay bound except for a short duration when $\tau$ drops suddenly from 20s to 10s around time 2500s. In this case, some packets (with $\tau = 10$s) are queued and their end-to-end delay may exceed the desired bound. The instantaneous duty cycles in this duration also increase because packets need to be exchanged in a higher frequency in order to reach new rendezvous times. Nevertheless, CyMAC can adapt to the traffic changes and re-stabilize the system quickly. Comparing with CyMAC, RI-MAC does not adapt to the traffic changes and has higher duty cycle and relative delay during most of the time.

*2) Star Topology:* We deploy the testbed network in a star topology, as illustrated in Fig. 8, where node 0 is the sink and other nodes can be data sources. We vary the number of source nodes and the packet generation interval $\tau$ in the experiment.

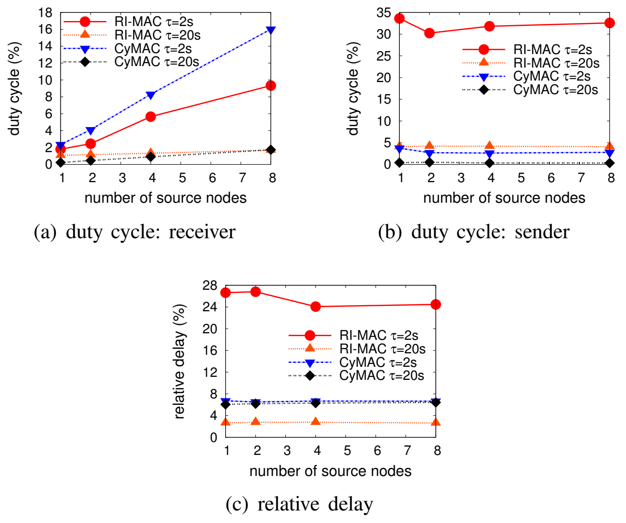


Fig. 12. Comparison of CyMAC and RI-MAC with the star topology as the number of source nodes and the packet generation interval $\tau$ at each source node vary. $\mu_{e2e}$ is 0.1.

Results are shown in Fig. 12. As a receiver in CyMAC sends out beacons at the scheduled beacon times to each of its senders, the time spent on sending beacons increases with the number of senders and with $\tau$. A receiver in RI-MAC, on the other hand, sends out beacons at a constant rate regardless of the number of senders or $\tau$. Also considering that a receiver in CyMAC and RI-MAC spends similar time for packet reception, the overall duty cycle of a receiver in CyMAC has higher duty cycle than its counterpart in RI-MAC when the number of senders is large and/or $\tau$ is small, as illustrated in Fig. 12(a). In this case, however, a sender in CyMAC has a much lower duty cycle than its counterpart in RI-MAC, as illustrated in Fig. 12(b), because CyMAC can significantly reduce the idle listening time for senders through setting up rendezvous times between sender and receiver.

Fig. 12(c) demonstrates that CyMAC always achieves the desired relative delay, regardless of the number of source nodes or $\tau$. RI-MAC limits the average absolute delay to half of the beacon interval, and thus the relative delay increases as $\tau$ decreases. Therefore, the relative delay in RI-MAC is not affected much by the number of source nodes but by $\tau$.

## B. Simulation Evaluation

CyMAC is evaluated in large-scale networks with the ns-2 simulator. Two scenarios are considered: a grid sensor network where one node is the sink and every other node is a data source; a random mesh network with multiple data flows.

*1) Grid Topology:* A total of 49 sensor nodes are deployed to form a 7×7 grid where nearby nodes are 70 meters apart. The node at the center is the sink while every other node is a data source. CyMAC and RI-MAC are run respectively in the network to compare their performances. The packet generation interval $\tau$ at each source node varies from 5 seconds to 80 seconds, and the desired end-to-end relative delay bound $\mu_{e2e}$ is set to 0.2 or 0.4.

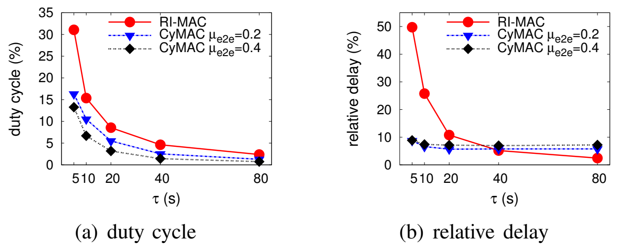


Fig. 13. Comparison of CyMAC and RI-MAC with the grid topology as the packet generation interval $\tau$ at each source node and the desired end-to-end relative delay bound $\mu_{e2e}$ vary.

As showed in Fig. 13, CyMAC always has lower duty cycle than RI-MAC. When the network traffic is heavy (e.g., $\tau = 5$s), a node in CyMAC may spend more time sending beacons to signal its senders than its counterpart in RI-MAC, but it spends much less time on idle listening for each packet that it sends; as the result of these two factors, CyMAC has lower duty cycle than RI-MAC, which is demonstrated by the simulation results. When the network traffic is light (e.g., $\tau = 80$s), CyMAC also has lower duty cycle than RI-MAC because a node in CyMAC has less beacons to send due to the larger $\tau$, but a node in RI-MAC still needs to send beacons at the same rate regardless of the change in traffic condition.

Fig. 13(b) depicts the changes of the end-to-end relative delay as $\tau$ varies. RI-MAC's absolute end-to-end delay is not affected much by $\tau$ because it is mainly determined by the beacon interval and the network hop-count diameter. Hence, as $\tau$ decreases, its relative delay, which is the ratio of the absolute delay to $\tau$, increases accordingly. On the other hand, CyMAC can adapt the rendezvous times between nodes to the change of $\tau$ and maintain a stable relative delay below the desired bound.

*2) Mesh Topology with Multiple Flows:* A total of 49 nodes form a mesh topology with five data flows passing through 25 nodes, as shown in Fig. 14. In this scenario, different flows have different sources, destinations, flow lengths and data generation intervals. They co-exist in the network and affect each other, which represents a more realistic situation than the line, star or grid topology.

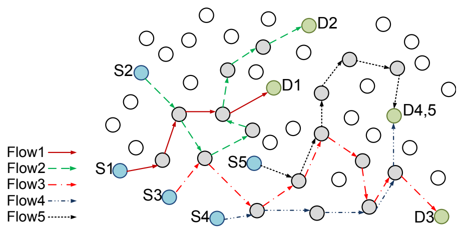
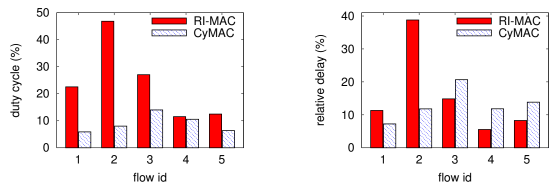


Fig. 14. Mesh topology with multiple flows. A total of 49 nodes are in the network and five flows pass 25 nodes. The numbers of nodes on these flows are 4, 7, 8, 5 and 6, respectively. The data generation intervals of the flows are 20s, 10s, 30s, 50s and 40s, respectively.

Fig. 15 shows that CyMAC has lower duty cycle than RI-MAC for nodes on every flow and all flows can achieve the desired delay bound. As the flow length is different in each flow and the per-hop delay bound is conservatively selected based on the network hop-count diameter, shorter flows achieve lower delay than longer ones. For example, flow 1 has a relative delay of 0.072, flow 3 has a relative delay of 0.207, and their flow lengths are 4 and 8 respectively.



(a) duty cycle (b) relative delay

Fig. 15. Comparison of CyMAC and RI-MAC with the mesh topology. The desired end-to-end relative delay bound is $\mu_{e2e} = 0.2$.

## IV. Conclusions and Future Work

In this work, we propose a new receiver-initiated sensor MAC protocol called CyMAC, and implement it in both TinyOS and the ns-2 simulator. Theoretical analysis and in-depth experiments/simulations demonstrate that CyMAC guarantees the desired delay bound for data delivery services under various traffic conditions. It yields a lower duty cycle than RI-MAC in most cases except when the required delay bound is very tight. In this case, CyMAC can still provide the delay bound guarantee at the cost of having a slightly higher duty cycle than RI-MAC. In addition, CyMAC can tolerate time asynchrony between sensor nodes.

Future work will be in the following directions: (1) CyMAC will be extended to support not only unicast but also multicast data services. (2) The issues and strategies for integrating CyMAC with data aggregation, a fundamental primitive for energy efficiency in sensor networks, will be studied. (3) The current CyMAC design assumes all data flows have the same desired delay bound. In practice, however, different data flows may belong to different applications and thus may have different delay requirements. CyMAC will be extended to guarantee per-flow delay bound in a scalable manner. (4) CyMAC will be enhanced to support highly irregular and dynamic traffic conditions where the packet arrival interval is unstable. One possible approach is as follows. The current online traffic estimation scheme is enhanced so that highly irregular and dynamic traffic conditions could be identified. Once these conditions appear, CyMAC reduces to a RI-MAC type protocol with an appropriately chosen beacon interval, since it could be very expensive to provide any types of delay guarantee under these situations.

## Acknowledgement

This work is supported partly by the NSF under Grant CNS-0831874 and by the ONR under Grant N00014-09-1-0748.

## References

[1] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *IEEE Infocom*, 2002.
[2] L. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *IEEE Infocom*, 2001.
[3] E. Shih, P. Bahl and M.Sinclair, "Wake on wireless: an event driven energy saving strategy for battery operated devices," in *MobiCom*, 2002.
[4] T. Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *SenSys*, 2003.
[5] S. Du, A. Saha and D. Johnson, "Rmac: A routing-enhanced duty-cycle mac protocol for wireless sensor networks," in *IEEE Infocom*, 2007.
[6] Y. Sun, S. Du, O. Gurewitz and D. Johnson, "Dw-mac: a low latency, energy efficient demand-wakeup mac protocol for wireless sensor networks," in *MobiHoc*, 2008.
[7] J. Polastre,J. Hill and D. Culler, "Versatile low power media access for wireless sensor networks," in *SenSys*, 2004.
[8] A. Hoiydi and J. Decotignie, "Wisemac: An ultra low power mac protocol for multi-hop wireless sensor networks," in *LECTURE NOTES IN COMPUTER SCIENCE*, 2004.
[9] M. Buettner, G. Yee , E. Anderson and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *SenSys*, 2006.
[10] Y. Sun, O. Gurewitz and D. Johnson, "Ri-mac: a receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks," in *SenSys*, 2008.
[11] H. Zhang, A. Arora, Y. Choi and M. Gouda, "Reliable bursty convergecast in wireless sensor networks," in *MobiHoc*, 2005.
[12] B. Hull, K. Jamieson and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *SenSys*, 2004.
[13] D. Estrin, R, Govindan, J. Heidemann and S.Kumar, "Next century challenges: scalable coordination in sensor networks," in *MobiCom*, 1999.
[14] W. Ye, F. Silva and J. Heidemann, "Ultra-low duty cycle mac with scheduled channel polling," in *SenSys*, 2006.
[15] R. Krashinsky and H. Balakrishnan, "Minimizing energy for wireless web access with bounded slowdown," in *MobiCom*, 2002.
[16] Y. Peng, Z. Li, D. Qiao, and W. Zhang, "The performance of cymac with time asynchrony," in *Technical Report, http://www.cs.iastate.edu/~ypeng/cymac.pdf*, 2010.
[17] K. Klues, G. Hackmann , O. Chipara and C. Lu, "A component based architecture for power-efficient media access control in wireless sensor networks," in *SenSys*, 2007.