This paper was presented as part of the main technical program at IEEE INFOCOM 2011

PW-MAC: An Energy-Efficient Predictive-Wakeup MAC Protocol for Wireless Sensor Networks

Lei Tang* Yanjun Sun† Omer Gurewitz‡ David B. Johnson*

*Department of Computer Science, Rice University, Houston, TX, USA
†Systems and Applications R&D Center, Texas Instruments, Dallas, TX, USA
‡Department of Communication Systems Engineering, Ben Gurion University, Israel

***Abstract*—This paper presents *PW-MAC (Predictive-Wakeup MAC)*, a new energy-efficient MAC protocol based on *asynchronous duty cycling*. In PW-MAC, nodes each wake up to receive at *randomized, asynchronous* times. PW-MAC minimizes sensor node energy consumption by enabling senders to predict receiver wakeup times; to enable accurate predictions, PW-MAC introduces an *on-demand* prediction error correction mechanism that effectively addresses timing challenges such as unpredictable hardware and operating system delays and clock drift. PW-MAC also introduces an efficient prediction-based retransmission mechanism to achieve high energy efficiency even when wireless collisions occur and packets must be retransmitted. We evaluate PW-MAC on a testbed of MICAz motes and compare it to X-MAC, WiseMAC, and RI-MAC, three previous energy-efficient MAC protocols, under multiple concurrent multihop traffic flows and under hidden-terminal scenarios and scenarios in which nodes have wakeup schedule conflicts. In all experiments, PW-MAC significantly outperformed these other protocols. For example, evaluated on scenarios with 15 concurrent transceivers in the network, the average sender duty cycle for X-MAC, WiseMAC, and RI-MAC were all over 66%, while PW-MAC's average sender duty cycle was only 11%; the delivery latency for PW-MAC in these scenarios was less than 5% that for WiseMAC and X-MAC. In all experiments, PW-MAC maintained a delivery ratio of 100%.**

I. Introduction

The main sources of energy consumption in sensor nodes include listening to the wireless channel and transceiving packets. In recent years, many energy-efficient MAC protocols have been proposed to improve the lifetime of sensor networks by reducing the energy consumed by *idle listening* and *overhearing*. The idle listening problem [13] refers to a node listening to the channel even though there are no radio transmissions to receive. The overhearing problem refers to a node receiving a packet it is not intended to receive.

An important mechanism for reducing energy consumption in sensor networks is *duty cycling*. The *duty cycling* technique saves energy by switching nodes between awake and sleeping states [1]. The average *duty cycle* measures the ratio of the time a node is awake to the total time. Existing duty cycling energy-efficient MAC protocols can be categorized into two types: synchronous and asynchronous.

Synchronous duty-cycling MAC protocols (e.g., S-MAC [13], TRAMA [9], SCP [14], and DW-MAC [10]) reduce sensor energy consumption by synchronizing the sensors' sleep and wakeup times. However, synchronous duty-cycling MAC protocols require multihop time synchronization. In addition, using fixed sleeping times and listening times [13] is inefficient in handling traffic with variable rates.

In contrast, *asynchronous* duty-cycling MAC protocols do not require such synchronization. They may be either sender-initiated (e.g., B-MAC [8], X-MAC [1], and WiseMAC [3]) or receiver-initiated (e.g., RI-MAC [11]). With the *sender-initiated* approach, a sender transmits a preamble before a packet transmission to notify the receiver of the upcoming packet. WiseMAC pioneered predictive wakeup in sensor network MAC protocols by fixing the node wakeup interval, thereby enabling a sender to deduce future receiver wakeup times and send a shortened wakeup preamble shortly before the receiver wakes up. However, this fixed node wakeup interval may allow repeated node wakeup schedule collisions, particularly in dense networks, thus degrading performance. With the *receiver-initiated* approach, in contrast, sender preambles are replaced with receiver wakeup beacons; as the beacon is substantially shorter than a preamble, wireless bandwidth usage and collisions are reduced [11].

In this paper, we present a new *asynchronous duty cycling* energy-efficient MAC protocol called *PW-MAC (Predictive-Wakeup MAC)*. PW-MAC achieves near-optimal energy efficiency both at receivers *and* at senders. In an optimally energy-efficient MAC protocol, when there is a packet to send, the sender and receiver wake up at the same time, transfer the packet reliably, and both then quickly go to sleep again. PW-MAC approaches this optimality in several ways.

Specifically, PW-MAC is a receiver-initiated protocol but introduces use of an independently generated pseudo-random sequence to control each node's wakeup times, allowing senders to accurately predict the time at which a receiver will wake up. Thus, whereas previous receiver-initiated protocols (e.g., [11]) reduce the duty cycle only at receivers, PW-MAC reduces the duty cycle for receivers and for senders. In addition, to prevent senders from missing the wakeup of receivers due to factors such as hardware and operating system latency and clock drift, PW-MAC introduces a novel on-demand prediction-error correction mechanism. Thus, unlike prior protocols using forms of wakeup prediction (e.g., [3]), PW-MAC is able to maintain accurate prediction; we have found in experiments on real sensor nodes that wakeup prediction without such correction can otherwise lead to significantly reduced performance.

Furthermore, the traffic in wireless sensor networks can be bursty, and multiple sensors may attempt to transmit at the

978-1-4244-9921-2/11/$26.00 ©2011 IEEE

same time, possibly to the same node, e.g., when a group of sensors detect the occurrence of an event and transmit a report to the sink. Wireless collisions are likely when there are multiple concurrent transceivers, and prior work did not specifically include a mechanism for energy-efficiently resolving collisions and retransmitting lost packets. PW-MAC provides a prediction-based retransmission mechanism to achieve high energy efficiency even when wireless collisions occur and packets need to be retransmitted.

Finally, PW-MAC scales well in large and dense networks owing to its small memory and message overhead. In PW-MAC, a node does not explicitly send its wakeup times to other nodes. Instead, a sender independently deduces future wakeup times of a receiver based on the sender's knowledge of the receiver's pseudo-random wakeup-schedule generator.

The contributions of this paper include the following:

- We present the design and implementation of PW-MAC (Predictive-Wakeup MAC). PW-MAC is designed to minimize sensor node energy consumption by enabling senders to predict receiver wakeup times, even given the challenges of unpredictable hardware and OS delay and clock drift. PW-MAC achieves very high energy efficiency by minimizing idle listening and overhearing.
- We present an efficient prediction-based retransmission mechanism to achieve high energy efficiency when wireless collisions occur and packets need to be retransmitted.
- Through experiments on MICAz motes, we show that the prediction error caused by hardware and OS latency can be much larger than that caused by clock drift. To enable a sender to wake up shortly before a receiver does, we introduce an on-demand prediction error correction mechanism, allowing a sender to resynchronize with a receiver when needed. Our experimental results on MICAz motes show that PW-MAC is very effective in controlling the prediction error.
- We present the results of experiments on a testbed of MICAz motes to evaluate the performance of PW-MAC compared with other energy-efficient MAC protocols (i.e., X-MAC, WiseMAC, and RI-MAC) under single-hop and multihop traffic flows, under hidden-terminal scenarios, and under scenarios in which nodes have wakeup schedule conflicts. In all experiments, PW-MAC significantly outperformed these other protocols.

The rest of this paper is organized as follows. Section II describes related work on duty-cycling MAC protocols for sensor networks. Section III presents the design of the PW-MAC protocol, and Section IV presents the evaluations of PW-MAC on a testbed of MICAz motes. Finally, Section V presents conclusions.

II. Related Work

Several asynchronous duty-cycling MAC protocols have recently been proposed in the literature [1], [3], [8], [11]. Such protocols are attractive, as they do not require multihop time synchronization among nodes as is required by synchronous duty-cycling MAC protocols [5], [10], [12], [13].

B-MAC [8] and X-MAC [1] are early examples of *sender-initiated* asynchronous duty-cycling protocols. In these protocols, before transmitting a DATA frame, a sender transmits a *preamble* of duration longer than the receiver's periodic wakeup interval, serving as a notification of the pending DATA frame transmission. In order to receive packets, each node periodically wakes up to check for activity on the wireless channel, and if activity is present, the node remains active to receive a possible incoming packet that may be destined to it. Each node will thus wake up to receive at least once during the preamble and will then be able to receive the DATA frame that follows the preamble. The UPMA package [6] for TinyOS implemented a modified version of X-MAC by using copies of the DATA frame itself as the short preambles. This variation of X-MAC simplified the implementation in TinyOS and helps a sender to determine whether the DATA was successfully delivered, as the receiver returns an ACK following receipt of the DATA.

However, with all of these sender-initiated protocols, a sender often shows much larger duty cycle than a receiver, transmitting the preamble until the receiver wakes up and thus capturing the wireless channel throughout this time. PW-MAC, in contrast, captures the channel only for very short intervals and achieves a very low duty cycle at receivers *and* at senders.

WiseMAC [3] is a sender-initiated protocol, similar to B-MAC. WiseMAC, however, pioneered predictive wakeup in sensor network MAC protocols to enable reducing the preamble length by fixing the node wakeup interval. Thus, a sender can simply predict the next wakeup time of a receiver based on this repeating schedule and send a shortened wakeup preamble beginning shortly before the predicted receiver wakeup time. However, as with B-MAC, when there are multiple concurrent transceivers in the network, the simultaneous preamble transmission from each of the multiple senders creates poor performance. In addition, each individual node with WiseMAC must use the same fixed, repeating wakeup interval over time, which can lead to persistent collisions if two nodes choose approximately the same wakeup time.

Unlike WiseMAC, nodes with PW-MAC wake up according to independently generated pseudo-random schedules, ensuring that nodes will not continuously generate the same wakeup times and thereby significantly reducing transmission collisions. PW-MAC also allows a node to change the parameters of its pseudo-random wakeup-schedule generator (e.g., if the node reboots or due to other circumstances). Through the on-demand *prediction state* request mechanism of PW-MAC, a sender can quickly learn to predict the changed wakeup times of a receiver.

Furthermore, prediction in WiseMAC uses only clock drift rate to adjust the predicted receiver wakeup times, ignoring the prediction error that may be caused by factors such as variable hardware and OS latency; the effect of these additional factors, as shown in our experiments, can lead to poor performance on real sensor nodes. WiseMAC also does not address the issue of energy-efficient collision resolution and packet retransmission, and its performance thus degrades significantly as the number

of concurrent traffic flows increases. In contrast, PW-MAC can effectively control the prediction error caused by hardware and OS latency as well as by clock drift, and achieves high energy efficiency on real sensor nodes. PW-MAC furthermore includes a prediction-based retransmission mechanism to achieve high energy efficiency even when wireless collisions have caused packet transmission failures.

Crankshaft [5] is a sender-initiated protocol that divides time into frames, with each frame divided into n slots. Crankshaft assumes global synchronization and assigns a wakeup slot to each receiver as its MAC address modulo n. Every receiver can use only a fixed slot in each frame, making Crankshaft inflexible in handling bursty traffic that may occur in sensor networks. In addition, using this fixed scheduling of receiver wakeup slots can lead to receivers repeatedly waking up at the same time, causing packet collisions.

O-MAC [2] is also a sender-initiated protocol that divides time into frames and frames into slots. However, although the authors of O-MAC [2] presented a careful analytical study on which they based the design of the O-MAC protocol, many important details of a complete protocol are unstated in their paper. Similar to PW-MAC, wakeup scheduling for each receiver in O-MAC is based on a pseudo-random sequence. However, in contrast to PW-MAC, nodes in O-MAC cannot learn their neighbor's wakeup schedules just by listening to the channel; they must run a special neighborhood discovery procedure (not specified in the paper) in order to learn these schedules and to synchronize time with each receiver. Furthermore, perfect synchronization between nodes is difficult to achieve and maintain. If the clocks drift between sender and receiver, packet transmissions in O-MAC may be lost since the sender may transmit the packet (or part of the packet) before the receiver wakes up or after it goes back to sleep; O-MAC also does not define a retransmission mechanism.

In contrast, in PW-MAC, we present an efficient prediction-based retransmission mechanism to achieve high energy efficiency when packets must be retransmitted. In addition, the use of slotted time in O-MAC limits the choices for the pseudo-random number that determines the slot within the receiver's frame in which the receiver will wake up next. This limitation increases the likelihood of collisions, where two receivers pick the same slot, which may particularly be a problem in dense networks. In contrast, time in PW-MAC is unslotted, allowing more variation in pseudo-random choices, and PW-MAC uses carrier sensing to avoid collisions between two nodes that pick closely separated wakeup times. Moreover, since each receiver in O-MAC can use only one slot within each frame, O-MAC cannot efficiently handle bursty traffic, whereas receivers in PW-MAC may remain awake if additional packets are transmitted to it after receiving one. Finally, although some analytical evaluation of O-MAC has been presented [2], no detailed simulation or experimental evaluation of the complete O-MAC protocol exists; in particular, the expected performance of O-MAC on real sensor node hardware in different scenarios remains unclear.

RI-MAC [11], in our prior work, introduced a *receiver-initiated* approach, in which the receiver-initiated wakeup beacons are used to avoid long sender-initiated preambles required by protocols based on sender-initiated transmissions. This approach increases channel utilization and enables more efficient collision detection. In RI-MAC, each node announces its wakeup with a *beacon*, and a sender starts the DATA transmission upon receiving a beacon from its intended receiver. However, when a sender in RI-MAC has a packet to send, it immediately wakes up to wait for the receiver, leading to a large sender duty cycle due to its idle listening until the receiver wakes up. Compared with RI-MAC, PW-MAC achieves near-optimal duty cycle at receivers and at senders.

III. PW-MAC Design

In this section, we present the detailed design of the PW-MAC (Predictive-Wakeup MAC) protocol for sensor networks. We first describe the basic predictive wakeup mechanism of PW-MAC in Section III-A. Then, Section III-B presents the prediction-based retransmission mechanism of PW-MAC, and Section III-C presents an on-demand prediction error correction mechanism that efficiently controls the prediction error caused by factors such as hardware and operating system latency and clock drift. Finally, Section III-D shows the algorithm used by a sender to compute its wakeup time when sending a DATA packet to some receiver.

A. PW-MAC Predictive-Wakeup Mechanism

The goal of PW-MAC is for a sender to wake up and turn on its radio right before the intended receiver wakes up. In addition, PW-MAC strives to avoid and efficiently resolve any radio collisions caused by multiple concurrent traffic flows.

To enable a sender to accurately predict the wakeup times of a receiver, we require every node in PW-MAC to compute its wakeup times using its *pseudo-random* wakeup-schedule generator rather than waking up on a truly random schedule. By using a pseudo-random wakeup schedule rather than a fixed schedule (which would also be predictable, as is done in WiseMAC [3]), we avoid the possibility of neighboring nodes consistently waking up at the same time, as such occurrences would significantly increase the chances of collisions from senders that are hidden with respect to each other; such collisions would also generally be persistent with a fixed wakeup schedule.

The predictive-wakeup mechanism of PW-MAC can be applied to receiver-initiated duty-cycling MAC protocols (e.g., RI-MAC) and to sender-initiated duty-cycling MAC protocols (e.g., WiseMAC and SCP). Our design for PW-MAC uses the receiver-initiated approach since it provides good receiver duty cycle performance and provides a beacon transmission mechanism that can be used as the basis for transmitting the prediction state of a node when needed.

There are many ways to build pseudo-random number generators suitable for use with PW-MAC. For the sake of simplicity, we take the linear congruential generator (LCG) [7] in Equation 1 as an example:

$$X_{n+1} = (aX_n + c) \bmod m \tag{1}$$

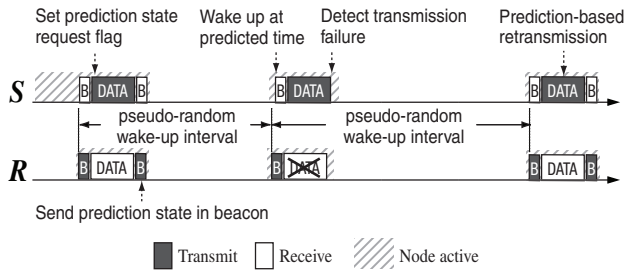


Fig. 1. A sender S in PW-MAC requests the prediction state of a receiver R and wakes up right before R does, after learning the prediction state of R. The prediction-based retransmission mechanism of PW-MAC enables S to detect the transmission failure and efficiently do packet retransmissions.

Here, $m > 0$ is the *modulus*, a $(0 < a < m)$ is the *multiplier*, c $(0 \leq c < m)$ is the *increment*, and X_n $(0 \leq X_n < m)$ is the current *seed*. Each X_{n+1} generated can be used as a pseudo-random number and becomes the new seed.

Different nodes have different parameters for their pseudo-random number generators to avoid nodes persistently generating the same numbers. If the m, a, c and X_n of the pseudo-random number generator of a node R are learned by another node S, S can deduce the values of all future pseudo-random numbers generated by R. When R uses these pseudo-random values as its wakeup intervals and S has learned the time difference between S and R's clocks, S can deduce all future wakeup times of R and wake up right before R does any time S needs to send a packet to R, significantly reducing sender duty cycle.

The *prediction state* of R learned by S comprises the parameters and current seed of the pseudo-random number generator of R (6 bytes in total), as well as the current time difference between S and R (4 bytes); a sender thus needs only 10 bytes of memory to store the prediction state of a receiver. Owing to its small memory and message overhead and its ability to maintain high energy efficiency under multiple concurrent traffic flows, PW-MAC scales well in large and dense networks. The message overhead of PW-MAC is analyzed in Section III-C.

Figure 1 illustrates the predictive-wakeup mechanism of PW-MAC. As in RI-MAC [11], each node periodically wakes up and broadcasts a beacon, denoted as "B" in the figure, to announce that it is awake and ready to receive DATA packets. In PW-MAC, as shown for node R in this figure, the interval between wakeups is calculated by a pseudo-random number generator such as the one in Equation 1.

A node S in PW-MAC learns or updates the prediction state of a neighbor R only *on-demand*, when necessary. If, when S has a packet (e.g., from the network layer) to send to R, S has does not have the prediction state of R, S turns on its radio and waits for a beacon from R. After receiving R's beacon, when S transmits the DATA packet, S then sets a special flag in the DATA packet header to request R's prediction state. Once R receives this DATA packet, R sends another beacon that serves both to acknowledge the DATA packet reception (i.e., an ACK beacon) and to allow additional DATA packets to be sent to R; in response to the prediction state request from S, R also embeds its current time and prediction state in the beacon. The current time of R is used by S to compute the time difference between S and R's clocks. In order to precisely determine this time difference, the current time and prediction state of R are added to the ACK beacon immediately before the packet is sent by R. As soon as the radio hardware of S receives the ACK from R, S computes the time difference between S and R to minimize the error of time difference caused by the sensor node's operating system delay.

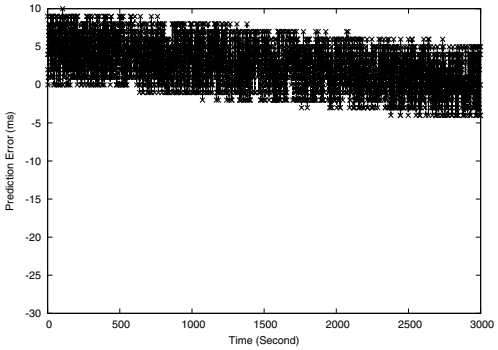
With the prediction information received from the ACK beacon, node S can predict future wakeup times of R. In the future, if S has another DATA packet for R, S wakes up shortly before the predicted wakeup time of R, as illustrated in Figure 1. In contrast to RI-MAC, in which a sender stays awake for on average half a wakeup interval, waiting for the receiver before starting DATA transmission, PW-MAC reduces this idle listening time to almost 0 once the prediction state of the intended receiver has been learned by the sender. In this way, PW-MAC greatly improves energy efficiency.

B. Prediction-Based Retransmission

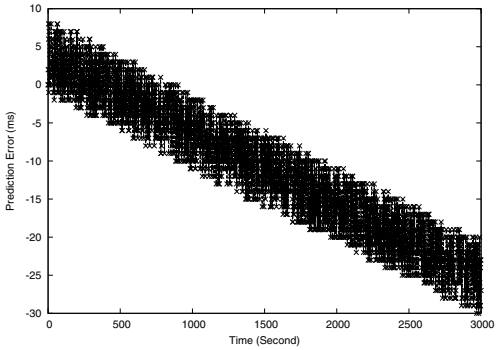
Energy-efficiently doing packet retransmissions is an important issue in a real sensor network. The traffic in wireless sensor networks can be bursty, and multiple nodes may transmit at the same time, causing wireless collisions. Wireless channel interference may also lead to the need for packet retransmissions. PW-MAC includes a prediction-based retransmission mechanism that achieves high energy efficiency even when such factors have caused packet transmission failures and packets must be retransmitted.

With existing energy-efficient asynchronous duty-cycling MAC protocols, the senders after a collision will do a short backoff and retransmit the DATA packets. This way of conducting retransmissions may significantly increase sender duty cycle since, after the collision, receivers may have gone back to sleeping state since they have not received valid packets or they regard the packet transmission as completed, thereby making these retransmissions futile. For instance, after packet transmission failures, senders in RI-MAC stay awake until receivers wake up again, leading to large sender duty cycles. Disregarding the state of the receivers, senders in WiseMAC repeatedly retransmit the packets until the packets are acknowledged, potentially causing further wireless collisions and large sender duty cycle.

In contrast, senders in PW-MAC achieve high energy efficiency even when packets have to be retransmitted, by detecting wireless collisions, switching to sleeping state and intelligently choosing when to wake up and retransmit the packets. Figure 1 illustrates the prediction-based retransmission mechanism of PW-MAC. If a sender S receives the wakeup beacon from receiver R but not an ACK beacon for the DATA packet sent, S recognizes that either the DATA packet or the ACK beacon transmission failed. S then switches to sleeping state and wakes up at the next predicted receiver wakeup time to retransmit the DATA packet, thereby minimizing the energy spent on waiting for the receiver.



(a) Factors such as variable operating system and hardware latency dominate the prediction error, as clock drift between these two motes is very small.



(b) Clock drift is much more significant between these two motes, dominating other factors such as variable operating system and hardware latency.

Fig. 2. The prediction error between two different, arbitrarily chosen pairs of MICAz motes over time.

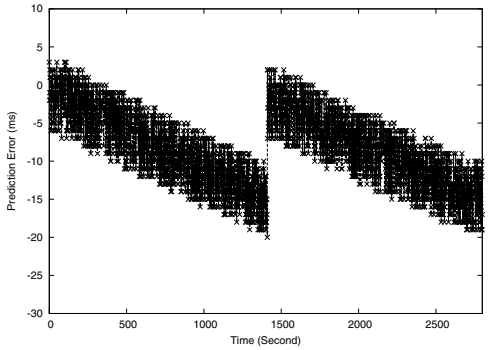


Fig. 3. The development of prediction error with on-demand correction. The correction threshold is configured as 20 ms, which is the same as the sender wakeup advance time. The same pair of MICAz motes as used in Figure 2(b) were used in this experiment.

If two senders happen to send DATA packets at the same time to a receiver, such sender transmission collision is resolved using the receiver-based collision resolution mechanism of RI-MAC [11], with which the receiver detects the collision through the Clear Channel Assessment (CCA) and notifies the senders to retransmit their packets after increasing their random backoff windows.

C. On-Demand Prediction Error Correction

We define the *prediction error* for a wakeup of a node R as the difference between the actual wakeup time and the predicted wakeup time of R. Controlling prediction error is an important issue because if a receiver wakes up before the predicted wakeup time, the sender will miss the wakeup of the receiver, prolonging the packet delivery latency at least by a full wakeup interval and significantly increasing the sender's duty cycle. On the other hand, if a receiver wakes up much later than the predicted wakeup time, the sender duty cycle will also increase, since the sender then has to remain awake until the receiver does wake up.

WiseMAC partially handles the prediction error by adjusting the sender wakeup time using a clock drift rate; however, this is not effective on real sensor nodes, as our experiments on MICAz motes indicate that the prediction error not only results from clock drift but also from other factors such as hardware and operating system latency. Section IV-A shows that WiseMAC suffers reduced performance due to not addressing the prediction error caused by these factors. Furthermore, the clock drift rate between nodes is subject to the influences such as temperature and humidity [4].

PW-MAC introduces an efficient on-demand prediction error correction mechanism that can effectively control the prediction error caused by various factors such as those discussed above. Figure 2 shows how the prediction error between two different, arbitrarily chosen pairs of MICAz motes develops over time. The nodes in these experiments use the TinyOS 2.1.0 operating system, but different motes are used in each figure. In each of the experiments of Figures 2(a) and 2(b), a node R sends its prediction state to a node S only at the beginning of the experiment. The average node wakeup interval is 1 second. Node S measures the difference between the predicted wakeup time and the actual wakeup time of R. Each figure shows 3000 measurement points, each indicating the prediction error at a particular time.

These results demonstrate that clock drift may not always be the dominating factor causing prediction error. Only using clock drift rate in the computation of the predicted wakeup time can cause senders to wake up too early or too late, depending on the parameter chosen for clock drift as part of the prediction. For instance, at the beginning of Figure 2(a), prediction error varies from 0 to 10 ms, whereas the prediction error caused by clock drift is almost 0 since R just sent its prediction state to S. Over time, the prediction error between these two motes grows very slowly, as their clocks appear to run at almost the same rate. For this pair of motes, the prediction error is mainly caused by hardware and operating system latency rather than by clock drift. In contrast, in Figure 2(b), the rate of clock drift between these two motes is much greater, even though the two motes used in this experiment are the same type of MICAz motes used in the experiment in Figure 2(a). Again, at the beginning of this experiment, the error caused by clock drift is almost 0, since as in Figure 2(a), R here just sent its prediction state to S. Over time, however, the prediction error quickly becomes dominated by the clock drift between this pair of motes.

PW-MAC utilizes a platform-dependent parameter, *sender*

wakeup advance time, to account for the hardware and operating system latency. A sender also adjusts its wakeup advance time based on the clock drift rate. With the on-demand prediction correction mechanism of PW-MAC, a node S requests an update of the prediction state of another node R when it detects that the prediction error is larger than the sender wakeup advance time. Figure 3 illustrates the effectiveness of the on-demand prediction error correction mechanism of PW-MAC on a pair of MICAz motes; the same pair of MICAz motes as used in the experiment of Figure 2(b) were used, but in this case, the on-demand prediction error correction was enabled. Once a node detects that the prediction error of the other node is more than the correction threshold, it quickly recovers the ability to precisely predict the other node wakeup times. Through this on-demand prediction error correction mechanism, a sender can effectively control the prediction error of a receiver to be within its wakeup advance time, thereby minimizing the cases in which a sender misses the wakeup of a receiver. On-demand prediction error correction also is very efficient in terms of message overhead. From Figure 3, a node only updates another node once every 1400 seconds and the size of the prediction state update is only 10 bytes.

D. Sender Wakeup Time Algorithm

The algorithm used by a sender S for computing its wakeup time for sending a DATA packet to an intended receiver R is shown in Figure 4. The value A here denotes the *sender wakeup advance time* parameter, and A_{min} denotes the minimum time required to power up a node ($A \geq A_{min}$). For MICAz motes with TinyOS 2.1.0, our experiments show that A_{min} is about 2 ms. The value *Drift* denotes the clock drift ratio, measured in our experiments for MICAz motes to be limited to about 40 ms per hour.

Through a previous prediction state request, S has obtained a random number generator seed of R (i.e., *randState*$[R]$, 2 bytes), the time difference between S and R (i.e., *timeDifference*$[R]$, 4 bytes), and the last wakeup time of R (i.e., *nextWakeupTime*$[R]$, 4 bytes). The value *timeDifference*$[R]$ is computed by S as the difference between its local time when receiving the prediction state from R and the current time of R embedded in the prediction state update from R. The value *lastUpdate*$[R]$ denotes the time that S received the prediction state from R, which is used to compute T_d, the prediction error caused by clock drift. In Figure 4, S computes each successive wakeup time of R until it finds one that is at least $T_d + A_{min}$ larger than the current time of R.

procedure COMPUTE-SENDER-WAKEUP-TIME(R):

```
if the time or random number generator info of R is unknown then
    return 0; {wake up now to wait R}
end if
{compute the current time of R}
curTime[R] ← localTime − timeDifference[R];
{compute the error caused by clock drift}
T_d ← Drift × (curTime[R] − lastUpdate[R]);
{if next wakeup time of S for sending to R has been computed before}
if nextWakeupTime[R] > curTime[R] then
    return (nextWakeupTime[R] − curTime[R]);
end if
while nextWakeupTime[R] ≤ (curTime[R] + T_d + A_min) do
    randState[R] = RandNum(randState[R], ID_R);
    nextWakeupTime[R] += randState[R];
end while
if nextWakeupTime[R] > (curTime[R] + A + T_d) then
    return (nextWakeupTime[R] − curTime[R] − A − T_d);
else
    return 0; {wake up now}
end if
```

Fig. 4. A sender S computes when it wakes up to send DATA packets to a receiver R.

IV. Evaluation on MICAz Motes

In this section, we evaluate PW-MAC by comparing its performance with that of WiseMAC, RI-MAC, and X-MAC in a testbed of MICAz motes.

We implemented PW-MAC in TinyOS 2.1.0. Each wakeup interval of a node is computed as a pseudo-random number between 500 to 1500 ms. The parameters a, c, and m of a node's pseudo-random number generator were configured as *node ID* $\times$ 20, 7, and 1000, respectively, such that the pseudo-random numbers generated have a full period [7]. We used the implementation of X-MAC under the UPMA framework [6]. WiseMAC has previously been evaluated only through simulations [3], without an implementation on real sensor node hardware; therefore, we implemented WiseMAC under the UPMA framework in TinyOS.

TABLE I
PW-MAC Performance with Different Sender Wakeup Advance Time Configurations

Wakeup Advance	Sender Duty Cycle	Receiver Duty Cycle	Delivery Latency	PDR
30 ms	6.6%	3.7%	519 ms	100%
20 ms	6.0%	3.7%	517 ms	100%
15 ms	6.8%	3.7%	511 ms	100%
10 ms	9.8%	3.7%	521 ms	100%

The following metrics are measured in our evaluation:

- *Data packet delivery ratio (PDR)*: the percentage of DATA packets that are successfully delivered from the source to the destination.
- *Average duty cycle*: the ratio of the time a node is awake to the total experiment time. The lower the duty cycle, the less the energy consumption.
- *Data packet delivery latency*: the average time taken by each delivered DATA packet to be delivered from the source to the destination.

A. Configuration of Sender Wakeup Advance Time

A sender must wake up slightly before the predicted wakeup time of the receiver. Since the sender wakeup advance time (i.e., the parameter A) is hardware-dependent, this section studies the configuration of wakeup advance time through experiments on real hardware.

There are 3 pairs of MICAz motes in this experiment. Each pair of motes comprises a sender and a receiver. The packet generation interval at each sender is randomly distributed between 0.5 to 1.5 seconds. We tested each wakeup advance time configuration 3 times and present the average results in Table I.

From these experiments, we concluded that overall performance for PW-MAC on this hardware is best when the sender

TABLE II
PERFORMANCE OF WISEMAC WITH AND WITHOUT USING THE SENDER WAKEUP ADVANCE TIME

WiseMAC with Wakeup Advance		WiseMAC without Wakeup Advance	
Sender Duty Cycle	Delivery Latency	Sender Duty Cycle	Delivery Latency
9.4 %	538 ms	39.7%	930 ms

wakeup advance time is 20 ms, which is consistent with the experimental results for prediction error shown in Figure 2. As shown in Figure 2, the prediction error of a mote caused by hardware and operating system latency was about 10 ms when one mote stayed awake to measure the wakeup times of the other mote. When both motes are duty cycling, the prediction error can be doubled to 20 ms. Configuring the sender wakeup advance time as 20 ms offsets the prediction error caused by hardware and operating system latency and reduces the occurrences of a sender missing a receiver.

When the sender wakeup advance time was configured as 10 ms, the sender duty cycle was higher than when 20 ms was used, since the prediction error caused by hardware and operating system latency can be larger than 10 ms, leading to senders missing the wakeups of the receivers. Based on these results, we set the sender wakeup advance time to 20 ms in the rest of our experiments.

WiseMAC considers only the prediction error caused by clock drift. Table II compares the performance of WiseMAC with and without using the sender wakeup advance time parameter, configured as 20 ms. Adding the sender wakeup advance time significantly improves the performance of WiseMAC. WiseMAC without this parameter has a high sender duty cycle and delivery latency, as the prediction error caused by hardware and operating system latency leads to the sender missing some wakeups of the receiver. To improve the performance of WiseMAC, we include the sender wakeup advance time parameter (20 ms) when conducting the experiments in the following sections.

B. Experimental Results with Conflicting Wakeup Schedules

Two nodes may happen to choose the same wakeup time, which can lead to the collisions of the packets transmitted to them. This section evaluates the performance of PW-MAC and WiseMAC when there are such wakeup schedule conflicts.

In the experiments, there are four motes ($S1$, $S2$, $R1$, and $R2$), with $S1$ being the sender to $R1$ and $S2$ being the sender to $R2$. All four motes were adjacent to each other. To create wakeup schedule conflicts, we made $R1$ and $R2$ use the same first wakeup time. Each experiment lasted 300 seconds and was conducted three times. Table III reports the average sender duty cycle and delivery latency of PW-MAC and WiseMAC in these experiments. PW-MAC achieved a low sender duty cycle, whereas WiseMAC had a much higher sender duty cycle. The reason for the high sender duty cycle of WiseMAC is that once two receivers wake up at the same time, they will continue waking up at the same time in the following cycles due to the fixed wakeup interval of WiseMAC. The two senders will transmit packets to the two receivers at the same time, causing a large number of collisions. When packets collide, WiseMAC does not have an efficient retransmission mechanism other than keeping senders awake to retransmit the packets.

TABLE III
AVERAGE SENDER DUTY CYCLE AND PACKET DELIVERY LATENCY OF PW-MAC AND WISEMAC WHEN THERE ARE WAKEUP SCHEDULE CONFLICTS.

	Sender Duty Cycle	Delivery Latency
PW-MAC	5.5%	579 ms
WiseMAC	78.1%	17425 ms

TABLE IV
AVERAGE SENDER DUTY CYCLE AND PACKET DELIVERY LATENCY OF PW-MAC AND WISEMAC IN HIDDEN-TERMINAL EXPERIMENTS

	Sender Duty Cycle	Delivery Latency
PW-MAC	6.8%	670 ms
WiseMAC	81.6%	10905 ms

In contrast, nodes in PW-MAC follow independent, pseudo-random wakeup schedules, ensuring that nodes waking up at the same time will have different wakeup times in following cycles, thereby greatly reducing the packet collision probability. In addition, when collisions occur, senders in PW-MAC are still able to maintain a small duty cycle through PW-MAC's prediction-based retransmission mechanism.

Due to significant packet collisions in WiseMAC, a packet has to be retransmitted multiple times to reach the receiver, resulting in the large delivery latency of WiseMAC. In these experiments, PW-MAC achieved 100% packet delivery ratio, whereas the packet delivery ratio of WiseMAC ranged from 99% to 100%. The receiver duty cycle of PW-MAC and WiseMAC were both smaller than 5%.

C. Experimental Results with Hidden Terminals

When senders are hidden to each other, the CSMA mechanism becomes ineffective and the transmissions from the hidden senders may collide at the receiver. To evaluate how efficiently PW-MAC and WiseMAC handle packet collisions and conduct retransmissions in hidden-terminal situations, we conducted experiments in which there were two senders hidden to each other, with one receiver placed between the two senders. Each experiment was conducted three times.

Table IV shows the experimental results. The average sender duty cycle of WiseMAC in these hidden-terminal experiments was very high, at 81.6%. WiseMAC also had a much larger delivery latency than that of PW-MAC. WiseMAC is unable to detect the packet collisions of the two hidden senders and does not have an efficient mechanism to handle packet retransmissions other than keeping senders awake to repeatedly retransmit the packets until receiving the ACK packets from the receiver; this causes persistent collisions and a high sender duty cycle and leads to packets staying in the queue for a long time before they are delivered to the receiver.

In contrast, PW-MAC achieved a much smaller sender duty cycle (6.8%) and delivery latency with hidden terminals because the senders in PW-MAC can detect collisions and efficiently retransmit the packets using PW-MAC's prediction-based retransmission mechanism. The packet delivery ratio of the two protocols was 100%, and the receiver duty cycle for

both was less than 8%.

D. Experimental Results in Multihop Networks

This section presents the evaluation of PW-MAC, WiseMAC, RI-MAC, and X-MAC in multihop networks. These experiments were conducted on an indoor testbed consisting of 15 MICAz motes in a 3×5 grid topology. There are between 1 and 3 concurrent multihop traffic flows, with hop count ranging from 1 to 4. Each traffic flow traverses a distinct multihop path. The packet sending interval of each traffic source is randomly distributed from 0.5 to 1.5 seconds. The packet size is 28 bytes. To evaluate the influences of wireless collisions on packet transmissions, motes were placed within the radio interference range of each other.

The left part of each graph in Figure 5 shows the performance of the four protocols when there is one multihop traffic flow, with lengths ranging from 1 to 4 hops. The right part of each graph in Figure 5 shows the performance of these protocols when there are one to three concurrent 4-hop traffic flows. A multihop traffic flow traverses multiple intermediate forwarders to reach the destination. The destination nodes only receive packets and do not generate or forward packets. The *senders* (i.e., sources and intermediate forwarders) may receive and send packets. Figure 5 shows the *destination duty cycle* and *sender duty cycle* in separate graphs to more clearly present the duty cycle performance of the motes.

In addition to its high sender duty cycle, the packet delivery ratio and delivery latency of X-MAC degrade as the hop-length of a traffic flow and the number of multihop traffic flows increase. With X-MAC, senders repeatedly transmit each DATA packet until it is acknowledged by the receiver, which, in a multihop network, can cause significant collisions. Once a collision occurs, senders do a short random backoff and retransmit the packets. The larger the number of senders, the more likely the retransmissions from multiple senders collide. These significant wireless collisions lead to the high sender duty cycle and the large delivery latency of X-MAC and cause the drop of its packet delivery ratio.

All four MAC protocols have a small destination duty cycle because, unlike the sources and intermediate packet forwarders, the destinations only periodically wake up to receive the packets without transmitting any packets. In addition, to reduce overhearing, a destination hearing garbled packets and packets destined to other nodes will go back to sleeping state.

When there was only one multihop traffic flow, WiseMAC achieved a smaller sender duty cycle than X-MAC and RI-MAC. However, when the number of concurrent multihop traffic flows increased, the performance of WiseMAC degraded significantly: the average sender duty cycle and packet delivery latency of WiseMAC when there was only one 4-hop traffic flow were 21% and 4.7 seconds, respectively, whereas these two metrics of WiseMAC deteriorated to 72% and 90 seconds, when there were three 4-hop traffic flows (each 4-hop traffic flow consisted of 4 senders and one destination). This performance degradation is due to the following reasons. First, the wakeup interval of a node in WiseMAC is fixed. Thus, as the number of senders increases from 4 to 12, so does the probability of having node wakeup schedule collisions, which, as shown in Section IV-B, causes persistent collisions and dramatic performance degradation in WiseMAC. Furthermore, WiseMAC uses preambles to send DATA packets. Once packet collisions occur, with WiseMAC, a sender does not have an efficient packet retransmission mechanism to resolve collisions other than repeatedly retransmitting the packets, which interferes with the normal transmissions of other senders and may further lead to additional collisions. With this retransmission mechanism, when the number of senders increased to 12, packet collisions increased significantly and senders had to stay wakeup for a long time to retransmit packets.

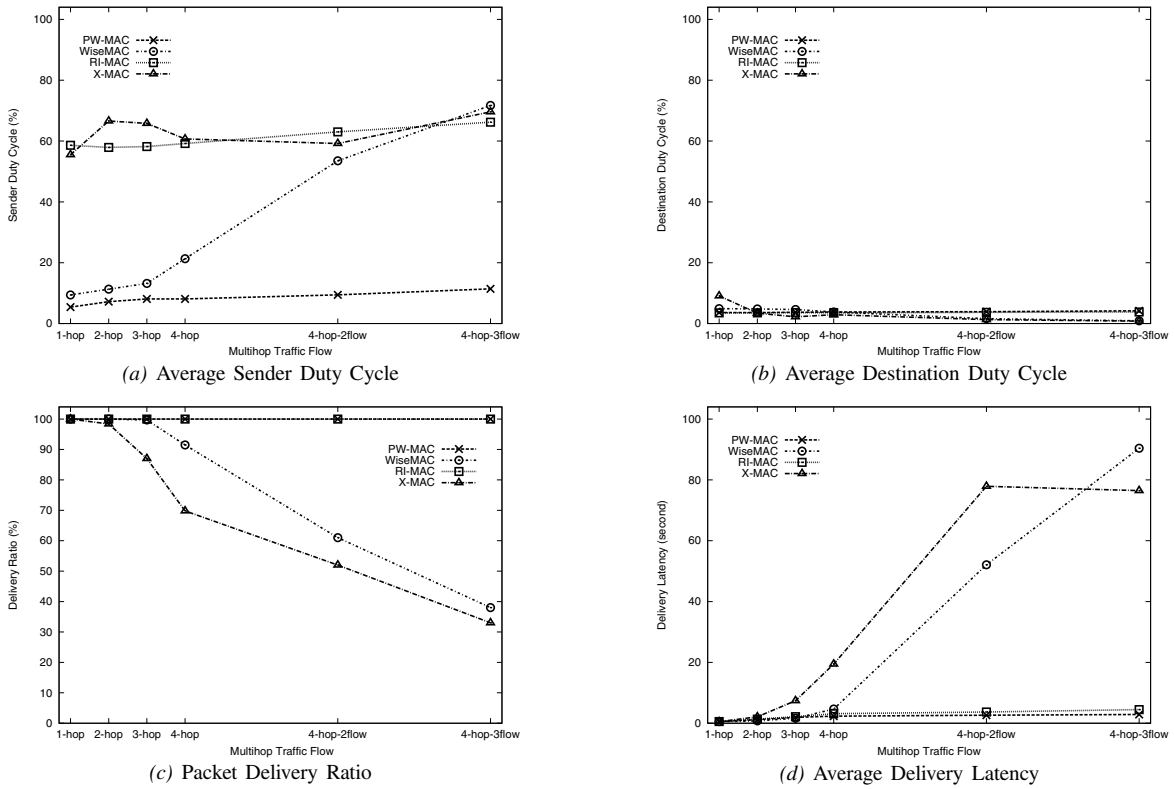
PW-MAC achieved a much smaller sender duty cycle than the other protocols. As the hop-length of a traffic flow and the number of traffic flows increased, so did the sender duty cycle performance margin of PW-MAC relative to the other MAC protocols, which indicates PW-MAC's high efficiency in handling dynamic traffic loads. When there were three 4-hop traffic flows in the network, the average sender duty cycles of X-MAC, RI-MAC, and WiseMAC were 70%, 66%, and 72%, respectively, whereas the average sender duty cycle of PW-MAC was only 11%.

PW-MAC also achieved the smallest delivery latency and achieved 100% packet delivery ratio. When there were three 4-hop traffic flows, the delivery latency of PW-MAC was less than 5% that of WiseMAC and X-MAC. The delivery latency of PW-MAC is less than that of RI-MAC because in our PW-MAC implementation, if a DATA packet transmission by a node is in progress when the node is scheduled to transmit its wakeup beacon, the beacon is transmitted only after the DATA transmission completes; in our earlier RI-MAC implementation [11], the DATA transmission is canceled and retransmitted later in order to maintain the beacon transmission schedule.

In summary, the significant performance margin of PW-MAC over other energy-efficient sensor network MAC protocols is mainly due to the following reasons. PW-MAC enables senders to accurately predict receiver wakeup times on real hardware and to wake up shortly before the receivers do, thereby minimizing the sender idle listening and overhearing. Furthermore, PW-MAC greatly reduces wireless collisions by effectively spreading node wakeup times. Even when wireless collisions occur and packets must be retransmitted, PW-MAC achieves a small sender duty cycle and packet delivery latency through its prediction-based retransmission mechanism.

V. Conclusion

This paper has presented the design and evaluation of *PW-MAC (Predictive-Wakeup MAC)*, an energy-efficient asynchronous duty-cycling MAC protocol for sensor networks. PW-MAC is designed to minimize energy consumption by enabling senders to predict receiver wakeup times even given the challenges of unpredictable hardware and operating system delay and clock drift. Nodes in PW-MAC wake up according to independently determined pseudo-random wakeup schedules to minimize wireless collisions. Transmission collisions may



(a) Average Sender Duty Cycle

(b) Average Destination Duty Cycle

(c) Packet Delivery Ratio

(d) Average Delivery Latency

Fig. 5. Performance of PW-MAC, WiseMAC, RI-MAC, and X-MAC in a multihop MICAz testbed. The hop-length increases from 1 to 4 and the number of 4-hop traffic flows increases from 1 to 3.

occur in wireless networks, especially with bursty traffic that may be present in a sensor network. PW-MAC introduces an efficient prediction-based retransmission mechanism to achieve high energy efficiency even when wireless collisions occur and packets need to be retransmitted. PW-MAC also introduces an on-demand prediction error correction mechanism that can effectively control the prediction errors caused by hardware and operating system latency and clock drift.

We conducted experiments on a testbed of MICAz motes to evaluate the performance of PW-MAC compared with WiseMAC, RI-MAC, and X-MAC. Our evaluation includes hidden-terminal scenarios, scenarios in which nodes have wakeup schedule collisions, and scenarios with multiple concurrent multihop traffic flows, ranging from 1 to 3 flows of hop-length ranging from 1 to 4. In all experiments, PW-MAC significantly outperformed the other protocols. For example, evaluated on scenarios with 15 concurrent transceivers in the network, the average sender duty cycle for X-MAC, RI-MAC, and WiseMAC were 70%, 66%, and 72%, respectively, whereas average sender duty cycle for PW-MAC was only 11%. The delivery latency for PW-MAC in these scenarios was less than 5% of that of WiseMAC and X-MAC. PW-MAC maintained a 100% packet delivery ratio in all experiments.

REFERENCES

[1] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks. In *SenSys 2006*, pages 307–320, November 2006.

[2] Hui Cao, Kenneth W. Parker, and Anish Arora. O-MAC: a receiver centric power management protocol. In *ICNP '06*, pages 311–320, 2006.

[3] Amre El-Hoiydi and Jean-Dominique Decotignie. WiseMAC: An ultra low power MAC protocol for multi-hop wireless sensor networks. In *ALGOSENSORS 2004*, pages 18–31, July 2004.

[4] Saurabh Ganeriwal, Deepak Ganesan, Hohyun Shim, Vlasios Tsiatsis, and Mani B. Srivastava. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *SenSys 2005*, pages 130–141, 2005.

[5] G. P. Halkes and K. G. Langendoen. Crankshaft: An energy-efficient mac-protocol for dense wireless sensor networks. In *EWSN '07*, 2007.

[6] Kevin Klues, Gregory Hackmann, Octav Chipara, and Chenyang Lu. A component-based architecture for power-efficient media access control in wireless sensor networks. In *SenSys 2007*, pages 59–72, 2007.

[7] Donald E. Knuth. The art of computer programming, third edition, volume 2: Seminumerical algorithms, section 3.2.1: The linear congruential method. pages 10–26. Addison-Wesley, 1997.

[8] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04*, pages 95–107, 2004.

[9] Venkatesh Rajendran, Katia Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *SenSys 2003*, pages 181–192, November 2003.

[10] Yanjun Sun, Shu Du, Omer Gurewitz, and David B. Johnson. DW-MAC: A low latency, energy efficient demand-wakeup mac protocol for wireless sensor networks. In *MobiHoc 2008*, 2008.

[11] Yanjun Sun, Omer Gurewitz, and David B. Johnson. RI-MAC: A receiver initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks. In *SenSys 2008*, 2008.

[12] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of SenSys 2003*, pages 171–180, November 2003.

[13] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of IEEE INFOCOM 2002*, pages 1567–1576, June 2002.

[14] Wei Ye, Fabio Silva, and John Heidemann. Ultra-low duty cycle MAC with scheduled channel polling. In *SenSys 2006*, pages 321–334, 2006.