Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks

Wei Ye, John Heidemann, Deborah Estrin

Abstract—**This paper proposes S-MAC, a medium-access control (MAC) protocol designed for wireless sensor networks. Wireless sensor networks use battery-operated computing and sensing devices. A network of these devices will collaborate for a common application such as environmental monitoring. We expect sensor networks to be deployed in an ad hoc fashion, with individual nodes remaining largely inactive for long periods of time, but then becoming suddenly active when something is detected. These characteristics of sensor networks and applications motivate a MAC that is different from traditional wireless MACs such as IEEE 802.11 in several ways: energy conservation and self-configuration are primary goals, while per-node fairness and latency are less important. S-MAC uses three novel techniques to reduce energy consumption and support self-configuration. To reduce energy consumption in listening to an idle channel, nodes periodically sleep. Neighboring nodes form *virtual clusters* to auto-synchronize on sleep schedules. Inspired by PAMAS, S-MAC also sets the radio to sleep during transmissions of other nodes. Unlike PAMAS, it only uses in-channel signaling. Finally, S-MAC applies *message passing* to reduce contention latency for sensor-network applications that require store-and-forward processing as data move through the network. We evaluate our implementation of S-MAC over a sample sensor node, the UCB Mote. The experimental results show that, on a source node, an 802.11-like MAC consumes 2–6 times more energy than S-MAC for traffic load with messages sent every 1–10s.**

Keywords—**Medium access control, Sensor network, Wireless network, Energy efficiency**

I. Introduction

WIRELESS sensor networking is an emerging technology that has a wide range of potential applications including environment monitoring, smart spaces, medical systems and robotic exploration. Such networks will consist of large numbers of distributed nodes that organize themselves into a multihop wireless network. Each node has one or more sensors, embedded processors and low-power radios, and is normally battery operated. Typically, these nodes coordinate to perform a common task.

Like in all shared-medium networks, medium access control (MAC) is an important technique that enables the successful operation of the network. One fundamental task of the MAC protocol is to avoid collisions so that two interfering nodes do not transmit at the same time. There are many MAC protocols that have been developed for wireless voice and data communication networks. Typical examples include the time division multiple access (TDMA), code division multiple access (CDMA), and contention-based protocols like IEEE 802.11 [1].

This work is in part supported by NSF under grant ANI-0220026 as the MACSS project and under grant ANI-9979457 as the SCOWR project, and by DARPA under grant DABT63-99-1-0011 as the SCADDS project and under contract N66001-00-C-8066 as the SAMAN project. The work is also support by the Center for Embedded Networked Sensing and a grant from the Intel Corporation. W. Ye (weiye@isi.edu) and J. Heidemann (johnh@isi.edu) are with the Information Sciences Institute (ISI), University of Southern California (USC). D. Estrin (destrin@cs.ucla.edu) is with the Center for Embedded Networked Sensing and the Computer Science Department, University of California at Los Angeles and USC/ISI.

To design a good MAC protocol for the wireless sensor networks, we have considered the following attributes. The first is the energy efficiency. As stated above, sensor nodes are likely to be battery powered, and it is often very difficult to change or recharge batteries for these nodes. In fact, someday we expect some nodes to be cheap enough that they are discarded rather than recharged. Prolonging network lifetime for these nodes is a critical issue. Another important attribute is scalability and adaptivity to changes in network size, node density and topology. Some nodes may die over time; some new nodes may join later; some nodes may move to different locations. A good MAC protocol should gracefully accommodate such network changes. Other typically important attributes including fairness, latency, throughput and bandwidth utilization may be secondary in sensor networks.

This paper presents sensor-MAC (S-MAC), a MAC protocol explicitly designed for wireless sensor networks. While reducing energy consumption is the primary goal in our design, our protocol has also achieved good scalability and collision avoidance by utilizing a combined scheduling and contention scheme. To achieve the primary goal of energy efficiency, we need to identify what are the main sources that cause inefficient use of energy as well as what trade-offs we can make to reduce energy consumption.

We have identified the following major sources of energy waste. The first one is *collision*. When a transmitted packet is corrupted it has to be discarded, and the follow-on retransmissions increase energy consumption. Collision increases latency as well. The second source is *overhearing*, meaning that a node picks up packets that are destined to other nodes. The third source is *control packet overhead*. Sending and receiving control packets consumes energy too. The last major source of inefficiency is *idle listening*, *i.e.*, listening to receive possible traffic that is not sent. This is especially true in many sensor network applications. If nothing is sensed, nodes are in idle mode for most of the time. However, in many MAC protocols such as IEEE 802.11 ad hoc mode or CDMA nodes have to listen to the channel to receive possible traffic. Measurements have shown that idle listening consumes 50–100% of the energy required for receiving. For example, Stemm and Katz measure that the idle:receive:send ratios are 1:1.05:1.4 [2], while the Digitan wireless LAN module (IEEE 802.11/2Mbps) specification shows idle:receive:send ratios is 1:2:2.5 [3]. Most sensor networks are designed to operate for long time, and nodes will be in idle state for long time. Thus, idle listening is a dominant factor of energy waste in such cases.

S-MAC tries to reduce energy waste from all the above sources. In exchange it accepts some reduction in both per-hop

fairness and latency. The first technique of S-MAC is to establish the low-duty-cycle operation on nodes in a multi-hop ad hoc network. It reduces idle listening by periodically putting nodes into sleep state. In the sleep mode, the radio is completely turned off. In protocols for traditional data networks like the IEEE 802.11, bandwidth utilization is a big concern, and nodes normally operate in fully active mode. Switching to low-duty-cycle mode (called power save mode in the IEEE 802.11 standard) is an option of each node, and it normally happens when a node has been idle for long time. In S-MAC, however, the low-duty-cycle mode is the default operation of all nodes in the network. Nodes only become more active when there is traffic in the network. To reduce the control overhead and latency, S-MAC introduces coordinated sleeping among neighboring nodes.

An important feature of wireless sensor networks is the in-network data processing. It can greatly reduce energy consumption compared to transmitting all the *raw* data to the end node [4], [5], [6]. In-network processing requires store-and-forward processing of messages. A message is a meaningful unit of data that a node can process (average or filter, *etc.*). It may be long and consists of many small fragments. In this case, MAC protocols that promote fragment-level fairness actually increase message-level latency for the application. In contrast, message passing reduces message-level latency by trading off the fragment-level fairness.

In traditional wireless voice or data networks, each user desires equal opportunity and time to access the medium, *i.e.*, sending or receiving packets for their own applications. Per-hop MAC level fairness is thus an important issue. However, in sensor networks, all nodes cooperate for a single common task. At any particular time, one node may have dramatically more data to send than some other nodes. In this case fairness is not important as long as application-level performance is not degraded. In our protocol, we re-introduce the concept of message passing to efficiently transmit very long messages. The basic idea is to divide the long message into small fragments and transmit them in a burst. The result is that a node who has more data to send gets more time to access the medium. From a per-hop, MAC level perspective, this is unfair for those nodes who only have some short packets to send, since their short packets have to wait a long time for very long packets. However, as we will show later, message passing can achieve energy savings by reducing control overhead and avoiding overhearing. And it is well suited to applications where nodes support in-network processing of data.

Latency can be important or unimportant depending on what application is running and the node state. During a period that there is no sensing event, there is normally very little data flowing in the network. Most of the time nodes are in idle state. Sub-second latency is not important, and we can trade it off for energy savings. S-MAC therefore lets nodes periodically sleep if otherwise they are in the idle listening mode. In the sleep mode, a node will turn off its radio. The design reduces the energy consumption due to idle listening. However, the latency is increased, since a sender must wait for the receiver to wake up before it can send out data.

To demonstrate the effectiveness and measure the performance of S-MAC, we have implemented it on our testbed wireless sensor nodes, *Motes*, developed by University of California, Berkeley [7] and manufactured and sold by Crossbow Technology, Inc. [8] The latest version of mote, Mica, has a 8-bit Atmel ATmega128L microcontroller running at 4 MHz. It has a low power radio transceiver module TR1000 [9] or TR3000 [10] from RF Monolithics, Inc., which operates at 916.5 MHz or 433.92MHz. The mote runs on a very small event-driven operating system called TinyOS [11]. In order to compare the performance of our protocol with some other protocols, we also implemented an IEEE 802.11-like MAC protocol on this platform.

The contributions of this paper are listed as follows.

- The scheme of periodic listen and sleep reduces energy consumption by avoiding idle listening. The use of synchronization to form virtual clusters of nodes on the same sleep schedule. These schedules coordinate nodes to minimize additional latency.
- The use of in-channel signaling to put each node to sleep when its neighbor is transmitting to another node. This method avoids the overhearing problem and is inspired by PAMAS [12], but does not require an additional channel.
- Applying message passing to reduce application-perceived latency and control overhead. Per-node fragment-level fairness is reduced since sensor network nodes are often collaborating towards a single application.
- Evaluating an implementation of our new MAC over sensornet specific hardware.

The early work of S-MAC was published in [13]. This paper includes significant extensions in the protocol design, implementation and experiments:

- Support for traffic-adaptive sleep schedules.
- Measurement and evaluation of the trade-offs on energy, latency and throughput.

II. Related Work

Medium access control is a broad research area, including work in the new area of low power and wireless sensor networks [14], [15], [16], [17]. Current MAC design for wireless sensor networks can be broadly divided into contention-based and TDMA protocols.

Contention-based MACs. The standardized IEEE 802.11 distributed coordination function (DCF) [1] is an example of the contention-based protocol, and is mainly built on the research protocol MACAW [18]. It is widely used in ad hoc wireless networks because of its simplicity and robustness to the hidden terminal problem. However, recent work [2] has shown that the energy consumption using this MAC is very high when nodes are in idle mode. This is mainly due to the idle listening. 802.11 has a power save mode, and we will discuss it shortly. PAMAS [12] made an improvement on energy savings by trying to avoid the overhearing among neighboring nodes. Our paper also exploits the same idea. The main difference of our work with PAMAS is that we do not use any out-of-channel signaling. Whereas in PAMAS, it requires two independent radio channels, which in most cases indicates two independent radio systems on each node. PAMAS does not attempt to reduce idle listening.

*TDMA-based MACs.*The other class of MAC protocols are based on reservation and scheduling, for example TDMA-based protocols. TDMA protocols have a natural advantage of energy conservation compared to contention protocols, because the duty cycle of the radio is reduced and there is no contention-introduced overhead and collisions. However, using TDMA protocol usually requires the nodes to form *real* communication clusters, like Bluetooth [19], [20] and LEACH [16]. Most nodes in a real cluster are restricted to communicate within the cluster. Managing inter-cluster communication and interference is not an easy task. Moreover, when the number of nodes within a cluster changes, it is not easy for a TDMA protocol to dynamically change its frame length and time slot assignment. So its scalability is normally not as good as that of a contention-based protocol. For example, Bluetooth may have at most 8 active nodes in a cluster.

Sohrabi and Pottie [15] proposed a self-organization protocol for wireless sensor networks. Each node maintains a TDMA-like frame, called super frame, in which the node schedules different time slots to communicate with its known neighbors. At each time slot, it only talks to one neighbor. To avoid interference between adjacent links, the protocol assigns different channels, *i.e.*, frequency (FDMA) or spreading code (CDMA), to potentially interfering links. Although the super frame structure is similar to a TDMA frame, it does not prevent two interfering nodes from accessing the medium at the same time. The actual multiple access is accomplished by FDMA or CDMA. A drawback of the scheme is its low bandwidth utilization. For example, if a node only has packets to be sent to one neighbor, it cannot reuse the time slots scheduled to other neighbors.

Woo and Culler [17] examined different configurations of carrier sense multiple access (CSMA) and proposed an adaptive rate control mechanism, whose main goal is to achieve fair bandwidth allocation to all nodes in a multi-hop network. They have used the motes and TinyOS platform to test and measure different MAC schemes. In comparison, our approach does not promote per-node fairness, and even trades it off for further energy savings.

Finally, we look at some work on low-duty-cycle operation of nodes, which are closely related to S-MAC. The first example is Piconet [14], which is an architecture designed for low-power ad hoc wireless networks. Piconet also puts nodes into periodic sleep for energy conservation. However, there is not any coordination and synchronization among neighboring nodes about their sleep and listen time. The scheme that Piconet uses to enable the communications among neighboring nodes is to let a node broadcast its address before it starts listening. If a sender wants to talk to a neighboring node, it must wait until it receives the neighbor's broadcast. In contrast, S-MAC tries to coordinate and synchronize neighbors' sleep schedules to reduce latency and control overhead.

Perhaps the power save (PS) mode in IEEE 802.11 DCF is the most related work to the low-duty-cycle operation in S-MAC. Nodes in PS mode periodically listen and sleep, just like that in S-MAC. The sleep schedules of all nodes in the network are synchronized together. The main difference to S-MAC is that the PS mode in 802.11 is designed for a single-hop network, where all nodes can hear each other, simplifying the synchronization. As previously observed by [21], in multi-hop operation, the 802.11 PS mode may have problems in clock synchronization, neighbor discovery and network partitioning. In fact, the 802.11 MAC in general is designed for a single-hop network, and there are questions about its performance in multi-hop networks [22]. In comparison, S-MAC is designed to operate in a multi-hop network, and does not assume that all nodes are synchronized together. Finally, although 802.11 defines PS mode, it provides very limited policy about *when* to sleep. Whereas in S-MAC, we define and measure a complete system.

Tseng *et al.* [21] proposed three sleep schemes to improve the PS mode in the IEEE 802.11 for its operation in multi-hop networks. Among them the one named periodically-fully-awake-interval is the most closest to the scheme of periodic listen and sleep in S-MAC. However, their scheme does not synchronize the sleep schedules of any neighboring nodes. The control overhead and latency can be large. For example, to send a broadcast packet, the sender has to explicitly wake up each individual neighbor before it sends out the actual packet. Without synchronization, each node has to send beacons more frequently to prevent long-time clock drift.

III. S-MAC Design Overview

S-MAC includes approaches to reduce energy consumption from all the sources that we have identified to cause energy waste, *i.e.*, idle listening, collision, overhearing and control overhead. Before describing the components in S-MAC, we first summarize our assumptions about the wireless sensor network and its applications.

Sensor networks will be composed of many small nodes to take advantage of physical proximity to the target to simplify signal processing. The large number of nodes can also take advantage of short-range, multi-hop communications to conserve energy [4]. Most communications will be between nodes as peers, rather than to a single base-station. Because there are many nodes, they will be deployed casually in an ad hoc fashion, rather than carefully positioned. Nodes must therefore self-configure.

In-network processing is critical to sensor network lifetime [5], [6]. Since sensor networks are committed to one or a few applications, application-specific code can be distributed through the network and activated when necessary or distributed on-demand. Techniques such as data aggregation can reduce traffic, while collaborative signal processing can reduce traffic and improve sensing quality. In-network processing implies that data will be processed as whole messages at a time in store-and-forward fashion, so packet or fragment-level interleaving from multiple sources only increases overall latency.

Finally, we expect that applications will have long idle periods and can tolerate some latency. In sensor networks, the application such as surveillance or monitoring will be vigilant for long periods of time, but largely inactive until something is detected. For such applications, network lifetime is critical. These classes of applications can often also tolerate some additional latency. For example, the speed of the sensed object places a bound on how rapidly the network must detect an object.

Fig. 1. Periodic listen and sleep.

Fig. 2. Neighboring nodes A and B have different schedules. They synchronize with nodes C and D respectively.

These assumptions about the network and application strongly influence our MAC design and motivate its differences from existing protocols such as IEEE 802.11.

A. Periodic Listen and Sleep

As stated above, in many sensor network applications, nodes are in idle for a long time if no sensing event happens. Given the fact that the data rate during this period is very low, it is not necessary to keep nodes listening all the time. Our protocol reduces the listen time by letting node go into periodic sleep mode. For example, if in each second a node sleeps for half second and listens for the other half, its duty cycle is reduced to 50%.

The basic scheme is shown in Figure 1. Each node goes to sleep for some time, and then wakes up and listens to see if any other node wants to talk to it. During sleep, the node turns off its radio, and sets a timer to awake itself later.

We call a complete cycle of listen and sleep a *frame*. The duration of listen is normally fixed according to physical-layer and MAC-layer parameters, *e.g.*, the radio bandwidth and the contention window size. The sleep interval can be changed according to different application requirements, which actually changes the duty cycle. For simplicity these values are the same for all nodes.

All nodes are free to choose their own listen/sleep schedules. However, to reduce control overhead, we prefer neighboring nodes to synchronize together. That is, they listen at the same time and go to sleep at the same time. It should be noticed that not all neighboring nodes can synchronize together in a multi-hop network. Two neighboring nodes A and B may have different schedules if they each in turn must synchronize with different nodes, C and D, respectively, as shown in Figure 2.

Nodes exchange their schedules by periodically broadcasting a SYNC packet to their immediate neighbors. This ensures that all neighboring nodes can talk to each other even if they have different schedules. For example, in Figure2 if node A wants to talk to node B, it waits until B is listening. The period for each node to send a SYNC packet is called the *synchronization period*.

One characteristic of our scheme is that it forms nodes into a flat topology. Neighboring nodes are free to talk to each other no matter what listen schedules they have. Synchronized nodes from a virtual cluster. But there is no real clustering and thus no problems of inter-cluster communications and interference. This scheme is adaptive to topology changes. We will talk about this issue later.

The downside of the scheme is that the latency is increased due to the periodic sleep of each node. Moreover, the delay can accumulate on each hop. However, in Section IV we present a technique that significantly reduces such latency.

B. Collision Avoidance

If multiple neighbors want to talk to a node at the same time, they will try to send when the node starts listening. In this case, they need to contend for the medium to avoid collisions. Among contention based protocols, the 802.11 does a very good job of collision avoidance. Our protocol follows similar procedures, including both virtual and physical carrier sense and RTS/CTS exchange. We adopt the RTS/CTS mechanism to address the hidden terminal problem [18].

There is a duration field in each transmitted packet that indicates how long the remaining transmission will be. So if a node receives a packet destined to another node, it knows how long it has to keep silent. The node records this value in an variable called the network allocation vector (NAV) [1] and sets a timer for it. Every time when the NAV timer fires, the node decrements the NAV value until it reaches zero. When a node has data to send, it first looks at the NAV. If its value is not zero, the node determines that the medium is busy. This is called virtual carrier sense.

Physical carrier sense is performed at the physical layer by listening to the channel for possible transmissions. The procedure includes a randomized carrier sense time, which is very important for collision avoidance. The medium is determined as free if both virtual and physical carrier sense indicate that it is free.

All senders perform carrier sense before initiating a transmission. If a node fails to get the medium, it goes to sleep and wakes up when the receiver is free and listening again. Broadcast packets are sent without using RTS/CTS. Unicast packets follow the sequence of RTS/CTS/DATA/ACK between the sender and the receiver. After the successful exchange of RTS and CTS, the two nodes will use their normal sleep time for data packet transmission. They do not follow their sleep schedules until they finish the transmission.

With the low-duty-cycle operation and the contention mechanism during each listen interval, S-MAC effectively addresses the energy waste due to idle listening and collisions. In the next section, we will present details of the periodic sleep coordinated among neighboring nodes. Then we will present two techniques that further reduce the energy waste due to overhearing and control overhead.

IV. Coordinated Sleeping

Periodic listen and sleep is an effective way to avoid idle listening, which is a major source of energy waste in wireless sensor networks. In S-MAC, nodes coordinate on their sleep schedules rather than randomly sleep on their own. This section details the procedures that all nodes follow to set up and maintain

their schedules. It also presents a technique to reduce latency due to the period sleep on each node.

A. Choosing and Maintaining Schedules

Before each node starts its periodic listen and sleep, it needs to choose a schedule and exchange it with its neighbors. Each node maintains a *schedule table* that stores the schedules of all its known neighbors. It follow the steps below to choose its schedule and establish its schedule table.

1. A node first listens for a fixed amount of time, which is at least the synchronization period. If it does not hear a schedule from another node, it immediately chooses its own schedule and starts to follow it. Meanwhile, the node tries to announce the schedule to its neighbors by broadcasting a SYNC packet. Broadcasting of SYNC packets must follow the normal carrier sense procedure. The randomized carrier sense time will reduce the chance of collisions on SYNC packets.
2. If the node receives a schedule from a neighbor before choosing or announcing its own schedule, it follows that schedule by setting its schedule to be the same. Then the node will try to announce its schedule at its next scheduled listen time.
3. There are two cases if a node receives a different schedule after it chooses and announces its own schedule. If the node has no other neighbors, it will discard its current schedule and follow the new one. If the node has one or more neighbors, *i.e.*, it is already a part of a network, it adopts both schedules by waking up at the listen intervals of two different schedules.

To illustrate this algorithm, consider a network where all nodes can hear each other.The node who starts first will have its timer fired fist, and its broadcast will synchronize all of its peers on its schedule. If two or more nodes start first at the same time, their timer will fire at the same time, and they will choose the same schedule independently. No matter which node sends out its SYNC packet first (wins the medium in carrier sense), it will synchronize the rest of the nodes.

We expect that nodes only rarely adopt multiple schedules, since every node tries to follow an existing schedule before choosing an independent one. However, two nodes may independently assign schedules either because they cannot hear each other in a multi-hop network or because they happen to transmit at nearly the same time. In this case, those nodes on the border between the two schedules will adopt both. In this way, when a border node sends a broadcast packet, it only needs to send it once. The disadvantage is that these border nodes have less time to sleep and consume more energy than others.

Another option is to let the nodes on the border adopt only one schedule, which is the one it receives first. Since it knows another schedule that some other neighbors follow, it can still talk to them. However, for broadcast packets, it needs to send twice to the two different schedules. The advantage is that the border nodes have the same simple pattern of periodic listen and sleep as other nodes.

When a new node starts, it needs to listen for at least a synchronization period. This provides a high probability that the node will follow an existing neighbor if there is one. However, it is still possible that a new node fails to discover an existing neighbor. First, the SYNC packet from the neighbor could be corrupted by collisions or interference. Second, the neighbor may not be able to send out a SYNC packet on time because the medium keeps busy. Finally, if the new node is on the border of two different schedules in a multi-hop network, it may only discover the first schedule if the two schedules do not overlap and the node immediately follows the first one after it is discovered.

To prevent the case that two neighbors could not find each other forever when they follow completely different schedules, S-MAC introduces periodic neighbor discovery, *i.e.*, each node periodically listens for the whole synchronization period. The frequency with which a node performs neighbor discovery depends on the number of neighbors it has. If a node does not have any neighbor, it performs neighbor discovery more aggressively than in the case that it has many neighbors. Since the energy cost is high during the neighbor discovery, it should not be performed too often. In our current implementation, the synchronization period is 10 seconds, and a node performs neighbor discovery every 2 minutes if it has at least one neighbor.

B. Maintaining Synchronization

Since the periodic listen and sleep are coordinated among neighboring nodes, the clock drift on each node can cause synchronization errors. We use two techniques to make it robust to such errors. First, all timestamps that are exchanged are relative rather than absolute. Second, the listen period is significantly longer than clock error or drift. For example, the listen duration of 0.5s is more than 10^5 times longer than typical clock drift rates. Compared with TDMA schemes with very short time slots, S-MAC requires much looser synchronization among neighboring nodes.

Although the long listen time can tolerate fairly large clock drift, neighboring nodes still need to periodically update each other with their schedules to prevent long-time clock drift. The synchronization period can be quite long. The measurements on our testbed nodes show that the clock drift between two nodes does not exceed 0.2ms per second.

As mentioned earlier, schedule updating is accomplished by sending a SYNC packet. The SYNC packet is very short, and includes the address of the sender and the time of its next sleep. The next sleep time is relative to the moment that the sender starts transmitting the SYNC packet. When a receiver gets the time from the SYNC packet it subtracts the packet transmission time and use the new value to adjust their timers.

In order for a node to receive both SYNC packets and data packets, we divide its listen interval into two parts. The first part is for receiving SYNC packets, and the second one is for receiving RTS packets, as shown in Figure 3. Each part is further divided into many time slots for senders to perform carrier sense. For example, if a sender wants to send a SYNC packet, it starts carrier sense when the receiver begins listening. It randomly selects a time slot to finish its carrier sense. If it has not detected any transmission by the end of the time slot, it wins the medium and starts sending its SYNC packet at that time. The same procedure is followed when sending data packets.

Figure 3 also shows the timing relationship of three possible situations that a sender transmits to a receiver. CS stands for carrier sense. In the figure, sender 1 only sends a SYNC

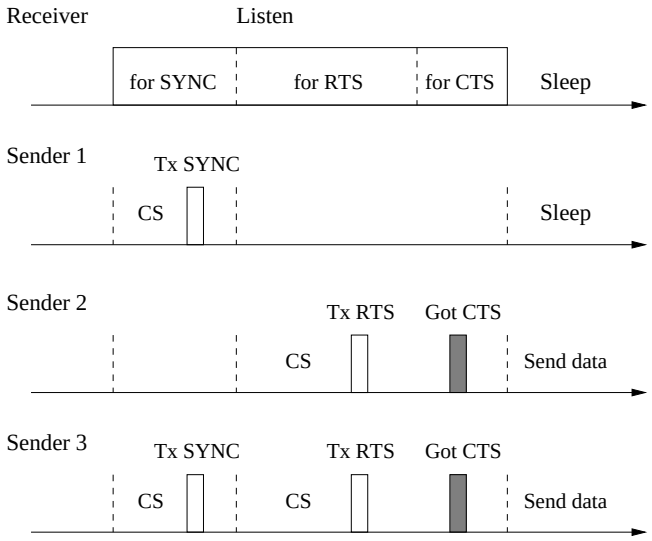

Fig. 3. Timing relationship between a receiver and different senders. CS stands for carrier sense.

packet. Sender 2 only wants to send data. Sender 3 sends a SYNC packet and a RTS packet.

C. Adaptive Listening

The scheme of periodic listen and sleep is able to significantly reduce the time spent on idle listening when traffic load is light. However, when a sensing event indeed happens, it is desirable that the sensing data can be passed through the network without too much delay. When each node strictly follows its sleep schedule, there is a potential delay on each hop, whose average value is proportional to the length of the schedule period. We therefore introduce a mechanism to switch the nodes from the low-duty-cycle mode to a more active mode in this case.

S-MAC proposes an important technique, called *adaptive listen*, to improve the latency caused by the periodic sleep of each node in a multi-hop network. The basic idea is to let the node who overhears its neighbor's transmissions (ideally only RTS or CTS) wake up for a short period of time at the end of the transmission. In this way, if the node is the next-hop node, its neighbor is able to immediately pass the data to it instead of waiting for its scheduled listen time. If the node does not receive anything during the adaptive listening, it will go back to sleep until its next scheduled listen time.

Let us look at the timing diagram in Figure 3 again. If the next- hop node is a neighbor of the sender, it will receive the RTS packet. If it is only a neighbor of the receiver, it will receive the CTS packet from the receiver. Thus, both the neighbors of the sender and receiver will learn from the duration field in the RTS and CTS packets about how long the transmission is. So they are able to adaptively wake up at the appropriate time.

The interval of the adaptive listening does not include the time for the SYNC packet as in the normal listen interval (see Figure 3). SYNC packets are only sent at scheduled listen time to ensure all neighbors can receive it. To give the priority to the SYNC packet, adaptive listen and transmission are not performed if the duration from the time the previous transmission is finished to the normally scheduled listen time is shorter than the adaptive listen interval.

It should be noted that not all next-hop nodes can overhear a packet from the previous transmission, especially when the previous transmission starts adaptively, *i.e.*, not at the scheduled listen time. So if a sender starts a transmission by sending out an RTS packet during the adaptive listening, it might not get a CTS reply. In this case, it just goes back to sleep and will try again at the next normal listen time.

D. Latency Analysis

This subsection analyzes the multi-hop latency of MAC protocols, and quantifies the delay introduced by periodic sleeping in S-MAC. For a packet moving through a multi-hop network, it experiences the following delays at each hop:

Carrier sense delay is introduced when the sender performs carrier sense. Its value is determined by the contention window size.

Backoff delay happens when carrier sense fails, either because the node detects another transmission or because collision occurs.

Transmission delay is determined by channel bandwidth, packet length and the coding scheme adopted.

Propagation delay is determined by the distance between the sending and receiving nodes. In sensor networks, node distance is normally very small, and the propagation delay can normally be ignored.

Processing delay. The receiver needs to process the packet before forwarding it to the next hop. This delay mainly depends on the computing power of the node and the efficiency of in-network data processing algorithms.

Queuing delay depends on the traffic load. In the heavy traffic case, queuing delay becomes a dominant factor.

The above delays are inherent to a multi-hop network using contention-based MAC protocols. These factors are the same for both S-MAC and 802.11-like protocols. An extra delay in S-MAC is caused by the periodic sleeping of each node. When a sender gets a packet to transmit, it must wait until the receiver wakes up. We call it *sleep delay* since it is caused by the sleep of the receiver.

We analyze the latency of different MAC protocols in the simple case that the traffic load is very light, *e.g.*, only one packet is moving through the network, so that there is no queuing delay and backoff delay. We further assume that the propagation delay and the processing delay can be ignored. In this case, only carrier sense delay, transmission delay and sleep delay are taken into account.

Suppose there are N hops from the source to the sink. The carrier sense delay is random at each hop, and we denote its value at hop n by $t_{cs,n}$. Its mean value is determined by the contention window size, and is denoted by t_{cs}. The transmission delay is fixed if the packet length is fixed, which is denoted by t_{tx}.

We first look at the MAC protocol without sleeping. When a node receives a packet, it immediately starts carrier sense and

tries to forward it to the next hop. The average delay at hop n is $t_{cs,n} + t_{tx}$. The entire latency over N hops is

$$D(N) = \sum_{n=1}^{N} (t_{cs,n} + t_{tx}) \quad (1)$$

So the average latency over N hops in the MAC without sleeping is

$$E[D(N)] = N(t_{cs} + t_{tx}) \quad (2)$$

Equation (2) shows that, in the MAC protocol without sleeping, the multi-hop latency linearly increases with the number of hops. The slope of the line is the average carrier sense time plus the packet transmission time.

Now we look at S-MAC, which introduces a sleep delay at each hop, denoted by $t_{s,n}$ for the nth hop. For simplicity, we assume that all nodes along the path follow the same sleep schedule. A frame is a complete cycle of listen and sleep, and its length is denoted by T_f. Recall that the listen interval is fixed, and the frame length can be changed by adjusting the sleep interval. To reflect a very low duty cycle, *e.g.*, $\leq 10\%$, we assume that T_f has a large value, which is much larger than t_{tx}. The delay at hop n is

$$D_n = t_{s,n} + t_{cs,n} + t_{tx} \quad (3)$$

In S-MAC without adaptive listening, contention (carrier sense) only starts at the beginning of each frame, *i.e.*, the time each node starts listening. After a node receives a packet in a frame, it has to wait until the next-hop node to wake up, which is the beginning of the next frame. This indicates

$$T_f = t_{cs,n-1} + t_{tx} + t_{s,n} \quad (4)$$

So the sleep delay at hop n is

$$t_{s,n} = T_f - (t_{cs,n-1} + t_{tx}) \quad (5)$$

Substituting by Equation (5), Equation (3) becomes

$$D_n = T_f + t_{cs,n} - t_{cs,n-1} \quad (6)$$

There is an exception on the first hop, because a packet can be generated on the source node at any time within a frame. So the sleep delay on the first hop, $t_{s,1}$, is a random variable whose value lies in $(0, T_f)$. Suppose $t_{s,1}$ is uniformly distributed in $(0, T_f)$. So its mean value is $T_f/2$. Combining it with Equation (6), we have the overall delay of a packet over N hops as

$$\begin{aligned} D(N) &= D_1 + \sum_{n=2}^{N} D_n \\ &= t_{s,1} + t_{cs,1} + t_{tx} + \sum_{n=2}^{N} (T_f + t_{cs,n} - t_{cs,n-1}) \\ &= t_{s,1} + (N-1)T_f + t_{cs,N} + t_{tx} \end{aligned} \quad (7)$$

So the average latency of S-MAC without adaptive listen over N hops is

$$\begin{aligned} E[D(N)] &= E[t_{s,1} + (N-1)T_f + t_{cs,N} + t_{tx}] \\ &= T_f/2 + (N-1)T_f + t_{cs} + t_{tx} \\ &= NT_f - T_f/2 + t_{cs} + t_{tx} \end{aligned} \quad (8)$$

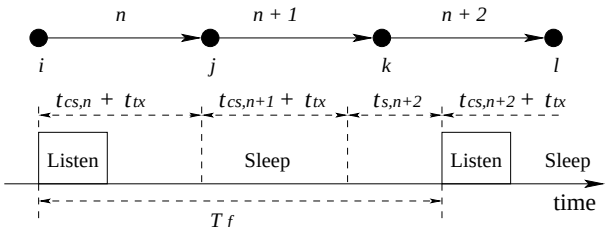

Fig. 4. Adaptive listen can reduce sleep latency by at least half.

Equation (8) shows that the multi-hop latency also linearly increases with the number of hops in S-MAC without adaptive listen, *i.e.*, each node strictly follows its sleep schedules. The slope of the line is the frame length T_f. Compared with Equation (2), T_f is normally much larger than $(t_{cs} + t_{tx})$ due to the very low duty cycles. Therefore, periodic sleeping introduces an additional delay at each hop.

However, with adaptive listening S-MAC can reduce the latency introduced by periodic sleeping by at least half. Figure 4 shows part of a multi-hop network, where the three hops are denoted as n to $(n+2)$. Again, we assume all nodes follow the same sleep schedule.

Suppose node i first waits for node j to wake up at its normally scheduled listen time, and starts carrier sense for sending data from that moment. The delay at hop n is still expressed as Equation (3).

During the RTS/CTS exchange from nodes i and j, the next hop node k is also listening, and overhears j's CTS packet. So node k knows when the transmission from i to j will finish. The adaptive listen mechanism will wake up node k immediately after the previous transmission is done. It also lets node j start carrier sense for sending to k at that time. Thus the delay at hop $(n+1)$ is

$$D_n = t_{cs,n+1} + t_{tx} \quad (9)$$

Compared with the delay at the previous hop, there is no sleep delay here. If the frame length T_f is larger than $(t_{cs,n} + t_{cs,n+1} + 2t_{tx})$, the packet will travel over two hops in just one frame. We assume this condition holds in the following analysis, since we have assumed that T_f is much larger than t_{tx}.

On the other hand, node l is two-hop away from node j. It may not be able to overhear j's CTS packet as k does. In this case, l cannot wake up when the transmission from i to j is done. When j starts sending to k during the normal sleep time, node l is not aware of it, since it is in sleep state. Therefore, node l will not be able to wake up when the transmission from j to k is done. Node k has to wait until l's normal listen time to start its transmission. The delay on hop $(n+2)$ is again expressed by Equation (3). However, as we just analyzed, its next hop $(n+3)$ will not have the sleep delay.

The latency over N hops can thus be calculated as

$$\begin{aligned} D(N) = t_{s,1} + t_{cs,1} + t_{tx} + t_{cs,2} + t_{tx} + t_{s,3} + \\ ... + t_{cs,N-1} + t_{tx} + t_{cs,N} + t_{tx} \end{aligned} \quad (10)$$

Note that (see Figure 4)

$$T_f = t_{cs,n} + t_{tx} + t_{cs,n+1} + t_{tx} + t_{s,n+2} \quad (11)$$

Equation (10) can be simplified as

$$D(N) = t_{s,1} + (N/2-1)T_f + t_{cs,N-1} + t_{cs,N} + 2t_{tx} \quad (12)$$

Hence the average latency over N hops in S-MAC with adaptive listen is

$$\begin{aligned} E\left[D(N)\right] &= T_f/2 + (N/2-1)T_f + 2t_{cs} + 2t_{tx} \\ &= NT_f/2 + 2t_{cs} + 2t_{tx} - T_f/2 \end{aligned} \quad (13)$$

We can see that the average latency in S-MAC with adaptive listen still linearly increases with the number of hops. Now the slope of the line is $T_f/2$. Compared with that of no adaptive listen (Equation (8)), it is reduced by half.

Equation (13) is obtained under the assumption that only 1-hop neighbors can hear each other, but 2-hop neighbors cannot hear each other. In real world this is not true in general. The theory and measurement results about radio propagation [23] have shown that the received signal power P_r decreases with the distance d as

$$P_r \propto P_t d^{\beta} \quad (14)$$

where P_t is the transmission power, and β is an environment-dependent constant normally between 2–5 [23]. It is clear that the transmission range does not suddenly stops at a certain distance.

Let us look at Figure 4 again. If node k can reliably receive from node j, say with correct reception rate of over 95%, node l may still have good chances to receive some of j's CTS packets (especially RTS and CTS packets are very short). If two-hop neighbors have 20%–30% probability to receive from each other, the overall latency can be further reduced, since some 2-hop-away nodes are also able to participate in adaptive listening.

V. Overhearing Avoidance and Message Passing

Collision avoidance is a basic task of MAC protocols. S-MAC adopts a contention-based scheme. It is common that any packet transmitted by a node is received by all its neighbors even though only one of them is the intended receiver. Overhearing makes contention-based protocols less efficient in energy than TDMA protocols.

A. Overhearing Avoidance

In 802.11 each node keeps listening to all transmissions from its neighbors in order to perform effective virtual carrier sensing. As a result, each node overhears many packets that are not directed to itself. This is a significant waste of energy, especially when node density is high and traffic load is heavy.

Our protocol tries to avoid overhearing by letting interfering nodes go to sleep after they hear an RTS or CTS packet. Since DATA packets are normally much longer than control packets, the approach prevents neighboring nodes from overhearing long DATA packets and the following ACKs. In the next subsection we describe how to efficiently transmit a long packet combining with the overhearing avoidance. Now we look at which nodes should go to sleep when there is an active transmission in progress.

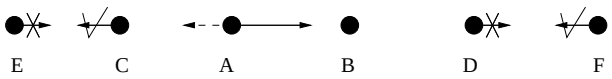

Fig. 5. Who should sleep when node A is transmitting to B?

As shown in Figure 5, node A, B, C, D, E, and F forms a multi-hop network where each node can only hear the transmissions from its immediate neighbors. Suppose node A is currently transmitting a data packet to B. The question is, which of the remaining nodes should go to sleep now.

Remember that collision happens at the receiver. It is clear that node D should go to sleep since its transmission interferes with B's reception. It is easy to show that node E and F do not produce interference, so they do not need to go to sleep. Should node C go to sleep? C is two-hop away from B, and its transmission does not interfere with B's reception, so it is free to transmit to its other neighbors like E. However, C is unable to get any reply from E, *e.g.*, CTS or data, because E's transmission collides with A's transmission at node C. So C's transmission is simply a waste of energy. In summary, all immediate neighbors of both the sender and the receiver should sleep after they hear the RTS or CTS packet until the current transmission is over.

Each node maintains the NAV to indicate the activity in its neighborhood. When a node receives a packet destined to other nodes, it updates its NAV by the duration field in the packet. A non-zero NAV value indicates that there is an active transmission in its neighborhood. The NAV value decrements every time when the NAV timer fires. Thus a node should sleep to avoid overhearing if its NAV is not zero. It can wake up when its NAV becomes zero.

B. Message Passing

This subsection describes how to efficiently transmit a long message in both energy and latency. A *message* is the collection of meaningful, interrelated units of data. It can be a long series of packets or a short packet, and usually the receiver needs to obtain all the data units before it can perform in-network data processing or aggregation.

The disadvantages of transmitting a long message as a single packet is the high cost of re-transmitting the long packet if only a few bits have been corrupted in the first transmission. However, if we fragment the long message into many independent small packets, we have to pay the penalty of large control overhead and longer delay. It is so because the RTS and CTS packets are used in contention for each independent packet.

Our approach is to fragment the long message into many small fragments, and transmit them in burst. Only one RTS packet and one CTS packet are used. They reserve the medium for transmitting all the fragments. Every time a data fragment is transmitted, the sender waits for an ACK from the receiver. If it fails to receive the ACK, it will extend the reserved transmission time for one more fragment, and re-transmit the current fragment immediately.

As before, all packets have the duration field, which is now the time needed for transmitting all the remaining data fragments and ACK packets. If a neighboring node hears a RTS

or CTS packet, it will go to sleep for the time that is needed to transmit all the fragments.

Switching the radio from sleep to active does not occur instantaneously. For example, the RFM radio on our testbed needs 20μs to switch from sleep mode to receive mode [9]. Therefore, it is desirable to reduce the frequency of switching modes. The message passing scheme tries to put nodes into sleep state as long as possible, and hence reduces switching overhead.

The purpose of using ACK after each data fragment is to prevent the hidden terminal problem. It is possible that a neighboring node wakes up or a new node joins in the middle of a transmission. If the node is only the neighbor of the receiver but not the sender, it will not hear the data fragments being sent by the sender. If the receiver does not send ACK frequently, the new node may mistakenly infer from its carrier sense that the medium is clear. If it starts transmitting, the current transmission will be corrupted at the receiver.

Each data fragment and ACK packet also has the duration field. In this way, if a node wakes up or a new node joins in the middle, it can properly go to sleep no matter if it is the neighbor of the sender or the receiver. For example, suppose a neighboring node receives an RTS from the sender or a CTS from the receiver, it goes to sleep for the entire message time. If the sender extends the transmission time due to fragment losses or errors, the sleeping neighbor will not be aware of the extension immediately. However, the node will learn it from the extended fragments or ACKs when it wakes up.

It is worth to note that IEEE 802.11 also has fragmentation support. However, in 802.11 the RTS and CTS only reserves the medium for the first data fragment and the first ACK. The first fragment and ACK then reserves the medium for the second fragment and ACK, and so forth. So for each neighboring node, after it receives a fragment or an ACK, it knows that there is one more fragment to be sent. So it has to keep listening until all the fragments are sent. Again, for energy-constrained nodes, overhearing by all neighbors wastes a lot of energy.

802.11 is designed to promote fairness. If the sender fails to get an ACK for any fragment, it must give up the transmission and re-contend for the medium. So other nodes have a chance to transmit. This causes a long delay if the receiver really needs the entire message to start processing. In contrast, message passing extends the transmission time and re-transmits the current fragment. Thus it has fewer contentions and a small latency. There should be a limit on how many extensions can be made for each message in case that the receiver is really dead or lost in connection during the transmission. However, for sensor networks, application-level fairness is the goal as opposed to per-node fairness.

VI. Protocol Implementation

The purpose of our implementation is to demonstrate the effectiveness of S-MAC and to compare it with protocols that do not have all the energy-conserving features of S-MAC.

We use *Motes*, developed at UCB [7], as our development platform and testbed. The motes are running TinyOS, an efficient event-driven operating system [11], [24]. It provides the basic mechanism for packet transmitting, receiving and processing. TinyOS promotes modularity, data sharing and reuse.

Before presenting the details of our current implementation, we first briefly describe an early implementation.

A. First Implementation on Rene Motes

Our early implementation of S-MAC is on Rene Motes, which has the Atmel AT90LS8535 microcontroller [25] with 8KB of programmable flash and 512B of data memory.

The radio transceiver on Rene motes is the model TR1000 from RF Monolithics, Inc [9]. It uses the OOK(on-off keyed) modulation, and provides a maximum transmission rate of 19.2 Kbps. The actual radio bandwidth implemented in the TinyOS is 10Kbps. The transceiver has three working modes, *i.e.*, receiving, transmitting and sleep, each drawing the input current of 4.5mA, 12mA (peak) and 5μA respectively.

We have implemented three MAC modules on Rene motes, as listed below.

1. 802.11-like protocol without sleep
2. S-MAC without periodic sleep
3. S-MAC with periodic sleep

For the purpose of performance comparison, we implemented an 802.11-like protocol on Rene motes. It has the following pieces as in IEEE 802.11 DCF: physical and virtual carrier sense, backoff and retry, RTS/CTS/DATA/ACK packet exchange, and fragmentation support. In this protocol, nodes never go to sleep. They are either in listen/receiving mode or in transmitting mode.

In the second module, the periodic sleeping is disabled so that each node runs in fully active mode. However, the techniques of overhearing avoidance and message passing are still there. Each node goes into the sleep mode only when its neighbors are in transmission.

The third module is the S-MAC with periodic sleep. However, adaptive listen was not implemented at that time. The listen time in each cycle is 300ms. The sleep time can be changed to different values according to different duty cycles of the radio. The period for each node to send a SYNC packet is 13 seconds.

B. Current Implementation on Mica Motes

Our current implementation is on the new generation of Motes, the Mica, which has the Atmel ATmega128L microcontroller with 128KB of flash and 4KB of data memory. Our Mica motes are equipped with the TR3000 radio transceiver from RF Monolithics, Inc. and a matched whip antenna (see Figure 6). The modulation scheme used by Mica motes is the amplitude shift keying (ASK).

Our implementation of S-MAC is not based on the standard communication stack that comes with the TinyOS release. Instead, we have implemented a new communication stack with some new features and functionality that are critical to S-MAC implementation.

First of all, our stack adopts a layered architecture between MAC and the physical layer. The layers are intended to provide standard interfaces and services, so that various protocols at different levels can be developed in parallel. Our stack clearly separates the functions of the physical layer and the MAC layer.

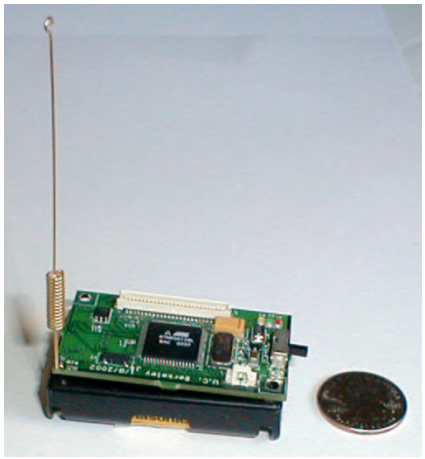

Fig. 6. The UCB Mica Mote with a whip antenna.

TABLE I
IMPORTANT PARAMETERS OF OUR IMPLEMENTED COMMUNICATION STACK: PHYSICAL LAYER AND S-MAC

Radio bandwidth	20Kbps
Channel coding	Manchester
Control packet length	10 bytes
Data packet length	up to 250 bytes
Duration of periodic listen	115 ms
Duty cycle	1% to 99%

The physical layer directly controls the radio and provides APIs for upper layers to put the radio into different states: sleep, idle, transmission and reception. It does start symbol detection, channel coding and decoding, byte buffering, and CRC check. It also provides the carrier sense functionality, but it gives the full control to the MAC layer.

Our stack uses a nested header structure for packet definition. It allows each component to freely define its own packet type as well as add its header fields to a packet comes from its upper layer. When a component defines its own packet format or header, it must include its immediate lower layer's header as its first field. In this way, each packet buffer includes all header fields from all lower layers. Therefore, it avoids memory copies across layers.

The physical layer supports packets with dramatically different lengths. This is important to a MAC layer like S-MAC, since all the control packets are very short (10 bytes) and data packets can be much longer. The maximum supported packet length is 250 bytes.

We described the details of our stack implementation in [26]. Some important parameters are listed here in Table I. We use Manchester code as the channel coding scheme. It is a robust DC-balanced code, and has a overhead of $1:2$. That is, each data bit becomes 2 bits after encoding. In [26] we compared the performance and overhead of a few coding schemes that has been used in TinyOS.

Our implementation allows a user to configure S-MAC into different modes by selecting different options at compile time. The followings are some important options, which are used in our experiments in the next section.

- Duty cycle selection. This option allows a user to select different duty cycles of S-MAC, from 1% to 99%.
- Fully active mode. This option completely disables the periodic sleep cycles. This mode is mainly used for performance comparison.
- Disable adaptive listen. Adaptive listen is enabled by default in the low-duty-cycle mode. With this option it is disabled, so that each node strictly follows its listen schedules.

Our current implementation coordinates radio sleeping. Other hardware on the node can also be put into sleep, including the CPU. Further work is required to integrate S-MAC and CPU control to maximize energy conservation.

VII. EXPERIMENTATION

The main goal of the experimentation is to measure the energy consumption of the radio for using different MAC modules we have implemented. Meanwhile we also measure the latency and throughput of S-MAC in different modes.

To facilitate the measurement of multiple messages traveling through a multi-hop network, we add a message queue at the application layer to buffer the outgoing message on each node.

A. Measurement of Energy Consumption

To measure the energy consumption on the radio, we measure the amount of time that the radio on each node has spent in different modes: sleep, idle, receiving or transmitting. The energy consumption in each mode is then calculated by multiplying the time with the required power to operate the radio in that mode. We measure energy indirectly in this way because of the difficulty in directly observing current draw on physically small, low power motes. We found the power consumption from the data sheet of the radio transceiver, which is 13.5mW, 24.75mW and 15μW, in receiving, transmitting and sleep respectively. There is no difference between listening and receiving in this radio transceiver model. We measure the energy consumption of each node when utilizing different MAC protocols and under different traffic loads.

A.1 Tests on a Two-Hop Network

Figure 8 is the first topology we used in our experiments. This is a two-hop network with two sources and two sinks. Packets from source A flow through node C and end at sink D, while those from B also pass through C but end at E.

We change the traffic load by varying the inter-arrival period of the messages. If the message inter-arrival period is 5 seconds, a message is generated every 5 seconds by each source node. In this experiment, the message inter-arrival period varies from 1s to 10s. For the highest rate with a 1s inter-arrival time, the wireless channel is nearly fully utilized due to its low bandwidth. For each traffic pattern, we have done 10 independent tests when using different MAC protocols.

In each test, each source periodically generates 10 messages, which in turn is fragmented into 10 small data packets (40 bytes each) supported by the TinyOS. Thus in each experiment, there are 200 TinyOS data packets to be passed from their sources to

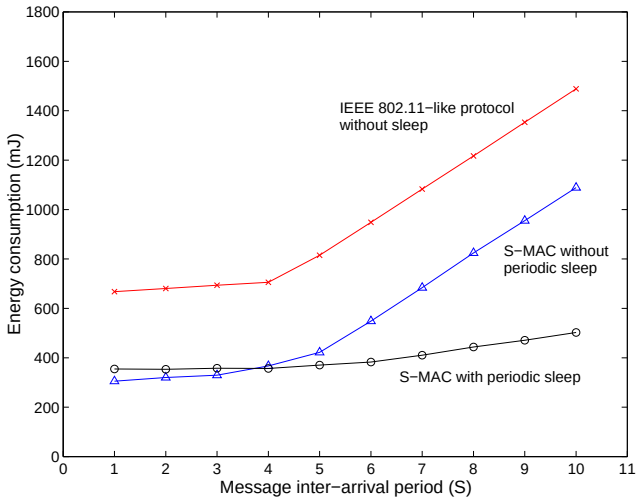

Fig. 7. Mean energy consumption on radios in each source node.

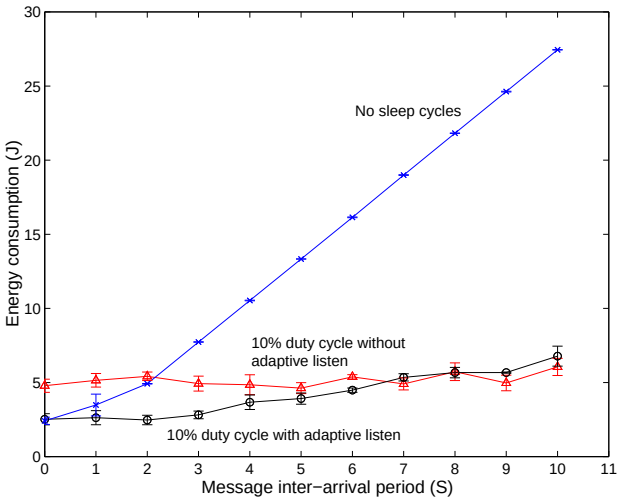

Fig. 9. Aggregate energy consumption on radios in the entire 10-hop network using three S-MAC modes.

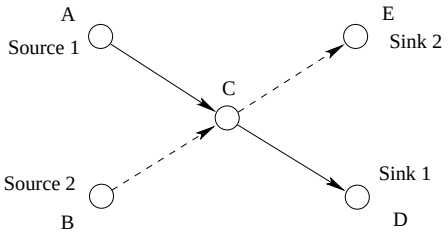

Fig. 8. Topology 1: two-hop network with two sources and two sinks.

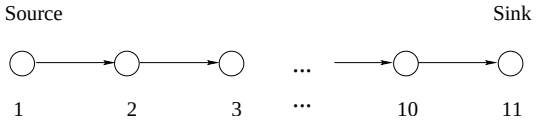

Fig. 10. Topology 2: ten-hop linear network with one source and one sink.

their sinks. We measure the energy consumption of the radio on each node to pass the fixed amount of data. The actual time to finish the transmission is different for each MAC module.

In the 802.11-like MAC, the fragments of a message are sent in a burst, *i.e.*, RTS/CTS is not used for each fragment. We did not measure the 802.11-like MAC without fragmentation, which treats each fragment as an independent packet and uses RTS/CTS for each of them, since it is obvious that this MAC consumes much more energy than the one with fragmentation. In S-MAC message passing is used, and fragments of a message are always transmitted in a burst. In the S-MAC module with periodic sleep, each node is configured to operate in 50% duty cycle.

Figure 7 shows the measured average energy consumption from the source nodes A and B. The traffic is heavy when the message inter-arrival time is less than 4s. In this case, 802.11 MAC uses more than twice the energy used by S-MAC. Since idle listening rarely happens, energy savings from periodic sleeping is very limited. S-MAC achieves energy savings mainly by avoiding overhearing and efficiently transmitting a long message.

When the message inter-arrival period is larger than 4s, traffic load becomes light. In this case, the complete S-MAC protocol has the best energy property, and far outperforms 802.11 MAC. Message passing with overhearing avoidance also performs better than 802.11 MAC. However, as shown in the figure, when idle listening dominates the total energy consumption, the periodic sleep plays a key role for energy savings. The energy consumption of S-MAC is relatively independent of the traffic pattern.

Compared with 802.11, message passing with overhearing avoidance saves almost the same amount of energy under all traffic conditions. This result is due to overhearing avoidance among neighboring nodes A, B and C. The number of packets to be sent by each of them are the same in all traffic conditions.

A.2 Tests on a Multi-Hop Network

In the multi-hop experiments, we set up a line topology with 11 nodes, as shown in Figure 10. The nodes are configured to send in the minimum transmission power, and are put in a 1-meter space. The first node is the source, and last node is the sink.

As before, we vary the traffic load by changing the packet inter-arrival time on the source node. This time the packet inter-arrival time changes from 0s to 10s, where 0s means all the packets are generated and queued at the same time on the source node. Under each traffic condition, the test is independently carried out for 5 times. In each test, the source node sends 20 messages that are 100-byte long each. There is no fragmentation on all messages.

We have compared three different operation modes of S-MAC. The first one is 10% duty cycle without adaptive listen. The second one is 10% duty cycle with adaptive listen. The last one is fully active mode, where periodic sleep is completely disabled. Since the periodic listen interval is 115ms, 10% duty cycle corresponds to a schedule period of 1.15s.

Figure 9 shows the measured energy consumption on radios

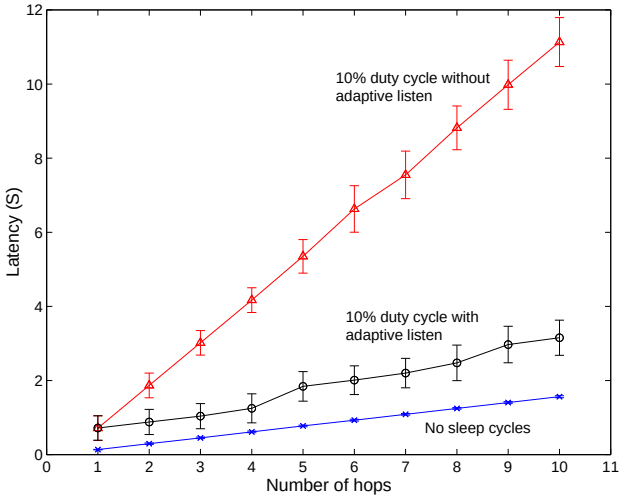

Fig. 11. Mean message latency on each hop under the lowest traffic load.

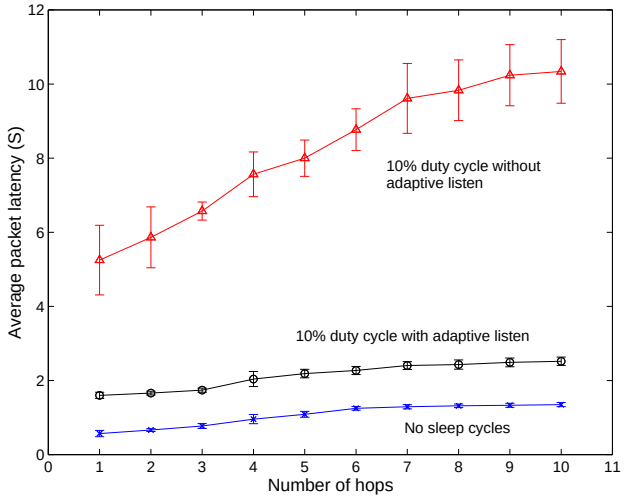

Fig. 12. Mean message latency on each hop under the highest traffic load.

in the entire network to pass the fixed amount of data from the source to the sink. The result conforms with that we have obtained on the two-hop network. S-MAC with periodic sleep achieves substantial energy savings over the MAC without periodic sleep in the multi-hop network, especially when traffic load is light.

Comparing the two MAC modules that both running at the 10% duty cycle, we can see that the one with adaptive listen achieves better energy efficiency than the one without adaptive listen, especially when traffic load is heavy. The main reason is that the adaptive listen largely reduces the overall time needed to pass the fixed amount of data through the network.

B. Measurement of End-to-End Latency

Since S-MAC makes the trade-off of latency for energy savings, we expect that it can have longer latency in a multi-hop network due to the periodic sleep on each node. Adaptive listen (Section IV-C) is designed to minimize such additional latency. To quantify latency and measure the benefits of adaptive listen, we use the same ten-hop network topology in Figure 10 to measure the end-to-end latency of S-MAC.

We consider two extreme traffic conditions, the lowest traffic load and highest traffic load. Under the lowest traffic load, the second message is generated on the source node after the first one is received by the sink. To do this, a coordinating node is placed near the sink. When it hears that the sink receives the message, it signals the source directly by sending at the highest power. In this traffic load, there is no queuing delay on each node. Compared with the MAC without sleep, the extra delay is only caused by the periodic sleep on each node. Under the highest traffic load, all messages are generated and queued on the source node at the same time. So there is a maximum queuing delay on each node including the source node. In both cases, we begin measuring the latency of each message from the time it is generated on the source node.

In each test, the source node generates 20 messages, each of 100 bytes. There is no fragmentation on these messages. For the lowest traffic load, the packet generation time is uniformly distributed within one schedule period. Under both the lowest and the highest traffic load, the measurement is repeated for 10 times. The measurement is on the same S-MAC modes as we used in measuring the energy consumption in the same ten-hop network.

Figure 11 shows the measured mean message latency on each hop in the lowest traffic load. In all three S-MAC modes, the latency increases linearly with the number of hops. However, S-MAC at 10% duty cycle without adaptive listen has much higher latency on each hop than the other two. This is because each message has to wait for one sleep cycle on each hop.

The latency of S-MAC with adaptive listen, by comparison, is very close to that of the MAC without any periodic sleep, because adaptive listening often allows S-MAC to immediately send a message to the next hop. However, it does not reach the shortest latency in the MAC of fully active mode. As described in Section IV-C, adaptive listen cannot guarantee the immediate transmission at each hop. If a node sends an RTS but fails to get a CTS from the intended receiver, it has to wait for its next cycle. Figure 11 shows that S-MAC with adaptive listen has about twice the average latency than the MAC in fully active mode (except the first 1 or 2 hops). We also observe that for either low-duty-cycle mode, the variance in latency is much larger than that in the fully active mode, and it increases with the number of hops. The large variance is due to the fact that some messages may miss sleep cycles of certain nodes.

Figure 12 shows the mean message latency on each hop in the highest traffic load. Again, the low-duty-cycle mode without adaptive listen has the highest latency. With adaptive listen, the latency is close to that in fully active mode, which is still about twice on average.

The large difference at the first hop between the two low-duty-cycle modes (with and without adaptive listen) is due to the queuing delay on the source node. Without adaptive listen and transmission, one message is sent in each cycle, so the last message has to wait for at least 19 cycles. As messages go further,

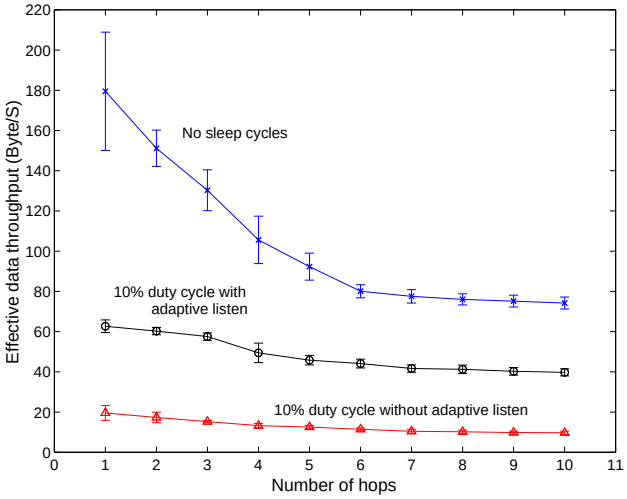

Fig. 13. Throughput at each hop under the highest traffic load.

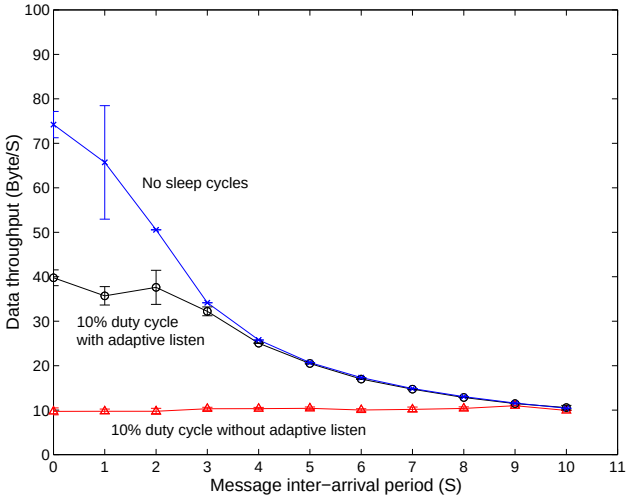

Fig. 14. Throughput over 10 hops under different traffic load.

later hops have less queuing delay. The overall result is that the low-duty-cycle mode without adaptive listen has a lower slope than that in Figure 11.

The low-duty-cycle mode with adaptive listen tracks the slope of the fully active mode, because it is always able to send data in such a heavy traffic load. This effect also reduces the variance in latency.

C. Measurement of End-to-End Throughput

Just as S-MAC may increase latency, it may also reduce the throughput. Therefore we next evaluate throughput in the same ten-hop network.

We first consider throughput for the highest traffic load, which is the same as that when measuring the latency in the highest traffic load. It delivers the maximum possible number of bytes of data in a unit time. The results do not count any control packets. Only data packets received at each hop are counted for the throughput.

Figure 13 shows the throughput measured at each hop in the highest traffic load. As expected, periodic sleeping reduces throughput. Compared with fully active mode, the low-duty-cycle modes with adaptive listen and without adaptive only achieve about $1/2$ and $1/8$ of the throughput at 10 hops. Throughput is lower because latency is higher (Figure 12), since sometimes sending is delayed. Similar to the reduced latency by adaptive listen compared with S-MAC without adaptive listen, it significantly improves the end-to-end throughput.

The results also show that, for all MAC variations, throughput drops as the number of hops increases, due to the RTS/CTS contention in the multi-hop network.

We next look at the end-to-end throughput in different traffic load. Figure 14 shows the measured throughput from the source to the sink for different message inter-arrival time on the source node. It is from the same data to measure the energy consumption in Section VII-A.2.

The results show that both the throughput of fully active mode and that of the adaptive listen mode reduce as traffic load decreases. When traffic load is very low, they all approach to that of the non-adaptive mode, because the three MAC modes spend about the same time to finish transmitting the same number of messages. Nothing happens during the long time between two messages. In this case, it is worthless to spend more energy trying to increase throughput. Since there is not enough traffic, the throughput cannot be increased.

D. Energy vs. Latency and Throughput

Now we look at the trade-offs that S-MAC has made on energy, latency and throughput from the above measurement results to understand if S-MAC succeeds in reducing overall cost to send a fixed amount of data. On one hand, we know that S-MAC reduces energy consumption, but this savings may be offset by decreased throughput.

To evaluate the combined effect of energy consumption and reduced throughput, we calculate the per-byte cost of energy and time to pass data from the source to the sink under different traffic load. The results are shown in Figure 15, which are obtained by combining data from Figure 9 and Figure 14.

We can see that when traffic load is very heavy (inter-arrival time less than 1.5s), adaptive listening and the no-sleep modes both show statistically equivalent performance that is significantly better than sleeping without adaptive listen. In this case, both adaptive listen and no-sleep are almost always active, while the added delay of non-adaptive sleep requires extra transmission time and lowers overall energy efficiency.

At lower traffic load, the energy cost without sleeping quickly exceeds the cost of sleep modes (at inter-arrival time longer than 4s). We believe that energy cost of no-sleep grows linearly in the limit, as shown also in Figures 7 and 9.

Adaptive and non-adaptive sleeping become statistically equivalent at lower traffic load (inter-arrival time at or above 9s). This result indicates that the overhead for adaptive listening is minimal. The benefits of adaptive listen occur at moderate to high traffic load.

In summary, periodic sleeping provides excellent energy per-

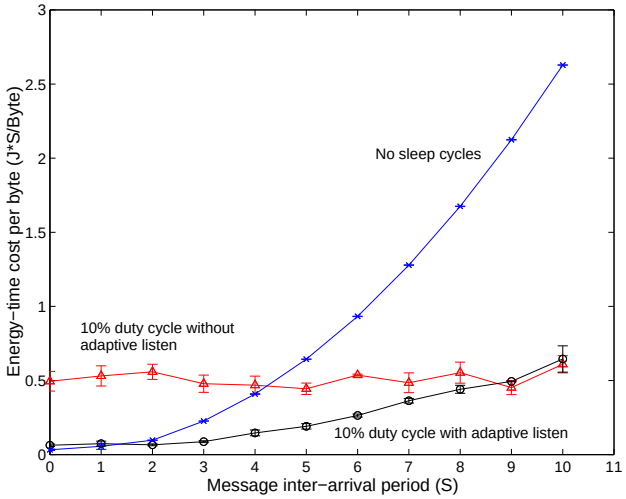

Fig. 15. Energy-time cost per byte on passing data from source to sink under different traffic load.

formance at light traffic load, but adaptive listening is able to adjust to traffic and provide energy performance as good as no-sleep at heavy load. It makes S-MAC with adaptive listening ideal for sensor networks where traffic is intermittent.

VIII. Conclusions

This paper presents S-MAC, a medium access control protocol specifically designed for wireless sensor networks. Energy efficiency is the primary goal in the protocol design. Low-duty-cycle operation of each node is achieved by periodic sleeping. Together with overhearing avoidance and message passing, S-MAC obtains significant energy savings compared with 802.11-like protocols without sleeping. It is able to greatly prolong the network lifetime, which is critical for real world sensor network applications.

Periodic sleeping increases latency and reduces throughput. However, adaptive listening largely reduces such cost for energy savings. It enables each node to adaptively switch mode according to the traffic in the network.

S-MAC has been implemented on the Mote hardware, and the source code is freely available to the research community. Experimental results have verified our design principles.

Acknowledgments

The authors would like to acknowledge the support and discussions from members of the SCADDS project, the Center for Embedded Networked Sensing, the TinyOS group at UC Berkeley and researchers at Intel Labs. Specifically, we would like to thank the following people: Ramesh Govindan, David Culler, Mark Yarvis, Mani Srivastava, Curt Schurgers for their in-depth discussions and feedback; Padmaparna Haldar for implementing S-MAC in *ns*-2 and providing detailed feedback and bug fixes; Athanasios Stathopoulos and Jerry Zhao for testing S-MAC on Mica motes; Jason Hill and Mohammad Rahimi for TinyOS and Mica hardware support.

References

[1] LAN MAN Standards Committee of the IEEE Computer Society, *Wireless LAN medium access control (MAC) and physical layer (PHY) specification*, IEEE, New York, NY, USA, IEEE Std 802.11-1997 edition, 1997.

[2] Mark Stemm and Randy H Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices," *IEICE Transactions on Communications*, vol. E80-B, no. 8, pp. 1125–1131, Aug. 1997.

[3] Oliver Kasten, *Energy Consumption*, `http://www.inf.ethz.ch/~kasten/research/bathtub/energy_consumption.html`, Eldgenossische Technische Hochschule Zurich.

[4] Gregory J. Pottie and William J. Kaiser, "Embedding the internet: wireless integrated network sensors," *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, May 2000.

[5] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Boston, MA, USA, Aug. 2000, pp. 56–67, ACM.

[6] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan, "Building efficient wireless sensor networks with low-level naming," in *Proceedings of the Symposium on Operating Systems Principles*, Lake Louise, Banff, Canada, Oct. 2001, pp. 146–159.

[7] `http://www.cs.berkeley.edu/~awoo/smartdust/`.

[8] Crossbow Technology Inc., `http://www.xbow.com/`.

[9] RF Monolithics Inc., `http://www.rfm.com/`, *ASH Transceiver TR1000 Data Sheet*.

[10] RF Monolithics Inc., `http://www.rfm.com/`, *ASH Transceiver TR3000 Data Sheet*.

[11] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister, "System architecture directions for networked sensors," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, USA, Nov. 2000, pp. 93–104, ACM.

[12] S. Singh and C.S. Raghavendra, "PAMAS: Power aware multi-access protocol with signalling for ad hoc networks," *ACM Computer Communication Review*, vol. 28, no. 3, pp. 5–26, July 1998.

[13] Wei Ye, John Heidemann, and Deborah Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the IEEE Infocom*, New York, NY, June 2002, pp. 1567–1576.

[14] Frazer Bennett, David Clarke, Joseph B. Evans, Andy Hopper, Alan Jones, and David Leask, "Piconet: Embedded mobile networking," *IEEE Personal Communications Magazine*, vol. 4, no. 5, pp. 8–15, Oct. 1997.

[15] Katayoun Sohrabi and Gregory J. Pottie, "Performance of a novel self-organization protocol for wireless ad hoc sensor networks," in *Proceedings of the IEEE 50th Vehicular Technology Conference*, 1999, pp. 1222–1226.

[16] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan, "Energy-efficient communication protocols for wireless microsensor networks," in *Proceedings of the Hawaii International Conference on Systems Sciences*, Jan. 2000.

[17] Alec Woo and David Culler, "A transmission control scheme for media access in sensor networks," in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Rome, Italy, July 2001, pp. 221–235, ACM.

[18] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A media access protocol for wireless lans," in *Proceedings of the ACM SIGCOMM Conference*, London, UK, Sept. 1994, pp. 212–225.

[19] Jaap C. Haartsen, "The Bluetooth radio system," *IEEE Personal Communications Magazine*, pp. 28–36, Feb. 2000.

[20] Bluetooth SIG Inc., "Specification of the Bluetooth system: Core," `http://www.bluetooth.org/`, 2001.

[21] Yu-Chee Tseng, Chih-Shun Hsu, and Ten-Yueng Hsieh, "Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks," in *Proceedings of the IEEE Infocom*, New York, NY, June 2002, pp. 200–209.

[22] Shugong Xu and Tarek Saadawi, "Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks?," *IEEE Communications Magazine*, pp. 130–137, June 2001.

[23] T. S. Rappaport, *Wireless Communications, Principles and Practice*, Prentice Hall, 1996.

[24] `http://webs.cs.berkeley.edu/tos`.

[25] Atmel Corporation, `http://www.atmel.com/`, *AVR Microcontroller AT90LS8535 Reference Manual*.

[26] Wei Ye, John Heidemann, and Deborah Estrin, "A flexible and reliable radio communication stack on motes," Tech. Rep. ISI-TR-565, USC Information Sciences Institute, Sept. 2002.