An Energy-Efficient MAC Protocol 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 almost every way: 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 Mote, developed at University of California, Berkeley. The experiment 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.**

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 a network normally consists of a large number of distributed nodes that organize themselves into a multi-hop 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].

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 the scalability to the change 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. The network topology changes over time as well due to many reasons. A good MAC protocol should easily accommodate such network changes. Other important attributes include fairness, latency, throughput and bandwidth utilization. These attributes are generally the primary concerns in traditional wireless voice and data networks, but in sensor networks they are secondary.

This paper presents sensor-MAC (S-MAC), a new MAC protocol explicitly designed for wireless sensor networks. While reducing energy consumption is the primary goal in our design, our protocol also has good scalability and collision avoidance capability. It achieves 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, and less useful data packets can be transmitted. 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 or CDMA nodes must listen to the channel to receive possible traffic. Many 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 2 Mbps Wireless LAN module (IEEE 802.11/2Mbps) specification shows idle:receive:send ratios is 1:2:2.5 [3].

S-MAC tries to reduce the waste of energy from all the above sources. In exchange we accept some reduction in both per-hop fairness and latency. Although per-hop fairness and latency are reduced, we will argue that the reduction does not necessarily result in lower *end-to-end* fairness and latency.

In traditional wireless voice or data networks, each user desires equal opportunity and time to access the medium, *i.e.*,

W. Ye (weiye@isi.edu) and J. Heidemann (johnh@isi.edu) are with the Information Science Institute (ISI), University of Southern California (USC). D. Estrin (destrin@cs.ucla.edu) is with the Computer Science Department, University of California, Los Angeles and USC/ISI.

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. Normally there is only one application. At certain time, a 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 a very long message. 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. This is unfair from a per-hop, MAC level perspective, 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.

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.

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.

To demonstrate the effectiveness and measure the performance of our MAC protocol, we have implemented it on our testbed wireless sensor nodes, *Motes*, developed by University of California, Berkeley [7]. The mote has a 8-bit Atmel AT90LS8535 microcontroller running at 4 MHz. It has a low power radio transceiver module TR1000 from RF Monolithics, Inc [8], which operates at 916.5 MHz frequency and provides a transmission rate of 19.2 Kbps. The mote runs on a very small event-driven operating system called TinyOS [9]. In order to compare the performance of our protocol with some other protocols, we also implemented a simplified IEEE 802.11 MAC on this platform.

The contributions of this work are therefore:

- 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 [10], 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 sensor-net specific hardware.

II. Related Work

The medium access control is a broad research area, and many researchers have done research work in the new area of low power and wireless sensor networks [11], [12], [13], [14].

Current MAC design for wireless sensor networks can be broadly divided into contention-based and TDMA protocols. 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 [15]. 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. PAMAS [10] made an improvement by trying to avoid the overhearings among neighboring nodes. Our paper also exploits similar method for energy savings. 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 address the issue of reduce idle listening.

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 [16], [17] and LEACH [13]. 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 [12] 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.

Piconet [11] is an architecture designed for low-power ad hoc wireless networks. One interesting feature of piconet is that it also puts nodes into periodic sleep for energy conservation. The scheme that piconet uses to synchronize neighboring nodes is to let a node broadcast its address before it starts listening. If a node wants to talk to a neighboring node, it must wait until it receives the neighbor's broadcast.

Woo and Culler [14] 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 trade it off for further energy savings.

III. Sensor-MAC Protocol Design

The main goal in our MAC protocol design is to reduce energy consumption, while supporting good scalability and collision avoidance. Our protocol tries 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. To achieve the design goal, we have developed the S-MAC that consists of three major components: periodic listen and sleep, collision and overhearing avoidance, and message passing. Before describing them we first discuss our assumptions about the wireless sensor network and it applications.

A. Network and Application Assumptions

Since sensor networks are somewhat different than traditional IP networks or ad hoc networks of laptop computers, we next summarize our assumptions about sensor networks and applications.

We expect sensor networks to be composed of many small nodes deployed in an ad hoc fashion. 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 communication (instead of long-range communication) to conserve energy [4]. Most communication 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.

We expect most sensor networks to be dedicated to a single application or a few collaborative applications, thus rather than node-level fairness (like in the Internet), we focus on maximizing system-wide application performance.

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.

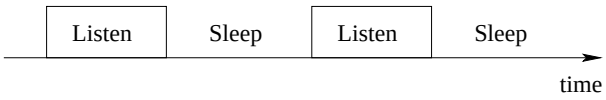

Fig. 1. Periodic listen and sleep.

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. (One application-level approach to manage latency is to deploy a slightly larger sensor network and have edge nodes raise the network to heightened awareness when something is detected.)

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

B. 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%. So we can achieve close to 50% energy savings.

B.1 Basic Scheme

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.

The duration of time for listening and sleeping can be selected according to different application scenarios. For simplicity these values are the same for all the nodes.

Our scheme requires periodic synchronization among neighboring nodes to remedy their clock drift. We use two techniques to make it robust to synchronization 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, our scheme requires much looser synchronization among neighboring nodes. All nodes are free to choose their own listen/sleep schedules. However, to reduce control overhead, we prefer neighboring nodes to syn-

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

chronize 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 broadcasting it to all its immediate neighbors. This ensures that all neighboring nodes can talk to each other even if they have different schedules. For example, in Figure 2 if node A wants to talk to node B, it just wait until B is listening. If multiple neighbors want to talk to a node, they need to contend for the medium when the node is listening. The contention mechanism is the same as that in IEEE 802.11, *i.e.*, using RTS (Request To Send) and CTS (Clear To Send) packets. The node who first sends out the RTS packet wins the medium, and the receiver will reply with a CTS packet. After they start data transmission, they do not follow their sleep schedules until they finish transmission.

Another 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 quite easy to adapt 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. So the latency requirement of the application places a fundamental limit on the sleep time.

B.2 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. The node first listens for a certain amount of time. If it does not hear a schedule from another node, it randomly chooses a time to go to sleep and immediately broadcasts its schedule in a SYNC message, indicating that it will go to sleep after t seconds. We call such a node a *synchronizer*, since it chooses its schedule independently and other nodes will synchronize with it.
2. If the node receives a schedule from a neighbor before choosing its own schedule, it follows that schedule by setting its schedule to be the same. We call such a node a *follower*. It then waits for a random delay t_d and rebroadcasts this schedule, indicating that it will sleep in $t - t_d$ seconds. The random delay is for collision avoidance, so that multiple followers triggered from the same synchronizer do not systematically collide when rebroadcasting the schedule.
3. If a node receives a different schedule after it selects and broadcasts its own schedule, it adopts both schedules (i.e., it schedules itself to wake up at the times of both is neighbor and itself). It broadcasts it own schedule before going to sleep.

We expect that nodes only rarely adopt multiple schedules, since every node tries to follow existing schedules before choosing an independent one. On the other hand, it is possible that some neighboring nodes fail to discover each other at beginning due to collisions when broadcasting schedules. They may still find each other later in their subsequent periodic listening.

To illustrate this algorithm, consider a network where all nodes can hear each other. The timer of one node will fire first and its broadcast will synchronize all of its peers on its schedule. If instead two nodes independently assign schedules (either because they cannot hear each other, or because they happen to transmit at nearly the same time), those nodes on the border between the two schedules will adopt both. In this way, a node only needs to send once for a broadcast packet. 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 period listen and sleep as other nodes.

B.3 Maintaining Synchronization

The listen/sleep scheme requires synchronization among neighboring nodes. Although the long listen time can tolerate fairly large clock drift, neighboring nodes still need to periodically update each other their schedules to prevent long-time clock drift. The updating period can be quite long. The measurements on our testbed nodes show that it can be on the order of tens of seconds.

Updating schedules 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 finishes transmitting the SYNC packet, which is approximately when receivers get the packet (since propagation delays are short). Receivers will adjust their timers immediately after they receive the SYNC packet. A node will go to sleep when the timer fires.

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.

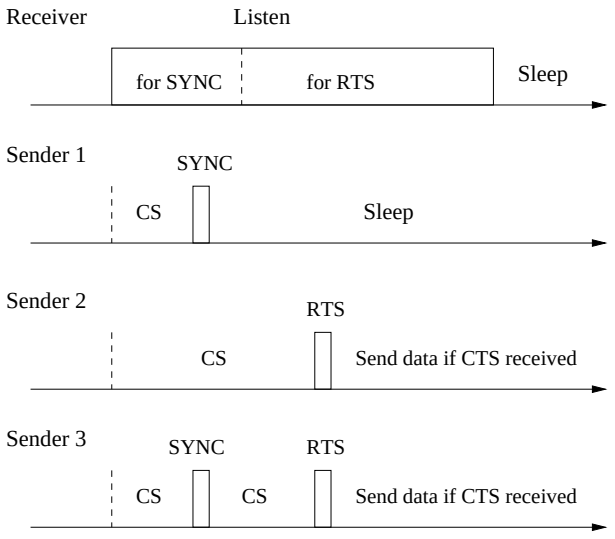

Fig. 3. Timing relationship between a receiver and different senders. CS stands for carrier sense.

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

Each node periodically broadcasts SYNC packets to its neighbors even if it has no followers. This allows new nodes to join an existing neighborhood. The new node follows the same procedure in the above subsection to choose its schedule. The initial listen period should be long enough so that it is able to learn and follow an existing schedule before choosing an independent one.

C. Collision and Overhearing Avoidance

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. So it needs to be avoided.

C.1 Collision Avoidance

Since multiple senders may want to send to a receiver at the same time, 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 [15].

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.

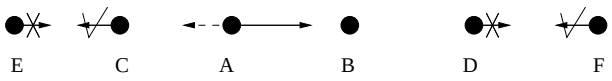

Fig. 4. Who should sleep when node A is transmitting to B?

Physical carrier sense is performed at the physical layer by listening to the channel for possible transmissions. The procedure was described in section III-B.3. The randomized carrier sense time 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.

C.2 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 a lot of 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 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 going on.

As shown in Figure 4, 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.

D. 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 [8]. 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 the fragmentation support. We should point out the difference between that scheme with our message passing.

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.

The reason for 802.11 to do so is 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 need 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.

E. Energy Savings vs. Increased Latency

This subsection analyzes the trade-offs between the energy savings and the increased latency due to nodes sleep schedules. We compare our protocol with protocols that do not have periodic sleep such as the IEEE 802.11,

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 failed, 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 nodes periodic sleeping. When a sender gets a packet to transmit, it must wait until the receiver wakes

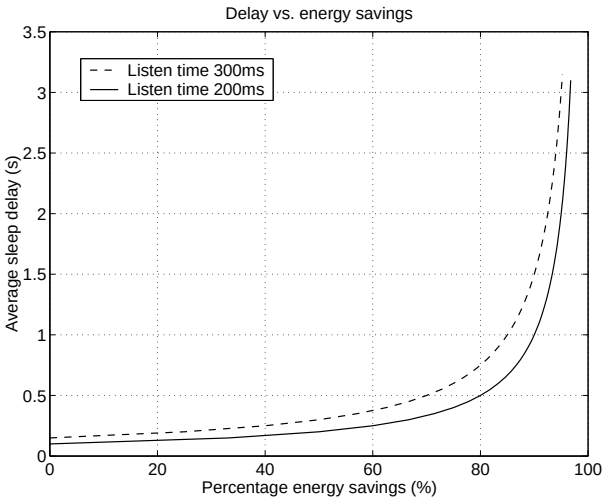

Fig. 5. Energy savings vs. average sleep delay for the listen time of 30ms.

up. We call it *sleep delay* since it is caused by the sleep of the receiver.

We call a complete cycle of the listen and sleep a *frame*. Assume a packet arrives at the sender with equal probability in time within a frame. So the average sleep delay on the sender is

$$D_s = T_{frame}/2 \tag{1}$$

where

$$T_{frame} = T_{listen} + T_{sleep} \tag{2}$$

Comparing with protocols without periodic sleep, the relative energy savings in S-MAC is

$$E_s = \frac{T_{sleep}}{T_{frame}} = 1 - \frac{T_{listen}}{T_{frame}} \tag{3}$$

The last item in the above equation is the duty cycle of the node. It is desirable to have the listen time as short as possible so that for a certain duty cycle, the average sleep delay is short. In our implementation we set the listen time as 300ms. Figure 5 shows the percentage of energy savings E_s vs. average sleep delay D_s on each node for the listen time of 300ms and 200ms. We can see that even if the sleep time is zero (no sleeping) there is still a delay. This effect is because contention only starts at the beginning of each listen interval.

IV. Protocol Implementation

The purpose of our implementation is to demonstrate the effectiveness of our protocol and to compare our protocol with 802.11 through some basic experiments.

A. Testbed

We use Rene Motes, developed at UCB [7], as our development platform and testbed (see Figure 6). A mote is slightly larger than a quarter. The heart of the node is the Atmel AT90LS8535 microcontroller [18], which has 8K bytes of programmable flash and 512 bytes of data memory.

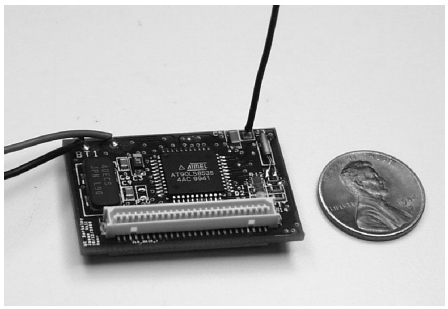

Fig. 6. The UCB Rene Mote.

The radio transceiver on the mote is the model TR1000 from RF Monolithics, Inc [8]. When using the OOK(on-off keyed) modulation, it provides a transmission rate of 19.2 Kbps. It has three working modes, *i.e.*, receiving, transmitting and sleep, each drawing the input current of 4.5mA, 12mA (peak) and 5μA respectively.

Our motes use TinyOS, an efficient event-driven operating system [9], [19]. It provides the basic mechanism for packet transmitting, receiving and processing. TinyOS promotes modularity, data sharing and reuse.

As of July 2001, the standard release of TinyOS has only one type of packet, which consists of a header, the payload and a cyclic redundancy check (CRC). The length of the header or the payload can be changed to different values. However, once they are defined, all packets have the same length and format. In our MAC implementation, the header, payload and CRC fields have 6B, 30B and 2B respectively.

Normally the control packets, such as RTS, CTS and ACK, are very short and without payload. So we have created an other packet type in TinyOS, the control packet, which only has the 6-byte header and the 2-byte CRC. We have modified several TinyOS components to accommodate the new packet. This enables us to efficiently implement MAC protocols and accurately measure their performance.

B. Implementation of MAC Protocols

We have implemented three MAC modules on the mote and TinyOS platform, as listed below.

1. Simplified IEEE 802.11 DCF
2. Message passing with overhearing avoidance
3. The complete S-MAC

For the purpose of performance comparison, we first implemented a simplified version of IEEE 802.11 DCF. It has the following major pieces: physical and virtual carrier sense, backoff and retry, RTS/CTS/DATA/ACK packet exchange, and fragmentation support.

The duration of each carrier sense is a random time within the contention window. The randomization is very important to avoid collisions at the first step. For simplicity, the contention window does not exponentially increase when backoff happens. The fragmentation support follows the same procedure as in

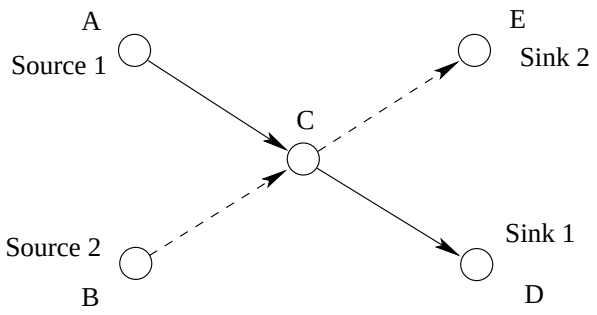

Fig. 7. Topology used in experiments: two-hop network with two sources and two sinks.

IEEE 802.11 standard [1] and is described in Section 3 of this paper.

With 802.11 the radio of each node does not go into sleep mode. It is either in listen/receiving mode or transmitting mode.

The second module is the message passing with overhearing avoidance. It achieves energy savings by avoiding overhearing, reducing control overhead and contention times. It does not include the period listen and sleep. So there is no additional delay comparing with the simplified IEEE 802.11. The radio of each node goes into the sleep mode only when its neighbors are in transmission.

With the message passing module we have incorporated periodic listen and sleep, and completed most basic functionalities in S-MAC. Currently, the listen time for each node is 300ms, and sleep time can be changed to different values, such as 300ms, 500ms, 1s, *etc.*, which makes different duty cycles of the radio. We can also specify the frequency that the SYNC packet is sent for schedule update between neighboring nodes. In our following experiments, we have chosen the sleep time as 1 second and the frequency for schedule update is 10 listen/sleep period, *i.e.*, 13 seconds.

It should be noted that the energy savings in the current implementation is only due to the sleep of the radio. In other words, the microcontroller does not go to sleep. It actually has a sleep mode, which consumes much less energy and can be waked up by a low-frequency watchdog timer. If we put the microcontroller into the sleep mode as well when the radio is sleeping, we are able to save more energy.

V. Experimentation

The main goal of the experimentation described here is to measure the energy consumption of the radio for using each of the MAC modules we have implemented.

A. Experiment Setup

Figure 7 is the 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. The topology is simple, but it is sufficient to show the basic characteristics of the MAC protocols.

We will look at the energy consumption of each node when utilizing different MAC protocols and under different traffic loads.

The two sources periodically generate a sensing message, which is divided into some fragments. In the simplified IEEE 802.11 MAC, these fragments are sent in a

burst, *i.e.*, RTS/CTS is not used for each fragment. We did not measure the 802.11 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 our protocol, message passing is used, and fragments of a message are always transmitted in a burst.

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 our following experiments, the message inter-arrival period varies from 1s to 10s.

For each traffic pattern, we have done 10 independent tests to measure the energy consumption of each node when using different MAC protocols. In each test, each source periodically generates 10 messages, which in turn is fragmented into 10 small data packets supported by the TinyOS. Thus in each experiment, there are 200 TinyOS data packets to be passed from their sources to their sinks. For the highest rate with a 1s inter-arrival time, the wireless channel is nearly fully utilized due to its low bandwidth.

We measure the amount of time that each node has used to pass these packets as well as the percentage time its radio has spent in each mode (transmitting, receiving, listening or sleep). The energy consumption in each node is then calculated by multiplying the time with the required power to operate the radio in that mode. 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.

B. Results and Analysis

The experiments are carried out on the three MAC modules we have implemented on our testbed nodes. In the result graphs, the simplified IEEE 802.11 DCF is denoted as 'IEEE 802.11'. The message passing with overhearing avoidance is identified as 'Overhearing avoidance'. The complete S-MAC protocol, which includes all pieces of our new protocol, is denoted as 'S-MAC'.

We first look at the experiment results on the source nodes A and B. Figure 8 is the measured average energy consumption from these two nodes. 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

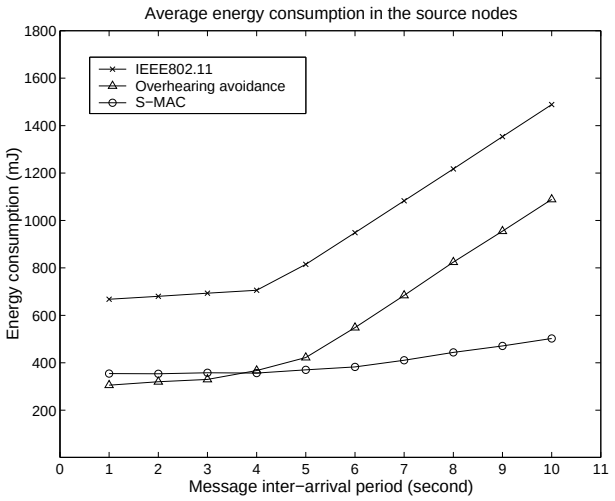

Fig. 8. Measured energy consumption in the source nodes.

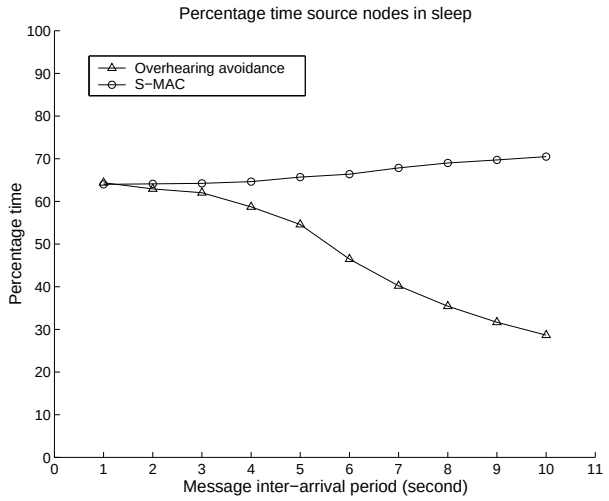

Fig. 9. Measured percentage of time that the source nodes in the sleep mode.

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.

Figure 9 shows the percentage of time that the source nodes are in the sleep mode. It is interesting that the S-MAC protocol adjusts the sleep time according to traffic patterns. When there is little traffic, the node has more sleep time (although there is a limit by the duty cycle of the node). When traffic increases, nodes have fewer chances to go to periodic sleep and thus spend more time in transmission.

This is a useful feature for sensor network applications, since the traffic load indeed changes over time. When there is no sensing event, the traffic is very light. When some nodes detects an event, it may trigger a big sensor like a camera, which will generate heavy traffic. The S-MAC protocol is able to adapt to the traffic changes. In comparison, the module of message passing with overhearing avoidance does not have periodic sleep, and nodes spend more and more time in idle listening when traffic load decreases.

Figure 10 shows the measured energy consumption in the intermediate node C. We can see in the light traffic case, it still outperforms 802.11 MAC. In heave traffic case, it consumes slightly more energy than 802.11. One reason is that S-MAC has synchronization overhead of sending and receiving SYNC packets. Another reason is that S-MAC introduces more latency and actually uses more time to pass the same amount of data.

In fact, if the traffic is extremely heavy and a node does not have any chance to follow its sleep schedule, the scheme of periodic listen and sleep does not benefit at all. However, message passing and overhearing avoidance are still effective means of saving energy. This has been illustrated in the results of the

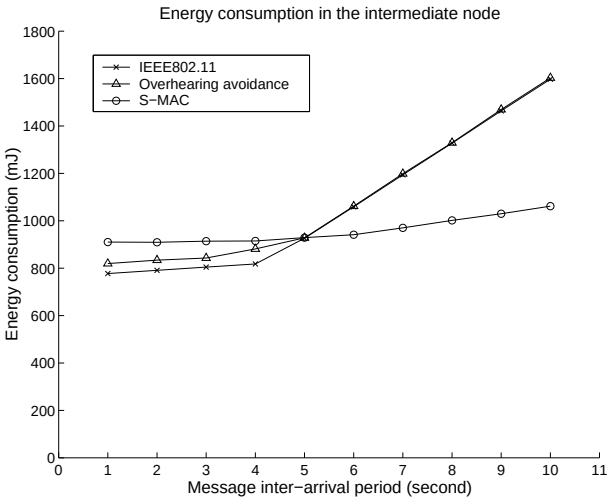

Fig. 10. Measured energy consumption in the intermediate node.

source nodes (Figure 8). But we cannot see similar results on the intermediate node C, since all packet transmissions involve this node. In this case, its energy consumption is about the same as that of using the 802.11 MAC.

VI. Conclusions and Future Work

This paper presents a new MAC protocol for wireless sensor networks. It has very good energy conserving properties comparing with IEEE 802.11. Another interesting property of the protocol is that it has the ability to make trade-offs between energy and latency according to traffic conditions. The protocol has been implemented on our testbed nodes, which shows its effectiveness.

Future work includes system scaling studies and parameter analysis. More tests will be done on larger testbeds with different number of nodes and system complexity.

ACKNOWLEDGMENTS

This work is supported by NSF under grant ANI-9979457 as the SCOWR project (http://robotics.usc.edu/projects/scowr/), and by DARPA under grant DABT63-99-1-0011 as the SCADDS project (http://www.isi.edu/scadds/) and under contract N66001-00-C-8066 as the SAMAN project (http://www.isi.edu/saman/) via the Space and Naval Warfare Systems Center San Diego.

The authors would like to acknowledge the discussions and suggestions from members of the SCOWR, SCADDS and SAMAN projects.

We would also like to thank the TinyOS group (http://tinyos.millennium.berkeley.edu/) at UCB for their support with TinyOS and Motes.

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.

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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, ACM.

[15] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "Macaw: A media access protocol for wireless lans," in *Proceedings of the ACM SIGCOMM Conference*, 1994.

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

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

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

[19] http://tinyos.millennium.berkeley.edu/.