Energy and Latency Control in Low Duty Cycle MAC Protocols

Yuan Li Wei Ye John Heidemann
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA, USA 90292
{liyuan, weiye, johnh}@isi.edu
Tel: 310-822-1511

Abstract

Recently, several MAC protocols such as S-MAC and T-MAC have exploited scheduled sleep/wakeup cycles to conserve energy in sensor networks. Until now, most protocols have assumed all nodes in the network were configured to follow the same schedule, or they assumed border nodes would follow multiple schedules, but did not evaluate those cases. This paper develops two new algorithms to control and exploit the presence of multiple schedules to reduce energy consumption and latency. The first one is the *global schedule algorithm* (GSA). Through experiments, we demonstrate that, because of radio propagation vagaries, large sensor networks have very ragged, overlapping borders where many nodes listen to two or more schedules. GSA is a fully distributed algorithm that allows a large network to converge on a single global schedule to conserve energy. Secondly, we demonstrate that strict schedules incur a latency penalty in a multi-hop network when packets must wait for the next schedule for transmission. To reduce latency in multi-hop paths we develop the *fast path algorithm* (FPA). FPA provides fast data forwarding paths by adding additional wake-up periods on the nodes along paths from sources to sinks. We evaluate both algorithms through experiments on Berkeley motes and demonstrate that the protocols accomplish their goals of reducing energy consumption and latency in large sensor networks.

Keywords: Medium access control, Sensor network, Wireless network, Energy efficiency

Energy and Latency Control in Low Duty Cycle MAC Protocols

Yuan Li Wei Ye John Heidemann
Information Sciences Institute, University of Southern California
USC/ISI Technical Report ISI-TR-595, August 2004

***Abstract*— Recently, several MAC protocols such as S-MAC and T-MAC have exploited scheduled sleep/wakeup cycles to conserve energy in sensor networks. Until now, most protocols have assumed all nodes in the network were configured to follow the same schedule, or they assumed border nodes would follow multiple schedules, but did not evaluate those cases. This paper develops two new algorithms to control and exploit the presence of multiple schedules to reduce energy consumption and latency. The first one is the *global schedule algorithm* (GSA). Through experiments, we demonstrate that, because of radio propagation vagaries, large sensor networks have very ragged, overlapping borders where many nodes listen to two or more schedules. GSA is a fully distributed algorithm that allows a large network to converge on a single global schedule to conserve energy. Secondly, we demonstrate that strict schedules incur a latency penalty in a multi-hop network when packets must wait for the next schedule for transmission. To reduce latency in multi-hop paths we develop the *fast path algorithm* (FPA). FPA provides fast data forwarding paths by adding additional wake-up periods on the nodes along paths from sources to sinks. We evaluate both algorithms through experiments on Berkeley motes and demonstrate that the protocols accomplish their goals of reducing energy consumption and latency in large sensor networks.**

I. Introduction

Wireless sensor networks use many small, wireless sensors to sense their environment. Wireless sensors are often battery operated to simplify deployment. With many nodes placed in their target environment, changing batteries becomes difficult or impossible, thus sensor nodes must be energy efficient.

The radio consumes the largest share of the power budget in many sensor nodes, so one way to minimize energy consumption is to keep the radio off as much as possible. Listening to an idle radio channel wastes energy, yet many sensor networks have long stretches of inactivity between event detections. Recently network protocols such as S-MAC [10] [11], T-MAC [8], and Zigbee [7] have adopted a loosely synchronized sleep/wakeup cycle to allow nodes to operate at low duty cycles while maintaining network-level connectivity.

Sleep/wakeup MAC protocols establish and maintain a schedule about when nodes listen for possible transmissions and when they sleep. When a new node joins the network, it listens for and tries to adopt an existing schedule. Sleep durations and energy savings are maximized when all nodes are on the same schedule.

Multiple schedules can occur in large networks even though the protocols are biased to promote a single schedule. Sometimes when a node starts it will fail to hear an existing schedule and so it will create a new schedule for itself. Moreover, in a large network we must expect that there are nodes that cannot hear each other directly or at certain times. Such nodes naturally generate different schedules because they must choose independently. Finally, if nodes are moving, then they can easily move between different parts of the network with different schedules.

Sleep/wakeup protocols must be prepared to detect and track neighbors operating on different schedules to maintain global network connectivity. In S-MAC, for example, nodes periodically listen for an entire synchronization period to detect any new neighbors that are on different schedules. We call nodes that share the same schedule a *virtual cluster*, and nodes with neighbors in two clusters *border nodes*. Ideally we expect a network to have one or only a few virtual clusters and few border nodes, since border nodes spend more time listening or sending data and therefore consume more energy. Over time, these border nodes will exhaust their batteries more rapidly than other nodes, possibly causing network partition.

Unfortunately, in Section III-A we demonstrate that multiple schedules are quite common in practice, even when we expect nodes to hear each other. We propose the *global schedule algorithm* (GSA) that allows all nodes to converge on a single global schedule (Section II-A).

A second weakness of a schedule-based MAC is that sleep/wakeup schedules can increase latency in a multi-hop network. When a packet arrives at a node it must be queued if the next-hop is sleeping. Potentially there can be many such *schedule misses* in a long multi-hop path. Two optimizations have been proposed to reduce this cost. *Adaptive listen* [11] in S-MAC uses the RTS-CTS mechanism to wake up neighbors of the sender and receiver after the packet is delivered, avoiding a schedule miss and halving the latency. T-MAC proposes a *future request-to-send* [8] scheme to let a node on the third hop know there is a message for it by sending a future-request-to-send (FRTS) packet, further reducing the sleep delay.

Although these optimizations reduce latency, the sleep/wakeup cycle still incurs additional delay compared to an always-on MAC protocol. This paper proposes a new al-

gorithm called *fast path algorithm* (FPA), which is designed to wake up nodes along a path at exactly the right times to avoid schedule misses. Fast paths are necessarily biased towards a given direction or destination. Key challenges with fast paths are handling their interactions with a default schedule and a system to set up and tear down fast paths as they are required.

The contribution of this paper is to explore the limits of scheduled MAC protocols. We develop algorithms for global schedule and fast paths to allow nodes to minimize energy consumption and control latency. We have implemented both on Mica-2 Mote hardware. We investigate each experimentally, showing that multiple schedules are not rare (as we previously expected) but quite common in real networks. Our global schedule algorithm is therefore important to eliminate the energy overhead of border nodes. We also demonstrate how fast paths eliminate the latency penalty due to the scheduled sleep/wakeup, and how they interact with each other and the standard schedule. Fast paths can be very important when low-latency transmission is required, as in monitoring and triggering applications. Although we implement those two algorithms in S-MAC, the general approaches are applicable to other MAC protocols with a sleep/wakeup schedule, such as T-MAC and Zigbee.

II. Design of Algorithms

This section describes the design of the new algorithms we propose: global schedule convergence and fast path forwarding. Although these algorithms can apply to all sleep/wakeup-based MACs, we describe the details in our implementation (based on S-MAC) when relevant.

A. Global Schedule Algorithm

As described above, nodes on the borders of different virtual clusters consume more energy than others due to multiple schedules. Since multiple schedules cannot be prevented in a large distributed system we next describe the *global schedule algorithm* (GSA), which allows all nodes to converge on a common schedule. Although it might seem simple to adopt a global schedule, in a decentralized system we must avoid oscillating between multiple schedules.

To converge on a single schedule it requires three components: a way to uniquely identify each schedule, a way to propagate new schedules to other nodes, and a way to discover new schedules that appear due to node movement or merging of network partitions. We assume the third component is handled by the basic sleep/wakeup MAC. For example, S-MAC requires each node to perform periodic neighbor discovery. We next describe how our GSA solves the first two problems.

A simple way to identify each schedule is to use the ID of the node that originates the schedule. The major problem with this approach is that a node may start a new schedule with the same ID when it reboots. An improvement is to assigned a random identifier each time it reboots. However, the random identifier is generated locally on a node, and is not guaranteed to be unique in the network. Our solution is to use the combination of the schedule originator's ID and the age of the schedule. The tuple uniquely identifies the schedule even if the originating node leaves the network or reboots.

The schedule age indicates how long the schedule has existed in the network. When a node originates a new schedule, it records the time when the schedule is generated. When it later advertises the schedule, it puts the schedule age into the packet. When a node receives a schedule update, it updates schedule age as advertised in the packet and record the current time as latest update time. It then compares it to the schedule it follows. If it is a different schedule and is older than its own schedule, the node will switch to the new schedule. In the case that two different schedules have the same age, the lower schedule ID breaks the tie. When a node switches to a new schedule, it will update its neighbors, so that the new schedule will propagate through its virtual cluster. Every time a node sends a schedule update, it increases the schedule age by adding the time since its last update (either by itself or by its neighbors). Over time, all nodes migrate toward the oldest schedule in the network as the global schedule.

It is worth to note that it is better to identify a schedule by its age than its birth time. Using birth time requires global time synchronization, and our GSA does not make such an assumption. It is also important to retain the ability to support multiple schedules, since there are many transient cases while the network converges to a single schedule. Our approach ensures that data transmissions are not interrupted when nodes switch schedules.

B. Fast Path Algorithm

Sleep/wakeup MAC protocols trade off latency for energy saving. When a packet travels over multiple hops it can be delayed when its next-hop node is sleeping. Although adaptive listen [11] and future-request-to-send [8] can reduce these schedule misses, they affect only the next hop or next two hops.

Our *fast path algorithm* is a new mechanism to explicitly manage schedules in a multi-hop path to avoid schedule misses. Given a source, sink, and the path between them, we add additional wakeup periods called *fast path schedules* along the path, scheduled such that they occur exactly when the previous-hop node will be ready to send the packet. For example, in Fig. 1, when nodes strictly follow their sleep/wakeup cycles, if data is transfered from node 1 to node 2 in regular listen time $t1$, node 2 must then wait until time $t3$ before sending it to node 3. By enabling fast

path schedules (shown as dotted boxes), node 3 knows to wake up at time $t2$, avoiding this delay.

Providing fast path schedules in a schedule MAC raises several concerns. First, one must determine the path and establish fast path schedules along it. We also must consider how fast paths interact with standard schedules, adaptive listen or future-request-to-send, and if multiple concurrent fast paths are supported. We review each of these issues below.

In a basic network we expect all nodes to follow the global schedule. A fast path may be desired when a flow is established from a source to a sink, or in some sensor networks there may be a single well-known sink. If a new path or distribution tree is created between a source and sink (or source and sinks), this routing information can trigger establishment of a fast path. Alternatively a single sink can establish a fast path from all nodes to itself. When a sink need to establish a fast path, it generates a *fast path request* which is passed along to the source node. The request is piggybacked on the resource discovery message. (In directed diffusion [3], this would be the interest message, or in AODV or DSR it would be the route request message.)

On receiving the request, the source node starts to establish the fast path. As show in Fig. 1, source node can communicate with the next node during their regular listen time, so there is no need to set up a fast path schedule on both of them. Starting from node 2, nodes need to inform next hop nodes where to add the extra wakeup period. Let us assume t_{cs} represents the maximum possible carrier sense delay at a node, and t_{tx} represents the time needed for transmission of a data packet with fixed length. d is set to be $t_{cs}+t_{tx}$. Fig. 1 shows node 3 needs to place the fast path at least d away from regular listen time. This is the earliest time for node 3 to wake up after transmission between previous hop. Similarly the position P for node 4 to place its fast path schedule is $2d$ from regular listen time. P is sent with the first data packet in the flow. This fast-path setup over an n hop path requires at most $O(n/2)$ frame durations assuming adaptive listen is in use. From Fig. 1, we can infer that the time needed to transfer each following data packet in a n hop network over a fast path is approximately $n \times d$, about the same as if the network was always awake.

Once a fast path is established, no special activity is required to send data over it. Ideally a regular or fast path period is available after the packet is received at each hop. If the packet is delayed due to interference or corruption, it may miss its slot, in the worst case loosing a frame.

An important question is how fast paths interact with the global schedule and other optimizations such as adaptive listen. When the fast path schedule overlaps with the regular listen time or they are so close that data transmission using

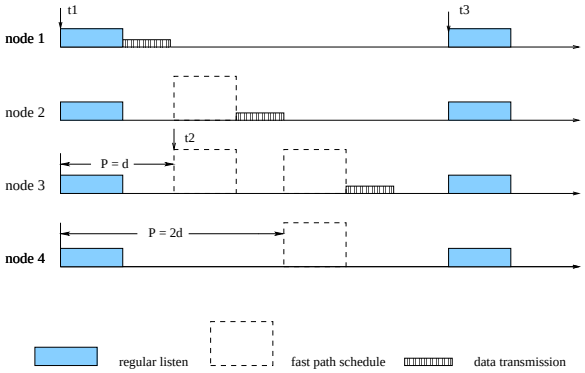

Fig. 1. Fast data forwarding when adding fast path schedules

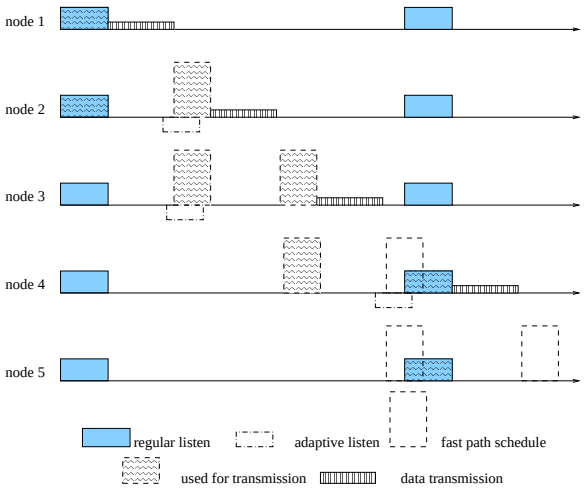

Fig. 2. Regular listen, adaptive listen and fast path schedule overlap

fast path schedule can not finish before regular listen time, we use the regular listen time as the fast path schedule instead. Fig. 2 shows this case, where node 4's fast path schedule overlaps with its regular wakeup period, so data is transferred during regular listen time instead.

Similarly, if an opportunity to do adaptive listen overlaps with a fast-path slot, we skip the adaptive listen. For example, in Fig. 2, after data is transmitted to node 2, we use the fast path rather than adaptive listen to send to node 3. We prefer fast paths to adaptive listen because it should reach all the way to the sink rather than just the next hop.

There is no explicit mechanism to remove fast path schedules. Instead, each node monitors fast path use. If a fast path slot goes unused for a long duration (in our case 5 frame durations), we time it out and discard it. Any use of a fast path schedule refreshes it. When the routing path changes, another fast path is established along with propagation of the new route, and the old fast path will expire.

So far, we have discussed how to apply fast path schedule

TABLE I

EXPERIMENTS SHOWING MULTIPLE SCHEDULES IN A LARGE NETWORK

Test 1:	Node ID	-	-	-	15	-	-	-	-	-	21	-	29	-	5	20	14	19	-	-	27	23	32	-	10	-	-	17	3	22	26	12	16	24	13	2	9	25	30	7	11	31	4	8	-	6	28	1	-	-	18
	Schedule 1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	o	-	-	o	-	-	-	-	o	o	-	o	o	o	o	o	X	X	o	o	-	X	X	o*	-	-	X
	Schedule 5	-	-	-	X	-	-	-	-	-	X	-	X	-	X*	X	X	X	-	-	X	X	X	-	X	-	-	X	X	X	X	X	X	X	X	X	X	X	X	X	o	-	X	X	-	-	o	X	-	-	-
Test 2:	Node ID	36	1	-	-	-	-	-	-	-	25	31	34	6	9	23	19	24	4	-	-	27	2	32	15	-	14	20	7	26	30	17	21	28	18	5	13	29	35	11	16	37	8	12	-	10	33	3	-	-	22
	Schedule 1	-	X*	-	-	-	-	-	-	-	-	X	-	X	X	-	X	X	-	-	-	-	-	-	X	-	X	-	-	-	-	-	-	-	-	-	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Schedule 3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	o	-	-	-	-	X	X	X	-	X	X	X	X	-	X	X	X	X	X	X	X	-	X	X	X*	-	-	X
	Schedule 15	X	-	-	-	-	-	-	-	-	X	-	X	-	o	X	o	o	X	-	-	X	X	X	o*	-	o	X	o	o	o	X	o	o	o	o	X	o	o	o	o	o	o	o	-	o	o	o	-	-	o
Test 3:	Node ID	40	1	28	19	22	-	7	41	31	25	33	36	-	8	24	20	23	4	-	37	29	2	34	14	-	13	18	5	26	30	16	21	27	17	-	12	32	38	10	15	39	6	11	-	9	35	3	-	-	-
	Schedule 1	-	X*	X	X	X	-	X	-	-	o	X	-	-	-	-	X	-	X	-	-	-	X	X	o	-	X	X	-	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Schedule 3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	-	X	o	X	X	X	-	o	X	X	X	X	X	X	X	-	X	X	X*	-	-	-
	Schedule 8	X	-	o	o	o	-	-	X	X	X	o	X	-	o*	X	o	X	-	-	X	X	-	o	X	-	-	o	o	X	-	o	o	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Schedule 14	o	o	o	o	o	-	o	o	o	o	o	o	-	X	o	o	o	o	-	o	o	o	o	o*	-	o	o	o	o	o	X	o	o	o	-	X	o	o	o	o	o	o	o	-	o	o	o	-	-	-
Test 4:	Node ID	33	-	24	-	22	-	-	34	-	-	-	-	-	8	21	17	20	3	-	31	25	1	29	13	-	10	18	5	23	27	14	16	26	15	4	11	28	32	6	12	-	-	9	-	7	30	2	-	-	19
	Schedule 1	-	-	X	-	X	-	-	-	-	-	-	-	-	-	-	-	-	X	-	-	X	X*	X	-	-	X	-	-	-	-	-	-	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Schedule 2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	o	X	-	X	X	o	X	X	X	X	-	-	X	-	X	X	X*	-	-	X
	Schedule 8	X	-	o	-	o	-	-	X	-	-	-	-	-	X*	X	X	X	-	-	X	o	-	o	-	-	o	o	o	o	o	o	o	X	o	-	-	o	o	-	-	-	-	-	-	-	o	-	-	-	o
	Schedule 13	o	-	o	-	o	-	-	o	-	-	-	-	-	-	o	o	o	o	-	o	o	-	o	X*	-	o	o	o	o	o	X	o	o	o	o	X	o	o	o	o	-	-	o	-	o	-	o	-	-	o

50 nodes are placed in a line topology. Node IDs in the top row indicate the order in which they turn on ("-" indicates nodes that failed to activate). Each line indicates a schedule and which nodes know about that schedule, with "*" showing the node initiating the schedule, "X" indicating the primary schedule for that node, "o" indicating seconder schedules.

to nodes on a single path. In a large sensor network, many fast paths may exist at the same time. We assume each node can support a small number of active paths. Each fast path schedule is maintained independently on each node. If a shared node is at different hops away from different sinks, separate fast path schedules are inserted at different positions and they will not interfere. On the other hand, if fast path slots from different paths overlap in time at a given node then there will be contention. In principle this slot could be lengthened; in practice we expect this event to be rare enough that the slight increase in delay due can be tolerated.

III. EXPERIMENTAL RESULTS

We next test on moderate-size sensor networks, showing that many schedules can coexist in a large network and that the use of fast path schedules can reduce latency.

A. Multiple Schedules

The goal of this experiment is to characterize schedule creation and the composition of virtual clusters in a large network. We use S-MAC without global scheduling in an outdoor network of 50 sensor nodes, capturing the effects of real protocol operation and radio propagation.

We implement a simplified version of global schedule algorithm as described in sectionII-A to trace schedule changes. Schedules are identified by the initializing nodes IDs. For example, schedule 1 is initialized by node 1.

a) Methodology: We set up a linear topology with 50 Mica2Dot motes, each running TinyOS [2] and S-MAC [11] at 10% duty cycle. Nodes are placed about 90cm apart, making a line about 45m long. The transmission power of each node is set to the lowest level (RF output power, -20dBm; PA_POW, 01). In separate experiments we expected 100% packet reception at ranges of about 2m.

In each test, we randomly activate each mote at the start and collect sleep schedule information in the network. We repeat the test for 4 times. Because schedule adoption is based on which nodes hear each other, the order in which nodes are activated is important. For example, we verified through a separate outdoor experiment that when nodes are activated from one end to the other in sequence, there is only one schedule eventually formed in the network. Because the newly activated nodes are physically close to the existing cluster, they always hear and follow the common schedule.

We adopt the following procedure to approximate a large deployment where nodes are turned on at different times while avoiding the case of physically sequential activation (the best case), or nearly concurrent activation. Initially, each node is turned on manually and enters a pre-experiment state where it listens continuously without using S-MAC. We then broadcast a message from a central node at high power to trigger all nodes to begin. Each node then picks a random time uniformly distributed between zero and one frame duration and then begins the normal S-MAC initialization procedure. (It listens for other schedules for one synchronization period and if it hears none, it selects its own schedule.)

Unfortunately, we found unpredictable radio propagation meant our trigger packet was not always received by the nodes and in most experiments, 18–36% failed to particpate.

TABLE II
NUMBER OF SCHEDULES KNOWN BY NODES

	number of schedules				
	1	**2**	**3**	**4**	**mean** (std. dev.)
Test 1	56%	44%	-	-	1.4 (0.50)
Test 2	32%	68%	0%	-	1.7 (0.47)
Test 3	0%	66%	34%	0%	2.3 (0.48)
Test 4	9%	44%	47%	0%	2.4 (0.58)

b) Results: Table I shows the sleep schedule information we collect from the four tests performed. All tests showed multiple schedules, with up to four different schedules being independently created. For example, in Test 4, there are four different schedules formed in the network of 34 nodes. Nodes 1, 2, 8, 13 initiate these schedules and all other nodes adopt one of these schedules.

A second observation is that many nodes know multiple schedules. Table II shows the percentage of nodes knowing different number of schedules in four tests respectively. In Test 4 only 10% of the total nodes know one schedule, while the others know two or three schedules. On average, a node knows 1.4–2.4 different sleep schedules in our experiment. Maintaining multiple schedules costs additional energy for border nodes: they must listen during the wakeup period of a schedule, and broadcast packets are sent on each schedule separately. (Since in S-MAC, border nodes listen only to the contention period of their primary schedule unless they need to send to a node on a different schedule.)

Why do so many nodes know multiple schedules? We expected that schedule boundaries would be relatively distinct, with most nodes following some schedule and only a few nodes on the border knowing multiple schedules. For example, in test 1, all nodes on the left belong to schedule 5 while some nodes on the right are in schedule 1, and a few nodes in the middle know both schedules.

Instead, we observe that cluster boundaries are quite complicated. This occurs for several reasons.

First, the node that initiates a schedule often actually leaves the schedule it creates. This result is due to an optimization in S-MAC called *singleton schedule elimination*. After a node announces its schedule, it considers that schedule temporary until it knows that someone else has adopted it. While the schedule is temporary the initiating node will switch to a new schedule it hears. This optimization is a good fit for a network where one node starts among an existing network, but it is a bad match for our experiment where many nodes select their schedules at about the same time. An example of this behavior can be seen by node 1 in test 1. It successfully initializes schedule 1, which is followed by some nodes to establish a virtual cluster, but before hearing any other node from this virtual cluster, node 1 hears schedule 5 and switches its schedule in attempt to eliminate singleton virtual cluster.

In Test 2, for example, node 2 presumably announced its own schedule but then adopts node 15's schedule. Since no other nodes adopt node 2's schedule, this elimination optimization reduces the total number of schedules. While it is helpful in this case, we would prefer to simplify schedule selection with our global schedule algorithm rather than with special cases like singleton elimination.

Finally, the main reason so many nodes discover multiple schedules is because of the "gray region" in radio propagation [12], [9]. While short-range radio reception is quite good to some distance (about 3m with our configuration), packets are occasionally received at about twice that distance. Thus, even if two distant nodes are forced to choose independent schedules, the overlap between these schedules will necessarily be large.

These experimental results motivate our global scheduling algorithm. In separate laboratory experiments and ACM Sensys 2003 [4], we have demonstrated that global schedule assignment will successfully allow all nodes to converge to a single schedule quickly.

B. Fast Path Schedules

In this section we evaluate our fast-path mechanism to show how it can reduce latency.

a) Methodology: In this experiment, we set up a linear topology of 10 Mica-2 Motes, each 2m apart. All nodes are configured to send at the lowest transmission power as previous experiments. In this case, nominal radio rage is about 3m (radio propagation differs between Mica-2s and Mica2Dots). The first node is the data source, and the node at the other end is the sink. In each test, the source sends 10 unicast messages, each 100B long. Each message is sent 25s after the previous, so there is never more than one in the network at a time. We measure the arrival time of the message at each node and repeat the experiment 5 times each with and without fast paths enabled.

All nodes run S-MAC, this time at 5% duty cycle. S-MAC uses adaptive listening and global schedule assignment, and in half of the experiments, a fast path. With global schedule assignment, all nodes always have one basic schedule, and in half of the experiments they also have a fast path. Since we have a static topology, when we enable the fast path algorithm we pre-configure a fast path schedule based on physical node position. (For these experiments we have not integrated the fast-path algorithm with a routing algorithm.) Our expectation is that we will see lower latency when fast-paths are used.

b) Results: In the 100 packets transmitted in all the experiments, two packets are lost and not successfully retransmitted after 7 retries. We ignore these packets when computing delays. In addition, six packets were lost and then successfully retransmitted by the MAC layer. The effect of these packets is described below.

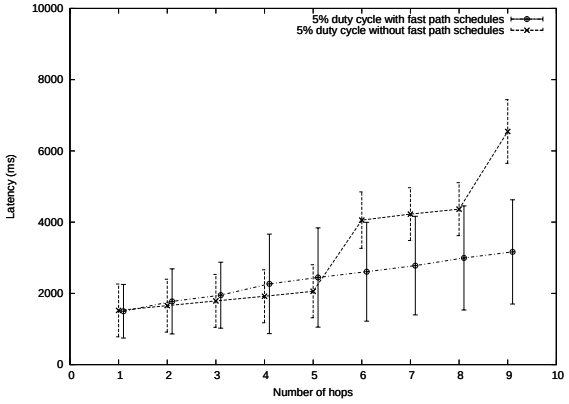

Fig. 3. Mean message latency on each hop of all successful transmissions

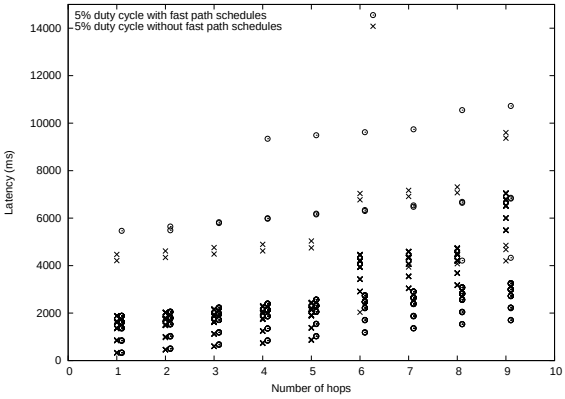

Fig. 5. Message latency on each hop of all successful transmissions

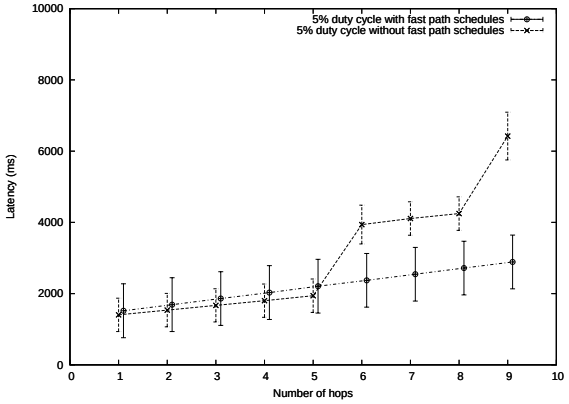

Fig. 4. Mean message latency on each hop of all successful transmissions with no retransmissions

Fig. 3 shows the measured mean latency on each hop of all the successfully transmitted packets. First, we observe that the latency with fast-pathing is very predictable. Both fast-path and standard S-MAC have a slight delay on the first hop, but fast-path nodes then have a short, consistent latency for each additional hop. This confirms our analysis that with fast-path, the per-hop delay is approximately the contention and transmit time $d = t_{cs} + t_{tx}$ as described in Section II-A. By contrast, standard S-MAC suffers a large delay when a schedule is missed, in this case between hop 5 and 6, hop 8 and 9.

With basic S-MAC we would expect to lose one frame duration each hop. However, in our experiment, no schedule miss occurs until hop 6. This result is due to the effects of the adaptive listen algorithm. In a network where each node can hear only its adjacent neighbors we would expect this algorithm to avoid a schedule miss every other hop, since only nodes that hear either the RTS or CTS know to wake for adaptive listen. Experimentally we observe *multiple* adaptive listen events, allowing the packet to travel five hops before suffering a schedule miss. This result is because of radio propagation distance. In our experiment, nodes for hops 1 through 5 were able to hear *all* packets exchanged on the first five hops, so we had three successive adaptive listen events. While this result occurred consistently in this experiment, we cannot expect such good fortune in all cases.

The general trend in Fig. 3 shows that fast-path packets travel consistently faster than packets without fast pathing. However, for hops 6–8 the error bar (standard deviation) of the fast-path case overlaps the mean of non-fast-path case. To understand why we see such large variance in our results, Fig. 5 shows a scatter plot of the successful transmissions in all the experiments. This plot shows individual events rather than means, and we observe that while most packets are sent successfully, some have large latencies at certain hops. This is due to packet retransmissions.

Because we must predict packet transmission time when planning fast paths, we cannot anticipate retransmission events. When a packet is retransmitted it will "fall off" the fast path and suffer a schedule miss. However, Fig. 5 shows that once the packet is retransmitted it will get on the next available fast path.

Fig. 4 shows the mean per-hop latencies of all packets *except* those that are retransmitted. The trend is the same as Fig. 3, except that the variances for the delays are reduced. For these packets, fast pathing is clearly faster than without.

We observe that S-MAC shows a fairly large latency in the first hop, both with and without fast-path support. This delay occurs because messages originate at random times, independent of a S-MAC schedule. The sender needs to wait till the next wakeup period to send the message. This initial delay could be avoided by synchronizing data generation with the S-MAC schedule, Alternatively, a possible future option would generate a fast-path schedule that is synchronized with data generation times.

IV. Related Work

T-MAC [8] is another contention-based low duty cycle MAC protocol. It demonstrates similar behaviors to S-MAC with adaptive listen. The authors discuss the *early sleeping problem* and propose several new techniques, including future-RTS (FRTS). Their algorithms reduce the latency incurred by a scheduled MAC, but approaches such as FRTS are limited to the 3-hop neighborhood of the originator.

There is another type of low duty cycle MAC protocols using the preamble sampling techniques to levitate the energy cost during idle listening. Receivers periodically wake up for a very short duration and sample the medium for activities. With knowledge of each neighbor's independent sampling schedule information WiseMAC [1] can further reduce the wakeup preamble and energy cost. WiseMAC saves energy from eliminating synchronization for different schedules, but since nodes are not coordinated, a sleep delay is introduced at each hop and could be as large as the duration of a sampling period. B-MAC [6] is similar work using this technique of preamble sampling. The main contribution of B-MAC is it provides an interface for reconfiguring MAC layer parameters to meet the application's new and dynamically changing demand.

The work most similar to our fast path algorithm is DMAC, independent work by Lu et al. [5]. DMAC is designed specifically for data gathering tree applications, where the multiple sources and a single sink in the network construct a data forwarding tree. In their paper, they mention *forwarding interruption problem*, where not all nodes on a multi-hop path to the sink are aware of the data delivery. They propose a similar strategy, to stagger the listen period on a node according to its depth in the data gathering tree. There are two important differences between their work and ours. First, they focus on the single-sink problem, while we propose fast paths as a general solution for an arbitrary number of paths. Second, their work is limited to analysis and simulation. To our knowledge, our paper is the first to provide an experimental evaluation of scheduled MAC protocols.

V. Conclusion

In this paper, we presented two new algorithms to reduce energy consumption and latency in low duty cycle MAC protocols. Global schedule algorithm allows all nodes to converge on a single global schedule to conserve energy, while fast paths allocate additional slots to avoid schedule misses and reduce latency. We demonstrate through testbed experiments multiple schedules are common in large network, motivating the global schedule algorithm. We also discover that the boundaries of virtual clusters are quite diffuse, many nodes become border nodes without the global schedule algorithm. We evaluate latency savings of our fast path algorithm on a 9-hop network with an implementation based on S-MAC.

VI. Acknowledgment

This work is in part supported by NSF under grant ANI-0220026 as the MACSS project and a grant from the Intel Corporation.

Source code to the algorithms is available on the author's web site.

References

[1] Amre El-Hoiydi and Jean-Dominique Decotignie. Wisemac: An ultra low power mac protocol for multi-hop wireless sensor networks. In *Algorithmic Aspects of Wireless Sensor Networks: First International Workshop, ALGOSENSORS 2004, Turku, Finland, July 16, 2004. Proceedings*, Turku, Finland, July 2004. Springer.

[2] 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*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.

[3] 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*, pages 56–67, Boston, MA, USA, August 2000. ACM.

[4] Yuan Li, Wei Ye, and John Heidemann. Demonstration of schedule and latency control in S-MAC. Technical Report ISI-TR-581, USC/Information Sciences Institute, November 2003.

[5] Gang Lu, Bhaskar Krishnamachari, and Cauligi Raghavendra. An adaptive energy-efficient and low-latency mac for data gathering in sensor networks. In *Workshop on Energy-Efficient Wireless Communications and Networks (EWCN '04), held in conjunction with the IEEE International Performance Computing and Communications Conference (IPCCC)*, April 2004.

[6] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems*, Baltimore,Maryland, USA, November 2004.

[7] IEEE Computer Society. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE, New York, NY, USA, 2003.

[8] Tijs van Dam and Koen Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, pages 171–180, Los Angeles, California, USA, November 2003.

[9] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, pages 14–27, Los Angeles, California, USA, November 2003. ACM.

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

[11] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated, adaptive sleeping for wireless sensor networks. *ACM/IEEE Transactions on Networking*, 12(3):493–506, June 2004. A preprint of this paper was available as ISI-TR-2003-567.

[12] Jerry Zhao and Ramesh Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, pages 1–13, Los Angeles, California, USA, November 2003.