Energy and Latency Control in Low Duty Cycle MAC Protocols

Yuan Li Wei Ye John Heidemann
{liyuan, weiye, johnh}@isi.edu
Information Sciences Institute, University of Southern California

***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 have assumed border nodes would follow multiple schedules, but have not evaluated 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, and many sensor networks have long stretches of inactivity between event detections. Recently network protocols such as S-MAC [12] [13], T-MAC [10], and Zigbee [9] 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 schedules 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 might fail to hear an existing schedule and 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. Nodes that share the same schedule form a *virtual cluster*, and nodes with neighbors in two clusters are *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* [13] 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* [10] 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 to some extent, the sleep/wakeup cycle still incurs additional delay compared to an always-on MAC protocol. This paper proposes a new algorithm called *fast path algorithm* (FPA), which is designed to wake up nodes along a path at exactly the right time 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 developing a system to set up and tear down fast paths when needed.

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 we need to address three problems: First, how to uniquely identify each schedule; second, how to propagate new schedules to other nodes, and third, how to discover new schedules that appear due to node movement or merging of network partitions. We assume the third problem is handled by the basic sleep/wakeup MAC. For example, S-MAC requires each node to perform periodical neighbor discovery. We next describe how our GSA solves the first two problems.

One way to identify schedules is to use the ID of the node that originates the schedule. The major problem with this approach is that nodes may start different schedules with the same ID each time they reboot. An improvement is to assign nodes random identifiers when they reboot. However, random identifiers are generated locally on nodes, and are not guaranteed to be unique in the network.

In our algorithm, we introduce *schedule age* indicating how long a schedule has existed in the network. Schedules with different sleep/wakeup time must have entered the network at different moments and thus have different ages. It is possible that schedules start independently at different moments but have the same wakeup time. Such schedules are naturally the same schedules but with different ages. Nodes need to update their schedule ages whenever they receive schedule updates from nodes on the same schedule but with elder ages.

When GSA is enabled, once nodes hear a different schedule older than their own, they will switch to the older one and broadcast the change immediately. Over time, all nodes migrate toward the oldest schedule in the network as the global schedule. 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.

Note that some care must be made to clearly identify unique schedules via schedule age. At a high level, we simply adopt the oldest schedule. To define schedule age, a node records the local time of each newly generated schedule, T_{gen}. Local time is defined as time since that node has booted. When a node advertises its schedule, it announces the schedule *age*, $\Delta_{age} = now - T_{gen}$. We advertise age rather than creation time because age is independent of any global clock since it is a relative measurement (this approach assumes propagation time is low). When another node adopts a schedule, it accepts the schedules age and computes the schedule creation time in its *local* clock, $T_{gen\text{-}local} = now - \Delta_{age}$. It then can compute the schedule age using its own clock and $T_{gen\text{-}local}$; it will later advertise this schedule with an age computed on its local clock, *i.e.* $\Delta_{age} = now - T_{gen\text{-}local}$. As with S-MAC, we assume nodes periodically resynchronize their schedules via SYNC packets to avoid problems due to clock drift.

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 [13] and future-request-to-send [10] 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 is ready to send the packet. For example, in Fig. 1, when nodes strictly follow their sleep/wakeup cycles and data is transferred from node 1 to node 2 during regular listen time $t1$, node 2 must then wait until time $t3$ before sending data to node 3. By enabling fast path schedules (shown as dotted boxes), node 3 wakes up at time $t2$, avoiding this delay.

Providing fast path schedules in a scheduled 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 FRTS, and how multiple concurrent fast paths are supported. We review each of these issues below.

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. Whatever routing protocol is in use, we expect the fast path to be established by piggybacking a fast-path setup message along with the first data message in the direction of data flow.

As one example of how FPA would integrate with a routing algorithm we consider original, two-phase directed diffusion [5]. To setup communication between a source and a sink with this protocol, the sink sends an interest message, the source then sends exploratory data that solicits a reinforcement message, and subsequent messages then proceed only on reinforced paths. With FPA, the fast path setup message would be piggybacked on the first data message sent on a reinforced path. The same approach would be used with one-phase push diffusion [3], even though that protocol omits interest messages. In one-phase pull diffusion, after receiving interest messages from sinks, sources send data on the preferred gradient without sending exploratory data, so the first data message should be used do fast path setup.

We have also considered fast-path Internet-based routing algorithms. With DSR, sinks send Route Request, and sources send Route Reply to establish routing paths. Either route message could be piggybacked with data. The fast path setup message should be piggybacked with the first data message, wherever it is.

We next consider the process handling a fast-path setup message. In a basic network we expect all nodes to follow the global schedule. As shown in Fig. 1, the source node communicates with node 2 during their regular listen time. Starting from node 2, nodes need to inform next hop nodes where to add the extra wakeup period. Assume that 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 that 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 node 1 and 2 finishes. Similarly the position P for node 4 to place its fast path schedule is $2d$ from regular listen time. P is piggybacked on the first data packet in the flow as we described above. On receiving the setup message, nodes along the path set up the fast path schedule accordingly.

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.

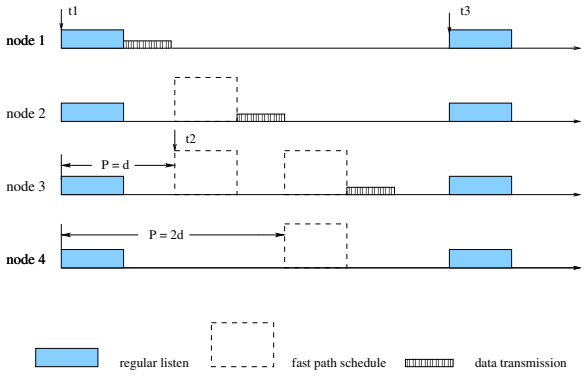

Fig. 1. Fast data forwarding when adding fast path schedules

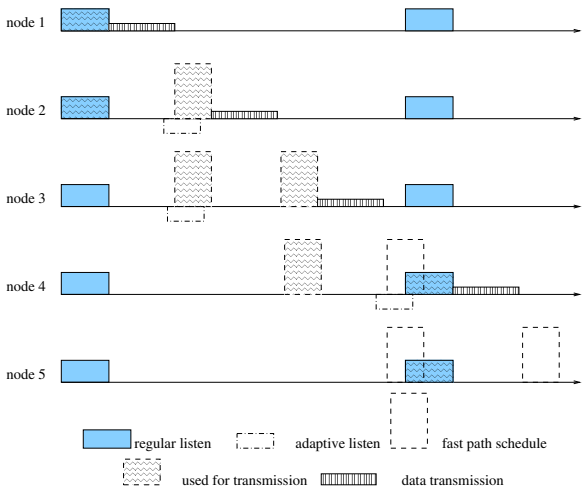

Fig. 2. Regular listen, adaptive listen and fast path schedule overlap

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 on next hop nodes so that data can be forwarded without interruptions. If packets are delayed due to interference or corruption, they may miss their slots, 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 fast path schedule can not finish before regular listen time, we use the regular listen time as the fast path schedule instead. In Fig. 2, fast path schedule overlaps with the regular wakeup period on node 4 and 5, so data is transmitted 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 with fast paths, data 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 eventually expire.

So far, we have discussed how to apply fast path schedules on a single path. In large sensor networks, 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 nodes. 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. In addition, we demonstrate that the use of our global schedule algorithm can save energy and 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 large networks. 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 [4] and S-MAC [13] 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). 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 verify 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 simulate networks 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. (Nodes listen for other schedules for one synchronization period and if they hear none, they select their own schedules.)

Unfortunately, we find unpredictable radio propagation means our control packet is not always received by the nodes and in most experiments, 18–36% fail to participate.

b) Results: Table I shows the sleep schedule information we collect from the four tests performed. All tests show multiple schedules, with up to four different schedules being independently created. For example, in Test 4, there are four different schedules formed in the

TABLE I

EXPERIMENTS SHOWING MULTIPLE SCHEDULES IN LARGE NETWORKS

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 fail 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 secondary schedules.

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)

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 are aware of 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. As described before, maintaining multiple schedules costs additional energy for these border nodes.

Why are so many nodes aware of 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. Instead, we observe that cluster boundaries are quite complicated. This occurs for several reasons.

First, nodes that initiate schedules sometimes can leave the schedules they create. This result is due to an optimization in S-MAC called *singleton schedule elimination*. After nodes announce their schedules, they consider the schedules temporary until they recognize that someone else has joined in. While the schedule is temporary the initiating node will switch to a new schedule it hears. This optimization is a good fit 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 other 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 adopted 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 simplifying schedule selection with our global schedule algorithm rather than with special cases like singleton elimination.

Finally, the main reason why so many nodes discover multiple schedules is because of the "gray region" in radio propagation [14], [11]. 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 the following separate laboratory experiments and ACM Sensys 2003 [6], we demonstrate that global schedule assignment will successfully allow all nodes to converge to a single schedule quickly.

B. Global Schedule Algorithm

In this experiment, we implement the global schedule algorithm as described in Section II-A. We use similar methodology to previous multiple schedule experiments and turn on 40 Mica2Dots randomly in a linear topology inside a building. Nodes initially run S-MAC with GSA disabled, then after the network becomes stable, they enable the global schedule algorithm. The purpose of the experiment is to measure energy consumption rate and convergence time for GSA. We count the time spent on each radio state to compute the energy indirectly, modeling power consumption during transmission at 60mW, during listening and receiving as 45mW, and sleep power consumption as 90μW [1].

Fig. 3 and Fig. 4 show the result from one test (additional runs produced qualitatively similar results and are not shown due to space constraints). Time is shown relative to activation of the GSA algorithm. In this test, without GSA, the 40 nodes select two independent schedules when the network initializes. When experiment starts, some nodes are aware of both schedules, and others only know one schedule. At time 0, GSA is enabled and nodes begin switching schedules. Fig. 3

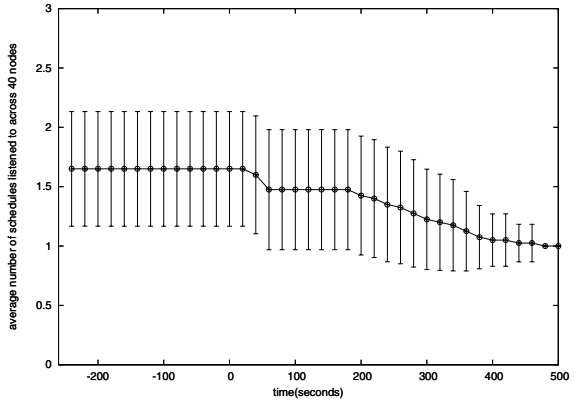

Fig. 3. Mean number of schedules listened to across all nodes in the experiment. Error bars show standard deviations.

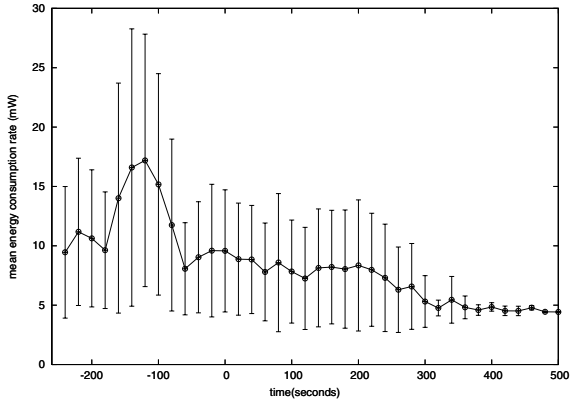

Fig. 4. Mean energy consumed per node over 1s period. GSA is activated at time 0. Error bars show standard deviations.

demonstrates that the number of schedules that each node listens drops after GSA is activated, and Fig. 4 shows the corresponding decreased energy consumption rate. In about 450s, the network stabilizes with all nodes listening to only a single schedule. Energy consumption converges on around 4.3mW per node, in line with what we expect at 10% duty cycle.

We were somewhat surprised at the delay in converging to a single schedule. All nodes settle on a single schedule very quickly, within 20ms of activating the GSA algorithm. However, nodes continue to listen on old schedules for much longer, *i.e.*, about 450s. This delay is because S-MAC is *very* conservative on giving up old schedules, because dropping a schedule that is still in use could result in network partition. S-MAC only discards an old schedule when it confirms that all nodes previously using that schedule have adopted a different schedule. It confirms this by hearing a SYNC packet from that node. Since only one node can succeed in sending a SYNC packet each frame and the neighborhood size is around 26, the minimal time needed to forget an old schedule is 49s (10s SYNC delay plus 26 neighbors each requiring a 1.5s frame to send). It takes about ten times longer than this minimum for two reasons. First, nodes must contend to send SYNC packets—with 26 neighbors, each node contends with many other nodes, cause occasional collisions. Second, distant nodes are weakly connected, so in addition to contention, their packets may be lost due to noise. One could reduce these effects by adapting SYNC frequency to neighborhood size. Evaluation of possible optimizations is an area of future work.

Another observation about GSA is that it reduces *variance* in energy consumption. Adopting a single schedule promotes even energy use. Finally, the large spike in energy consumption around -100s in Fig. 4 is due to the S-MAC neighbor discovery algorithm. During neighbor discovery, nodes remain awake for a whole synchronization period to determine if there are any previously unknown schedules.

We conclude from this test that a global schedule reduces energy consumption. The savings is about a factor of two here; we would expect to see more energy savings in larger and more geographically dispersed networks where more schedules arise. From the experiment, we also demonstrate the GSA algorithm converges reasonably quickly.

C. 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. We selected this topology to highlight multi-hop fast-path behavior; experimental exploration of more complex topologies (perhaps with overlapping paths) are future work. 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 25 seconds 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, here configured at a 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. Since we have a static topology, when we enable the fast path algorithm we pre-configure fast path schedules based on nodes physical positions. (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.

Our experiments compare our GSA and FPA against unmodified S-MAC with adaptive listening [13]. We were not able to compare against other scheduled MAC protocols such as T-MAC because implementations were not available to us at the time of our experiments. However, we expect that T-MAC would show similar performance to S-MAC since T-MAC latency is similar to S-MAC with adaptive listening.

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

Fig. 5 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 d as described in Section II-B. In 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

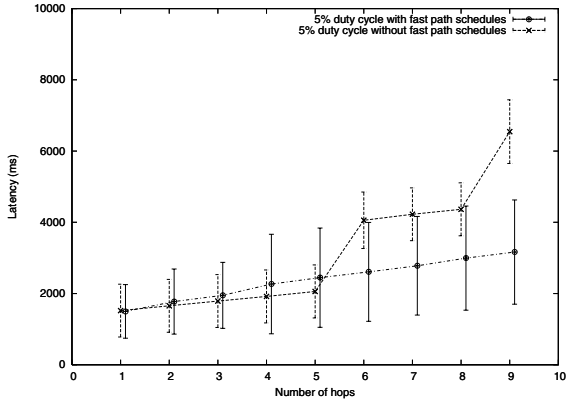

Fig. 5. Mean message latency on each hop of all successful transmissions

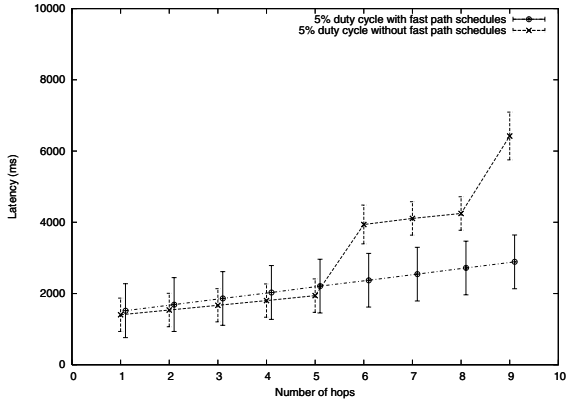

Fig. 6. Mean message latency on each hop of all successful transmissions with no retransmissions

hop 6. This is due to the effects of the adaptive listen algorithm. In networks where nodes can hear only their 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 will wake for adaptive listen. Experimentally we observe *multiple* adaptive listen events, allowing packets to travel five hops before suffering a schedule miss. This happens because of long radio propagation distance. In our experiment, nodes from hop 1 through hop 5 are able to hear *all* packets exchanged between the first five hops, so we observe three

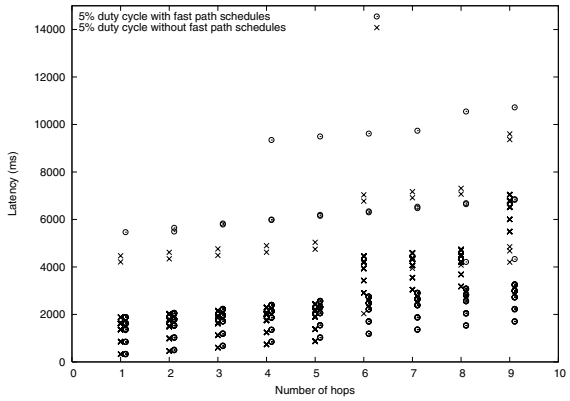

Fig. 7. Message latency on each hop of all successful transmissions

successive adaptive listen events. While this result occurs consistently in this experiment, we cannot expect such good fortune in all cases.

The general trend in Fig. 5 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 with the mean of non-fast-path case. To understand why we see such large variance in the results, Fig. 7 shows a scatter plot of all the successful transmissions in the experiment. 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. 7 shows that once the packet is retransmitted it will get on the next available fast path.

Fig. 6 shows the mean per-hop latencies of all packets *except* those that are retransmitted. The trend is the same as Fig. 5, 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 S-MAC schedules. The sender needs to wait till the next wakeup period to send messages. This initial delay could be avoided by synchronizing data generation with S-MAC schedules, Alternatively, a possible option is to generate fast-path schedules that are synchronized with data generation times.

IV. Related Work

T-MAC [10] 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 [2] 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 [8] 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. [7]. 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 nodes according to their depths 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. Secondly, 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 present 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 experiments multiple schedules are common in large networks, motivating the global schedule algorithm. We also discover that the boundaries of virtual clusters are quite diffuse, and many nodes become border nodes without the global schedule algorithm. We evaluate the energy savings of our global schedule algorithm in a two schedule network and 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. The authors were also partially supported by ChevronTexaco through USC CiSoft. Source code to the algorithms is available on the author's web site.

References

[1] `http://www.chipcon.com/files/CC1000_Data_Sheet_2_2.pdf`.

[2] 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.

[3] 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*, pages 146–159, Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] van Dam, Tijs, 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.