

Network Scheduling for Data Archiving Applications in Sensor Networks

Yong Yao*

S. M. Nazrul Alam*

Johannes Gehrke*

Sergio D. Servetto†

*Department of Computer Science, †School of Electrical and Computer Engineering
Cornell University

{yao,smna,johannes}@cs.cornell.edu, servetto@ece.cornell.edu

ABSTRACT

Since data archiving in sensor networks is a communication intensive application, a careful power management of communication is of critical importance for such networks. An example is FPS, an adaptive power scheduling algorithm that combines slotted scheduling with a CSMA MAC [7]. In this paper, we first propose a new global power scheduling protocol called *Multi-Flow Power Scheduling (MPS)* that delivers more data and consumes less energy than existing power scheduling protocols. MPS sets up a transmission schedule through standard data aggregation and dissemination operations, however since it creates a global schedule it does not scale to large networks. We then present a new power scheduling protocol called *Hybrid Power Scheduling (HPS)* that retains the scalability of FPS while maintaining the energy efficiency and high data delivery rate of MPS. In a thorough simulation study, we compare HPS and MPS, and our results show the efficacy of HPS.

1. INTRODUCTION

Data archiving applications in sensor networks periodically gather snapshots of the state of the data field observed by the network, and store them in a central repository outside the network for later data analysis [2, 14]. Since sensor nodes are battery powered and battery replacement is not feasible in many deployments, efficient power management is very important for the long term success of such data archiving networks. As radio communication dominates power usage in sensor networks [11], minimizing the cost of wireless communication is crucial for maximizing the lifetime of a sensor network. In data archiving networks, communication power can be conserved by 1) reducing the total amount of data that needs to be sent, 2) turning off the radio in idle states, and 3) scheduling message transmissions to reduce the probability of communication interference and collisions [1, 4, 5, 6, 10]. In this paper, we focus on the scheduling of message transmission.

Scheduling of message transmission involves a *network power scheduling protocol* that creates a transmission schedule specifying the time period when each node can send and receive messages. When not sending or receiving, the node can turn the radio off to conserve energy. Previous research on scheduling of message transmission focused on lightly-loaded sensor networks with a periodic workload, e.g., to monitor ambient temperature and report a reading every 30 seconds [3, 7]. At such a low data rate, the primary tasks of power scheduling protocols are to reduce the duty cycle of communication and the probability of collisions while minimizing overhead.

However, recent data archiving networks produce data at a much higher rate. For example, a recent deployment for volcano-monitoring recorded two channels of data at a rate of 100Hz per node [2] and a sensor network for structural monitoring produced data at a rate of 600bps [14]. At such high data rates, collision and interference between nodes can significantly increase power usage unless transmissions are carefully scheduled.

In this paper, we propose a novel scheduling protocol called *Multi-Flow Power Scheduling (MPS)* that coordinates transmissions of neighboring nodes such that interference and collision is minimized. MPS delivers data at a higher rate and consumes less energy as compared to existing approaches (e.g., Flexible Power Scheduling (FPS) [7] or Duty Cycling [3]). However, MPS does not scale well to very large networks, due to the global nature of its scheduling. We deal with the scalability issue by combining MPS with FPS, to arrive at *Hybrid Power Scheduling (HPS)*, which achieves the scalability of FPS, while still maintaining the higher data rate and energy efficiency of MPS. We implemented these protocols in TinyOS, and our experiments show that HPS and MPS have higher data delivery rates and are more energy efficient than other existing approaches.

The rest of paper is organized as follows. We introduce our terminology in Section 2, where we also discuss related work. We describe our new algorithms in Section 3, and we report results on their performance in Section 4. We conclude in Section 5.

2. PRELIMINARIES

A transmission schedule divides time into slots of equal size, and specifies the radio state of each time slot, i.e., send, receive, or idle. In this section we discuss related work on network power scheduling, and we present our system model. We assume that the reader is familiar with sensor networks and basic power scheduling terminology.

2.1 Network Power Scheduling: Related Work

Duty Cycling. In TASK, every node is turned on for a short, synchronized period in each round [3]. The period is long to allow transmissions of data generated by the node and other traffic. The length of a period is configurable by the workload and the size of the network. The protocol has little overhead, and it works well for broadcasting messages. However, this approach does not reduce collisions, and as a result the throughput is low.

Depth Based Scheduling. In a *Depth Based Scheduling* protocol, nodes are organized in a tree, and they wake up in sequential order according to their depth in the routing tree [8]. For example, nodes of depth d and $d + 1$ are awake at the same time period for transmissions from nodes of depth $d + 1$ to their parents of depth d . In the next period, nodes of depth $d - 1$ and d are scheduled for transmission. The Depth Based Scheduling protocol is particularly suitable for data aggregation. However, such a protocol still has collisions and resulting transmission failures.

Flexible Power Scheduling (FPS). FPS is designed for lightly loaded networks where each node has a similar workload within each period [7]. Nodes are divided into children and parent nodes, where parent nodes act as arbitrators that pick non-conflicting transmission slots for their children. This avoids all collisions between the children nodes of a parent. FPS is adaptive to a slowly changing workload via exchange of periodic advertisement and reservation messages that change the schedule. However, since in wireless communication the interference range is usually larger than the communication range, in FPS collisions still exist between nodes that have a different parent, especially in heavily loaded sensor networks.

FPS is designed to be adaptive by allocating to each node an *extra unit of demand*, a special time slot. Thus when a reservation is made locally, bandwidth is immediately available all the way to the sink [7]. However, for a static network, this extra unit of demand in each node causes significant overhead in the form of idle listening which increases energy consumption significantly. We can remove those slots to reduce this overhead, however that would make FPS less adaptive. We will compare our work to both versions of FPS, referring to the original FPS as *Adaptive FPS* and to FPS with no extra unit of demand as *Static FPS*.

Data Transmission Algebra. DTA develops an algebraic framework that allows a centralized optimizer to generate a schedule for transmission [12, 15]. The optimizer needs to first collect information about network topology, transmission delays, data rate, etc., in order to calculate the optimal schedule. In practice, it incurs a very high cost of collecting and maintaining the meta information.

2.2 System Model

The major components of a sensor network data archiving system include *data reduction*, *network services*, and *power scheduling*. Other useful components are *time synchronization* and *localization*. Let us discuss each of them to give a picture of our system model.

Data Reduction. Recent studies have shown that sensor readings are highly correlated in both time and space [5]. Therefore we assume original sensor readings are first processed by a *data reduction component* which explores inherent data redundancy and reduces the size of data. For example, sensor readings of a group of neighboring nodes could be compressed jointly, we could build complex models of sensor data based on historical observations to predict future sensor readings [6], or we could install predicate filters that discard unwanted data locally [1]. The output of the data reduction component must be reliably delivered to a gateway node which we call the *server*; no further compression is possible, and loss of a single message could result in large data loss.

Network Services. From the data reduction component, data is sent along a multi-hop path back to the server. The multi-hop path is constructed and maintained by a *multi-hop routing protocol*, typically as a tree rooted at the server. A new node joins the tree by selecting one of its neighbors as its parent that minimizes the total transmission cost, a function of both the number of hops to the root and the link qualities along the path.

Time synchronization and localization. Sensor readings describe events in the physical world, and are meaningless without time and location of the event. Both multi-hop routing and time synchronization components periodically broadcast a message to recover from potential drifts and failures. Such control messages are important and need to be scheduled as well.

Power Scheduling. In our data archiving system, the power scheduling component, called *ScheduledComm*, resides between the MAC layer and the network layer in the network protocol stack. It constructs and maintains a transmission schedule that controls the transmission of all messages. Time is divided into slots of equal size in the schedule. The size of a time slot is chosen such that it is long enough to send several messages if there is no interference. Current time synchronization algorithms of sensor networks have an error bound about tens of micro-seconds, less than one-thousandth of the length of a time slot [9]. Thus all time slots in the network can be aligned with a very small overhead. The *ScheduledComm* component receives various types of messages for transmission from the upper components, e.g., data messages, routing messages, and control messages to setup and adjust the schedule. It allocates separate time slots to different types of messages. In this paper, we focus on data messages as they dominate communication in a data archiving network.

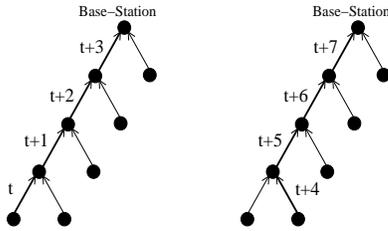


Figure 1: Single Flow Power Scheduling (SPS) – (left: chunk 1; right: chunk 2).

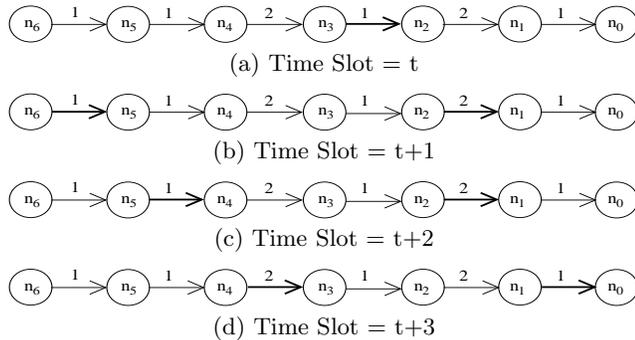


Figure 2: Multi-Flow Power Scheduling (MPS)

3. ALGORITHMS

In this section, we describe our new scheduling protocols *Multi-Flow Power Scheduling (MPS)* and *Hybrid Power Scheduling (HPS)*.

3.1 Multi-Flow Power Scheduling (MPS)

Multi-Flow Power Scheduling (MPS) sets up a distributed transmission schedule in the network with the goal to minimize the probability of interference and collisions during data transmission. We assume that nodes have been assembled in a routing tree rooted at the server.¹ Every node has a local schedule, specifying when the node will send and receive data. The local schedules of nearby nodes are coupled such that whenever a node receives data from its child, the node always forwards the data to its parent in the next time slot.

In MPS, the output of the data compression component is divided into *chunks* as the smallest unit of data that is scheduled. Each chunk of data is transmitted without interruption from the originating node all the way up the routing tree to the server, as the further data reduction benefit of pre-compressed data in intermediate nodes is usually low. All intermediate nodes are scheduled one after another. As a result, only small message buffers are required.

The simplest approach for the transmissions of all chunks in the network is to schedule the chunks sequentially; only one chunk is under transmission at any time, and the transmission of the next chunk starts after the current chunk has been delivered to the server. In particular, chunks of the

same node are scheduled consecutively, and chunks of different nodes are arranged in the preorder sequence of the original nodes in the routing tree. We refer to this approach as *Single-Flow Power Scheduling (SPS)*. Figure 1 shows an example of how two chunks are transmitted in SPS. The time slot when a chunk is transmitted through a link is shown next to the link. Delivery of Chunk 1 starts at time slot t and is completed in time slot $t+3$. Time slots $t+4$ through $t+7$ are used for the delivery of Chunk 2.

Although SPS does not have any interference, it severely limits the network capacity since there are no parallel transmissions: only one pair of nodes is sending and receiving at a time. MPS tries to improve upon SPS by allowing overlap between transmissions of chunks while trying to minimize interference. Two types of parallelism can be exploited: intra-node parallel transmissions of chunks produced by the same node, and inter-node parallel transmissions of sequential chunks from different nodes. In general, to schedule two chunks C_1 and C_2 , C_2 will be subject to a delay and start later than C_1 to avoid any potential interference with C_1 . The delay depends on both the network topology and the link quality, since it takes more time slots to send a chunk over a less reliable link due to retransmission. The number of time slots needed to send a chunk is called the *linkDelay*, and varies a lot in sensor networks. The *linkDelay* can be calculated directly from the link quality, which is maintained by most routing protocols to choose the optimal route.

Figure 2 shows a simple example of inter-node parallelism in MPS. The *linkDelay* is displayed over each link, and active links that are busy for transmission are drawn in bold. The interference range is less than 2 hops. Thus both n_3 and n_6 can transmit simultaneously without a collision. In time slot t , a chunk C_1 is being forwarded by n_3 . In the next time slot, C_1 is being forwarded by n_2 , and a new chunk C_2 is being sent by n_6 . Although n_3 and n_6 are outside the interference range, C_2 has to wait until C_1 has been delivered to n_2 ; otherwise, the transmission of C_2 will collide with C_1 in time slot $t+2$, since the *linkDelay* of n_2 is longer than n_5 .

While MPS sets up a complete transmission schedule for the whole network, it is quite simple to compute the local schedule on every node with a low overhead. To construct the schedule, the network generates at most two messages for each edge of the routing tree; an intermediate node may receive and forward hundreds of data messages during one schedule in a heavily-loaded network. One reason for the low overhead of MPS is that it has a hierarchical structure that utilizes the structure of the underlying routing tree, so we can construct the schedule in a bottom-up manner from the leaf nodes by repeatedly combining the schedules of nodes of the subtree. Another reason is that a local schedule has a very small size. In fact, only a small fraction of communication slots are determined after the schedule is initialized. Most remaining communication slots will be resolved later during data transmission by attaching additional information to data messages to fill in the schedule.

¹The routing algorithm we use is a variant of MINTRoute [13]. Our scheduling algorithms can be easily extended to any tree-based routing algorithms.

Algorithm 1 InitSchedule

Require: numChunks, linkDelay, chunkDelay

- 1: $ss.linkDelay = linkDelay$
- 2: **if** ($numChunks > 0$) **then**
- 3: $ss.length = linkDelay + chunkDelay \times (numChunks - 1)$
- 4: $ss.recvSlot = 0$
- 5: **if** ($chunkDelay > linkDelay$) **then**
- 6: $ss.1stChunkDelay = chunkDelay - linkDelay$
- 7: **else**
- 8: $ss.1stChunkDelay = 0$
- 9: **end if**
- 10: **else**
- 11: $ss.length = 0$
- 12: **end if**

Algorithm 2 AggregateSchedule

Require: ss_i (ss structure from child i)

- 1: **if** ($ss.length > 0$) **then**
- 2: $ss.length = ss.length + ss_i.1stChunkDelay + ss_i.length + ss.linkDelay$
- 3: **else**
- 4: $ss.length = ss_i.length + ss.linkDelay$
- 5: $ss.recvSlot = ss_i.recvSlot + ss_i.linkDelay$
- 6: **if** ($ss_i.1stChunkDelay > ss.linkDelay$) **then**
- 7: $ss.1stChunkDelay = ss_i.chunkDelay - ss.linkDelay$
- 8: **else**
- 9: $ss.1stChunkDelay = 0$
- 10: **end if**
- 11: **end if**

3.1.1 Schedule Initialization

In this section, we explain how MPS sets up a transmission schedule for the whole network. The sensor network is organized as a tree by the multihop routing protocol. For each subtree of the network, MPS constructs a local schedule at the root node of the subtree that specifies 1) when the node will send its own chunks to its parent, and 2) when the node will receive the first chunk from each of its children.

The schedule initialization process consists of two phases: a bottom-up schedule aggregation phase that generates a schedule framework, followed by a top-down schedule dissemination phase that completes the schedule. The construction of a schedule mirrors the computation of an aggregate query in sensor networks. As shown in Algorithm 1, a leaf node calculates the length of its local schedule from the time when it sends the first chunk to the time when the parent receives the last chunk, by inserting a *chunkDelay* before each chunk except for the first one. The node generates a struct *ScheduleSummary* (or in short *SS*) as a description of its local schedule and sends the *SS* to its parent. The *SS* also includes the first reception slot (*recvSlot*) during which the parent will receive a chunk from the child, and the length of the child's schedule (*length*). We explain the details of the computation of *chunkDelay* in MPS in Section 3.1.2.

The size of *SS* is a constant, and the struct is small enough to fit in a single message. After receiving the *SS* from all

Algorithm 3 CalcChunkDelay

Require: $n_p, n_a, chunkDelay_p$

// parent of n , parallel ancestor of n , chunkDelay of n_p

- 1: $delay = SumOfLinkCost(n, n_p, \dots, n_a)$
- 2: $chunkDelay = MAX(delay, chunkDelay_p)$

of its children, the parent node merges all *SS* and create a new schedule for the whole subtree below it. Algorithm 2 shows how to *aggregate* the local *SS* with a SS_i produced by a child. The algorithm inserts a delay before the first chunk of SS_i if necessary (*1stChunkDelay*). This process repeats until the server has generated a complete schedule. Any existing system that supports user-defined in-network aggregation can be extended to compute the MPS schedule.

After the schedule aggregation phase, two parameters are still missing of a local schedule: the starting slot of the schedule, and the allocated size of the schedule. (Since the overall schedule has a maximum size, not all chunks can necessarily be delivered within one run of the schedule.) In the schedule dissemination phase, the server notifies its children of the starting slot and the actual length of their schedules by sending another message to each of them. Recursively, these nodes calculate and distribute the schedule information to their children, and so on.

The transmission of the schedule initialization messages is under the control of the Depth Based Scheduling protocol as described in Section 2.1. By allocating sufficient time and enforcing an ACK for each message, the probability of message loss is kept very low. In addition, the length of an MPS schedule is no more than a couple of minutes. As observed in Section 4, the schedule initialization overhead is very low in MPS, and it is feasible to set up a new schedule in every cycle. Therefore, MPS is able to recover from node or link failures relatively fast, once the routing tree is repaired by the routing protocol. It is also adaptive to sensor networks that have a dynamic workload.

3.1.2 chunkDelay Computation

In order to allow parallel transmission of two sequential chunks without causing any interference, MPS postpones sending of the second chunk. For two chunks of the same node, as shown in Algorithm 1, a *chunkDelay* determines when the second chunk can start after the first chunk. For two sequential chunks of different nodes, MPS computes a delay at the lowest common ancestor of the two nodes during the schedule aggregation phase, determining when the second chunk can start after the ancestor forwards the first chunk. Suppose that node n receives two schedules SS_1 and SS_2 . To combine SS_1 and SS_2 , the Algorithm 2 needs to evaluate the delay for the last chunk C_1 of SS_1 and the first chunk C_2 of SS_2 . Let n_2 denote the originating node of C_2 . If the *chunkDelay* of C_2 is larger than the total linkCost of the path from n_2 to n , then an extra delay of their difference will be inserted at n_2 after n forwards C_1 ; otherwise, n_2 can send C_2 simultaneously as n forwards C_1 . Essentially, the delay measures the length of time from the time when n forwards C_1 until the time when C_2 is being sent by its originating node. It must be long enough such that 1) the first transmission of C_2 does not interfere with the concurrent

transmission of C_1 at n or n 's ancestor, and 2) no further transmissions of C_2 will interfere with any remaining transmissions of C_1 . The second condition is necessary because if C_1 would be forwarded over a series of less reliable links (with a larger *linkDelay*) than C_2 , then C_2 may catch up C_1 in a few hops.

Let us introduce the notion of the *parallel ancestor* p of a node n , the first ancestor of the node in the routing tree such that n and p can send messages simultaneously without any anticipated collisions in the network, if both packets were sent up all the way to the root of the routing tree. In most cases, the parallel ancestor is close to the node (about three to four hops), and any ancestors above the parallel ancestor do not interfere with the node either. Initially, the parallel ancestor of a child of the root is just the root. To combine its computation with the process of multi-hop routing, a node that has found its parallel ancestor attaches the path to its parallel ancestor in the routing message. After receiving this list of links, a child of the node caches the list and selects a new parallel ancestor from the list.

Algorithm 3 shows how to compute the *chunkDelay* in a top-down manner at a node n , according to the cached link list at n and the *chunkDelay* of n 's parent. The *chunkDelay* of the server is zero. The *chunkDelay* of any child of the server equals the *linkCost* from the child to the server. For any other node n , the *chunkDelay* of n is at least the *chunkDelay* of its parent. Next, n needs to increase its *chunkDelay* only if the transmission of a chunk at n collides with the transmission of the previous chunk generated by n . Since n does not interfere with its parallel ancestor, thus if the total cost of the path from n to its parallel ancestor is smaller or equal to the *chunkDelay*, then the previous chunk has already been received by n 's parallel ancestor after *chunkDelay* slots, and thus n does not need to increase its *chunkDelay*. Otherwise, we set n 's *chunkDelay* to the total *linkCost* of the path.

The algorithm is performed at the same time as updates to the routing tree are made, thus it has very little overhead and automatically updates the *chunkDelay* once the network topology changes.

The last problem is to determine whether two links collide with each other while choosing their parallel ancestors. While it is hard to get accurate results, there exists a maximum interference range of radio communication, where two links are guaranteed not to interfere if they are far away from each other. As mentioned in Section 2.2, locations are available in most data archiving networks. Thus if the distance between two links is larger than the maximum interference range, these two links will (likely) not collide. In dense networks, if the node location is not available, we can use the depth of the link as an approximation. In general, it is not necessary to remove all collisions in MPS. The remaining collisions can be easily resolved by the underlying MAC protocol.

3.1.3 Data Transmission

Not all communication slots are settled down after the schedule initialization process. To fit the description of a partial schedule in a single message, SS only includes the first re-

ception slot. Further communication slots are resolved in the data transmission process whenever a node receives a data message. After receiving a chunk, the node immediately forwards the chunk to its parent in the next available slot. The next reception time slot from the same child is attached to the data messages of the current chunk. Note that if a node misses all data messages of a chunk from a child at pre-allocated time slots due to a network failure, the node will switch to a low duty cycle mode, where it wakes up for a short period of time in the beginning of every time slot, until it has received another chunk from the child, or the schedule of the child has expired.

3.2 Hybrid Power Scheduling (HPS)

The biggest drawback of MPS is its lack of scalability. Since MPS sets up a complete transmission schedule by aggregating the local schedules and disseminating schedule details, it does not scale efficiently to large networks. Specifically, the length of the schedule initialization process is proportional to the depth of the routing tree; the probability of transmission failures of schedule initialization messages also increases in large networks. In an extreme case, the network condition can even change when the schedule initialization process finishes. On the other hand, the maximum data rate of a node is limited by the maximum network capacity as determined by the server, which is inversely proportional to the network size. In data archiving applications, nodes far away from the root usually have fewer data to send and forward as compared to nodes close to the root. The cost of setting up the schedule for MPS may exceed the benefit of reducing the interference and collision for such nodes. So we propose hybrid power scheduling (HPS) where nodes in a scheduling region that includes the server run MPS, nodes outside the region run FPS, and nodes on the boundary run both FPS and MPS.

Typically, nodes outside the scheduling region are far away from the root, and have a small amount of data to send. Thus, FPS for these nodes does not result in many collisions. In HPS, we thus combine the scalability of FPS with the efficient power usage and data delivery characteristics of MPS. Certainly the performance of HPS depends on the choice of the scheduling region, which depends on data workload, network topology and other network parameters. In our experiments, all nodes with depth $< d$ run MPS, where d is configured such that any node of depth d or higher have less than one tenth of their time slots occupied for data transmission.

4. EVALUATION

We implemented the *ScheduledComm* component and the MPS and HPS algorithms in TOSSIM, and evaluated the performance of various power scheduling protocols for data transmission in an 8 by 8 grid network of 64 nodes. The server is located at the top-left corner. The distance between two vertically or horizontally adjacent nodes is 10m. The communication range is 15m, thus two diagonally adjacent nodes can communicate directly as well. The maximum interference range is 30m. (We obtained similar experimental results for other values of the interference range such as 25m and 35m.) Our system utilizes the existing multihop routing and time synchronization components of TinyOS to maintain the routing tree and the global time; every 30 sec-

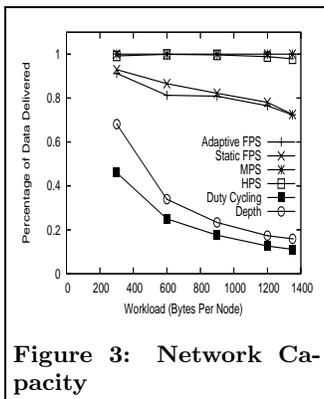


Figure 3: Network Capacity

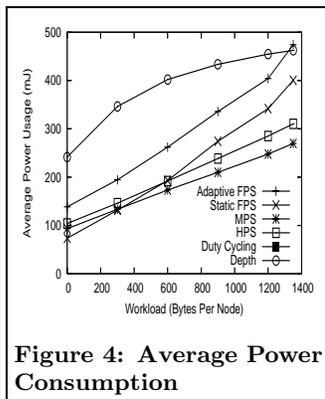


Figure 4: Average Power Consumption

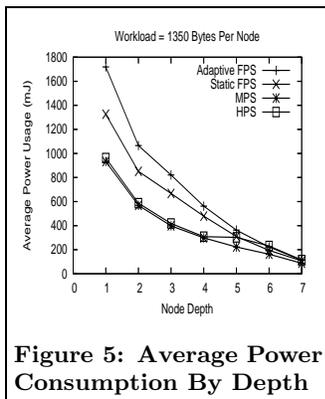


Figure 5: Average Power Consumption By Depth

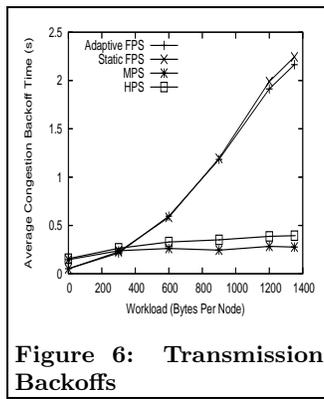


Figure 6: Transmission Backoffs

onds, each node broadcasts a routing and time synchronization message. These messages are scheduled by the Duty Cycling protocol as described in Section 2.2.

Every node has a fixed amount of data to deliver to the root before the end of the simulation, and we vary this amount during our experiments. We only measure the performance of the network in steady state after every node has joined the routing tree and has become time synchronized. We have tested six different power scheduling protocols for the transmission of data messages: MPS, HPS, adaptive FPS, static FPS, Duty Cycling, and Depth Based Scheduling. Adaptive FPS is the original FPS with one initial demand, and static FPS does not have the extra unit of demand. The size of an MPS schedule is one minute, and there is a schedule initialization process at the beginning of every minute. The size of an FPS cycle is 15 seconds, and the same schedule is used in every cycle. The Duty Cycling and Depth Based Scheduling protocols are configured such that each node is awake approximately 10 percent of the time, in order to make their power consumption comparable to the other protocols. The simulation time is four minutes, such that each protocol can generate a sufficient number of cycles.

4.1 Network Capacity

Figure 3 shows the percentage of data delivered to the root at the end of the simulation. MPS delivers all data to the root, and it does so for all tested workloads. For FPS, the percentage of data delivered decreases as the workload increases. This is because more bandwidth is consumed due to interference and collisions within the network. With a workload of 1350 bytes, about one-third of the data is not delivered to the root. Duty Cycling and Depth Based Scheduling have the worst capacity due to the large amount of collisions in the network. Although their capacity can be improved by increasing the amount of time a node is awake, it also increases the energy consumption without improving the energy efficiency.

4.2 Power Consumption

Figure 4 compares the average radio power consumption per node. We use the default energy model of mica2 nodes in TOSSIM. MPS consumes the least energy and saves between 30% to 50% energy compared to the Static FPS and Adaptive FPS, respectively. Note that the power consumption of MPS and HPS increases linearly and slowly as the workload rises. The energy efficiency of MPS and HPS even

increases in a heavily-loaded network, as the overhead of schedule initialization becomes a marginal cost. The power consumption of FPS increases much faster: as more time slots are filled up for data transmission, the probability of interference and collisions increases as well. Duty Cycling and Depth Base Scheduling consume the most energy. They are simply not well suited to the constraints under which a data archiving network must operate.

Power consumption is not uniformly distributed in data archiving networks. Nodes close to the root receive and forward much more data than nodes near the leaves. Such nodes close to the root form the bottleneck of the network lifetime, since they usually run out of power first. Thus it is more important to cut down the power usage of these nodes than to reduce the total power consumption of the network. Figure 5 displays the average power consumption according to node depth for a fixed workload. Nodes are grouped by their depth in the routing tree. As shown in the figure, although MPS and FPS have a similar cost at the leaf nodes of the tree, MPS saves almost 50% energy at nodes near the root. HPS consumes more energy at nodes towards the leaf, but saves the same energy as MPS at nodes near the root. Thus the extra energy consumed by HPS has very little effect on the network lifetime.

4.3 Transmission Backoffs

To better understand the performance of various power scheduling protocols, we count the number of transmission backoffs at the MAC layer. The MAC component of TinyOS always checks activity at the communication channel before sending a message, and automatically backs off if the channel is not clear. Note that this backoff does not resolve all collisions due to the hidden terminal problem, and still consumes energy during the backoff. If the network is busy, a node may backoff many times before it gains the access to the channel. The average congestion backoff time per node approximates the level of interference. As shown in Figure 6, in MPS this time is almost constant, as most interference exists only during the schedule initialization phase and the transmission of routing messages. In FPS, the number of backoffs increases exponentially as expected.

5. CONCLUSIONS

In this paper we propose a solution to conserve communication energy via power scheduling in data archiving networks. Such networks generate different types of messages with a different pattern. A new *ScheduledComm* component between the MAC layer and the network layer supports various scheduling protocols, and allows for separate transmissions of different types of messages using distinct scheduling protocols. We focus on the transmission of data messages, which dominates radio communication in data archiving networks. We present a novel scheduling protocol (MPS) that is energy efficient by significantly reducing the probability of communication interference and collisions, especially for heavily-loaded networks. We also propose a hybrid scheduling protocol (HPS) that combines the existing local power scheduling protocol to improve the scalability without sacrificing the network lifetime. Our simulation results show that our protocols save more energy than the previous approaches, and have a higher capacity as well. Future work includes extending our scheduling protocols to mesh networks and networks with multiple servers.

Acknowledgments. We thank Barbara Hohlt for the code for FPS and for clarifying discussions, and we thank the anonymous reviewers for helpful comments. This work is supported by CAREER Grant IIS-0133481 and Grant IIS-0330201 from the National Science Foundation (NSF). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

6. REFERENCES

- [1] D. Abadi, S. Madden, and W. Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. In *VLDB*, 2005.
- [2] G. Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. Deploying a wireless sensor network on an active volcano. *IEEE Personal Communications*, Special Issue on Data-Driven Applications in Sensor Networks, 2006.
- [3] P. Buonadonna, D. Gay, J. Hellerstein, W. Hong, and S. Madden. Task: Sensor network in a box. In *EWSN*, 2005.
- [4] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *ICDE*, 2006.
- [5] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *SIGMOD*, 2004.
- [6] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.
- [7] B. Hohlt, L. Doherty, and E. Brewer. Flexible power scheduling for sensor networks. In *Proc. of IPSN*, 2004.
- [8] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, USA, December 2002.
- [9] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *Proc. of ACM SenSys*, 2004.
- [10] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *SenSys*, 2004.
- [11] G. Pottie and W. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 4(5), May 2000.
- [12] D. Sharma, V. Zadorozhny, and P. Chrysanthis. Timely data delivery in sensor networks using whirlpool. In *DMSN*, 2005.
- [13] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys*, 2003.
- [14] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *SenSys*, 2004.
- [15] V. Zadorozhny, P. Chrysanthis, and P. Krishnamurthy. A framework for extending the synergy between mac layer and query optimization in sensor networks. In *DMSN*, 2004.