Implementing a Low Power TDMA Protocol Over 802.11

Jim Snow, Wu-chi Feng, Wu-chang Feng
(jsnow, wuchi, wuchang)@cs.pdx.edu
Portland State University
Systems and Networking Lab

***Abstract*— A wireless network of remotely located, self-powered video sensors must make efficient use of limited network capacity and power. An ideal media-access protocol for such a network would fully utilize available throughput while using no more power than necessary and maintaining a reasonable level of fairness.**

To accomplish these goals, this paper proposes and evaluates a time-division multiple-access protocol that may run on top of unmodified 802.11 hardware. Our protocol uses an 802.11 access point as a controller for a set of 802.11 clients. The controller allocates timeslots to its clients, during which the clients may send to and receive from the controller. Each client may turn its radio off during other clients' timeslots to save power that would otherwise be wasted contending for a busy channel.

We compare and contrast our protocol to other wireless power-saving mechanisms, then discuss the design and implementation of our protocol and its performance compared to standard 802.11 in terms of throughput, power, and fairness. We conclude that our protocol offers a significant reduction in power consumption and improved fairness. Finally, we show throughput of our protocol to be similar to standard 802.11.

I. Introduction

While most traditional sensor networks measure simple low bit-rate scalar values such as temperature, velocity, and salinity, some environmental observation and security surveillance applications require sensors to gather high bit-rate video data as well. Unfortunately, it is often impractical to connect sensors to the power grid or wired networks. Two main obstacles to video-based sensor network deployment are limited available power and limited network capacity. Such sensors must use limited battery, solar, or wind energy to power themselves and their wireless network interface. The availability of low-power sensor platforms with fast processors has allowed us to build sensors capable of compressing reasonable quality video from a USB webcam in real time prior to transmission using no more than a few watts [1]. However, the network interface on these sensors consumes a third or more of the total power.

In a wireless network scenario, having no network cable limits our throughput, while having no electrical cable limits available power. Our application, however, requires us to fully utilize a power-hungry network adapter. Therein lies the conflict for which this paper hopes to find a reasonable compromise. We would like a wireless MAC protocol that adheres to the following constraints:

- The wireless network interface consumes no more energy than necessary,
- The available data throughput is near full capacity,
- The protocol is easy to implement on commodity hardware, and
- Latency may be high, but not more than a few seconds.

In this paper, we will discuss current approaches to power conservation in sensor networks. We then propose and evaluate an energy conserving media access protocol that may be used on top of 802.11 (or similar protocols). This protocol reduces the amount of idle listening by scheduling traffic in advance, and periodically broadcasting this schedule to all stations. Stations may save power by turning their receivers off when it is not their turn to transmit or receive. Later, we compare the throughput, fairness, and power savings of an 802.11b network with and without this protocol.

II. Related Work in Energy-Conserving Wireless

Many researchers have investigated a wide variety of techniques to limit the power consumption of wireless network interfaces. Feeney and Nilsson [2] show that an Orinoco Silver 802.11 card consumes, on average, 47.4 mW with the receiver turned off (sleeping), 739.4 mW while listening to an idle channel, 900.6 mW while receiving data, and 1346.2 mW while transmitting. Their results show us that unnecessary transmissions are costly, but so is leaving the receiver on when it is not receiving anything. Unfortunately, once we have compressed our video, there is little we can do to reduce network transmissions without sacrificing quality. Therefore, we will attempt to reduce idle listening.

A. Listening less

Many techniques attempt a significant reduction in power usage by leaving the radio in the sleeping state as much as possible. However, this has an unavoidable negative impact on access latency, since a sleeping station is deaf to all network traffic. Periodically, a sleeping station will wake up to exchange information with its access point (in an infrastructure network) or neighbors (in an ad-hoc mesh).

Ye, Heidemann, and Estrin introduce S-Mac [3], a media access control algorithm for low power sensors which trades latency and throughput for a reduction in power consumption.

This material is based upon work supported by the National Science Foundation under Grants ANI-0087761, EIA-0130344, and ITR-0121475.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

In S-Mac, neighboring sensors are synchronized to turn on and off at the same time to save power. While on, sensors contend for the channel using a CSMA-CA protocol similar to 802.11. S-Mac may significantly increase battery-powered sensor lifetime, but it incurs a throughput penalty proportional to power savings.

The 802.11 standard employs a different approach to energy conservation with its Power-Save Mode (PSM) [4, section 11.2]. 802.11 provides a mechanism by which a power-constrained client may notify its access point that it is about to enter PSM. It may then switch off its radio, briefly turning it back on at a regular interval to receive a Traffic Indication Map (TIM) broadcast by the AP. If the TIM indicates that the AP is buffering data for the client, the client must retrieve its data by polling the AP.

Self-tuning power management (STPM), introduced by Anand, Nightingale, and Flinn, [5] is an algorithm to adaptively switch between 802.11 PSM and the default continuously-aware mode. It bases its decisions on network access patterns, the energy savings expected from PSM, the time and power required to change modes, and the total power consumption of the device.

Though PSM is a good method for idle clients to save power, it offers no benefits for busy stations, which must leave their radios on to contend for the channel. PAMAS, introduced by Singh and Raghavendra, [6], reduces power when the network is used at full capacity as well. PAMAS is an adaptation of MACA, requiring two radios for each station: One low-power, low-throughput signaling radio and one high-throughput, high-power radio used for data transfer. The signaling radio is used to send request-to-send (RTS) and clear-to-send (CTS) packets. Each station may leave it's high-throughput radio off until another station sends it an RTS packet.

B. Wireless TDMA Protocols

In some situations throughput may be the first thing we sacrifice. However, some applications may require reduced power consumption while maintaining high throughput, and the second radio required by techniques like PAMAS may be impractical. Time-division multiple-access media access protocols fit these requirements nicely. By requiring pre-scheduled channel access, stations may switch their receivers off while it isn't their turn to participate. This takes advantage of the fact that if n stations are trying to send data to an access point over a shared channel, each station will only receive on average $\frac{1}{n}$ of the bandwidth, and will not be doing anything useful the other $\frac{n-1}{n}$ of the time. If the MAC protocol allows the clients to leave their radios off for that $\frac{n-1}{n}$ of the time, they are likely to save a considerable amount of energy.

Sharma et al. introduce Wireless Rether [7], an 802.11 adaptation of Rether [8], which is, in turn, a token ring protocol for ethernet. They determined that token passing directly from one client to the next was unreliable in a wireless network, so they used a central controller to broadcast tokens. The purpose of their protocol was to enforce QoS, rather than conserve power. Their design required a token to be broadcast between each timeslot. Also, stations may complete their transmissions and terminate their time slots before they expire. Unfortunately, the other stations must always be awake because they could receive the next token at any time.

E2Mac [9] and ECMac [10] are proposed reservation-based TDMA MACs. Similar protocols have been used in commercial products, such as MMDS, wireless DOCSIS, and 802.16. In these protocols, all stations within a wireless network connect to a single base station that decides traffic schedules and broadcasts that information back to its client stations. Since stations know their traffic schedules in advance, they may leave their radios off most of the time. In most respects, our TDMA protocol is similar to these protocols but may be implemented using inexpensive, unmodified commodity wireless hardware such as 802.11.

C. Why not use PCF?

The 802.11 [4] standard includes an optional protocol extension called the Point-Coordinated Function (PCF) [4, sections 9.3 and 11.2.1.5]. Similar to Wireless Rether, the access point polls each client station in some order, inquiring if each has data to send. This is in contrast to 802.11's default mode, Distributed Coordination Function (DCF), in which each station competes for the channel using random backoffs. Since PCF client stations do not know when they will be polled by the AP, they must leave their radios on at all times, unless they are in Power-Save Mode.

Similarly to DCF PSM, the AP buffers all traffic to PCF PSM clients, and notifies them by periodically broadcasting a TIM. Unlike DCF, a PSM client with data to send cannot quickly gain access to the channel, because the AP is not allowed to poll a sleeping client. Fortunately, the client will not have to wait forever, since a PCF network periodically enters a contention period, where all the usual DCF rules apply.[1] If we were to use PCF with PSM, only traffic from the AP to the clients would benefit from PSM. In a video sensor network, however, most of the traffic would be from the clients to the AP, and only be transmitted during the contention period, leaving the contention-free period underutilized.

III. 802.11 With Time-Division Multiple-Access Protocol Description

To address the inefficient use of power by 802.11 due to idle listening and contention, we propose a time division multiple access protocol that may be run on unmodified commodity 802.11 hardware. In our protocol, there is a controlling station and a group of clients, all within range of the controlling station. The controlling station periodically transmits a broadcast packet containing start times and durations of time slots for each of its clients (hereafter referred to as a Schedule Frame or SF). A client may send or receive packets only during its time slot. Contention between the controller and the client is handled by 802.11's regular CSMA/CA algorithm.

[1] Alternating a contention period with a contention free period also allows stations to communicate even when they don't directly support PCF.

Outside their timeslots, clients may turn their radios off to conserve power. Any traffic between clients must be buffered and retransmitted by the controller since clients will not be on at the same time. For this reason, it is usually most convenient to make the AP act as controller and gateway.

The AP has the option of dynamically tuning the length of each client's timeslot. If a client uses its whole time slot, the controller could give it a larger slot next time. If a client's traffic volume is light, it could give it a smaller time slot. There is also an optional idle time slot, during which no client is allowed to transmit. This allows the controlling station to turn off its receiver when its available power is low or the network is being utilized at less than capacity.

All time slots advertised in the SF are listed in units of time relative to when the schedule frame was sent. Thus, there is no need for clients to synchronize their clocks to any absolute time (if the propagation delay is negligible relative to time slot length). The sum of the time slot lengths yields the time until the next schedule frame. Each client must turn its radio back on before then. If, for some reason, it misses the SF, it need only leave its receiver on and wait for the next one.

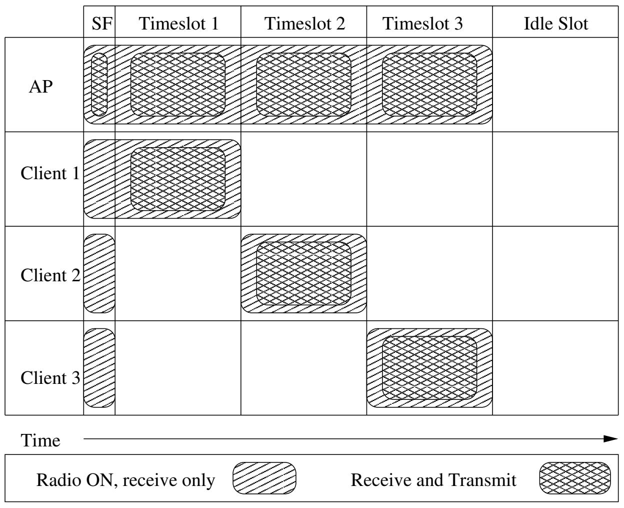


Fig. 1. Example timeslot allocation for a controller and three clients.

A. When to Transmit Scheduling Information

Some additional power and channel time is consumed by the schedule frame. There are many options of when and how often to send this frame, and how far in advance the schedule should be determined. One option is to send a SF prior to each time slot containing the duration of the single slot and the identifier of the station that should use the channel. This approach gives the controller maximum flexibility, because it only has to make decisions one time slot in advance. This approach has two disadvantages, however: every station must wake up after every time slot to find out if they're next, and a large proportion of time and energy is consumed by SFs. As we mentioned earlier, this approach is used by the PCF mode of 802.11 [4] and Wireless Rether [7].

Another option is to send a SF once every n time slots, containing the schedule for the next n time slots. A reasonable choice of n might be the number of clients, so an SF would contain the schedule for a complete cycle (assuming a round-robin scheduler).

In some cases, it may be advantageous for a SF containing scheduling data for n time slots to be broadcast several times every n time slots instead of once to increase reliability[2] or to reduce the number of radio state changes required on each client (for instance, by ignoring SFs not sent adjacent to its timeslots when it knows it will have some future timeslot adjacent to a SF).

When deciding which method to use, there is a trade off between sleep time and flexibility to be considered. If the scheduler broadcasts a schedule a long time in advance, stations will have the opportunity to turn their receivers off for a long time. However, the scheduler is stuck with its decision. If a station's bandwidth requirements increase dramatically in the short term, it will accumulate a large backlog of data, thus increasing latency. On the other hand, if a station's bandwidth requirements drop suddenly, its time slot will go underutilized. In addition to wasted bandwidth, the client and receiver will both waste power by leaving their receivers on longer than necessary. Scheduling too far into the future to save power may be self-defeating if the bandwidth requirements change too quickly for the controller to react.

There is a similar trade off when choosing time slot lengths. Long time slots minimize the schedule frame overhead, minimize the number of state changes, and ammortize the effects of inaccurate timing. However, they also add considerable communications latency.

B. Connection Requests

To allow clients to come and go as they please, we added a simple connection request mechanism: if a client has data to send, but the last SF doesn't contain a slot for that client, it will transmit a connection request that includes a station identifier and a number of timeslots. This request can be sent at any time (except the idle slot), and the 802.11 MAC will prevent collisions. By requesting a specific number of timeslots, the controller will eventually stop allocating a slot for that client if it turns itself off for a long period of time. Alternatively, the client may politely leave the network by requesting zero timeslots.

C. Implementation

For our experiments, we used an 802.11b network in infrastructure mode. All stations were personal computers running a Linux 2.4 kernel with Prism 2.5 based 802.11b cards. The controller used the HostAP [11] wireless ethernet driver in *Master* mode (enabling it to function as an access point). The other three used HostAP in *Managed* mode (allowing them to act as clients).

[2]Remember that the schedule frame is broadcast. In 802.11, broadcast packets are considerably less reliable than unicast packets because they do not employ immediate acknowledgment and retransmission.

In video sensor networks, it may be acceptable for data transmission to lag a second or two behind real time. In addition, we can buffer out short-term variations in available throughput.[3]. Therefore, for our purposes, we can use long time slots of hundreds of milliseconds or even seconds.

Implementation of the protocol requires a method of transmitting and receiving schedule frames, and a method to control the radio. The first is easily accomplished with a simple user space daemon on the controller and each client. The second, controlling radio power states and data transmission times, is more difficult.

On each client, the radio has three states: *on* (Tx and Rx allowed), *sleeping* (no Tx or Rx), and *receive-only* (Rx but no Tx). On is the common case, nothing unusual needs to be done. Setting the radio to the sleeping state is easily accomplished with the Linux wireless tools [4]. Receive-only is harder to implement, as there is no easy way to prevent a network interface that is up from transmitting.

We tried several techniques to prevent a client from sending. The first was purely at the application layer: the client application checks with a network scheduling daemon to see if it is allowed to send before writing to its socket. This approach, while simple, suffers from low accuracy. The operating system buffers a great deal of data, resulting in a significant lag between when the client stops writing and the network stops sending, typically in the range of 50-100 milliseconds. This lag may be reduced somewhat by limiting the size of the application's socket buffer, and by limiting the network device's packet queue.

Reducing the device's packet queue can result in unexpected behavior, though. In Linux, the kernel maintains a socket buffer for each TCP or UDP socket, and a queue of packets for each device. The TCP implementation keeps the socket buffer separate from the device queue. The UDP implementation, however, does not maintain a socket buffer independent of the device queue. When an application writes to a UDP socket, new packets are appended immediately to the device queue. If there is no room on the queue, the data is discarded and the OS returns control to the sending application as though the send had succeeded. Though this behavior is somewhat controversial [12], it has been allowed to persist. To avoid this problem, the socket buffer ought to be smaller than the device's packet queue so the program blocks rather than write large volumes of data that will never be sent.

A second alternative is to use a traffic shaper, [5] and manually switch the allowed throughput back and forth between zero and infinity. Unfortunately, none of the traffic shapers we tried reacted quickly enough to configuration changes, often requiring several seconds or more before changes took effect.

[3]Our current software[1] compresses video frames from a USB webcam as jpegs, which can be dropped in any order when throughput is limited. Currently, the USB link is the bottleneck, allowing us to generate usable video with a bit-rate of approximately 1mbps.

[4]iwconfig wlan0 txpower off

[5]In the kernel and associated routing documentation, these are called queuing disciplines, or qdiscs.

As a third option, we tried modifying the network stack directly. We added an extra bit to the network device's state indicating whether transmission was allowed or not. This bit was toggled via a system call. The network scheduling code was modified to check this bit before handing a packet off to the network device's driver. This option performed better than the traffic shaper, but did not give us quite the precision we hoped for. In effect, it performed similarly to the pure application-layer solution. Our current hypothesis is that the network device contains a large buffer that does not drain immediately.

The controller requires a more elaborate mechanism than the client, as it must be able to selectively transmit to one client at a time. This can be done in the application layer, as well. An in-kernel solution is likely to require a custom traffic shaper.

Another limitation on timeslot accuracy is timer latency. Timers in linux 2.4 are accurate to within 10 milliseconds in ideal situations, but under heavy load can be much worse. This can be improved by using a higher interrupt timer frequency or using a high precision timing mechanism [13]. In our experiments, 10 millisecond average latency was tolerable, but it could be a problem if timeslots are much shorter.

IV. Experiments

In the following experiments, we configured three Linux clients with 802.11 cards to send data at their maximum rate to the access point for thirty seconds. Data throughput (not including headers) was measured at the AP.

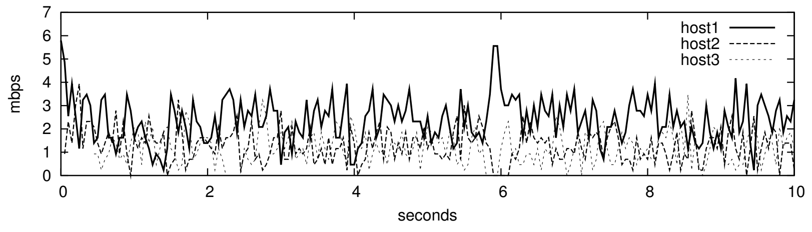
A. Standard 802.11b Experiments

We measured the UDP and TCP throughput of the three clients in figures 2 and 3, respectively. Initially, we were surprised by the low throughput of 802.11b networks. It turns out that 802.11 headers, immediate acknowledgments, and interframe spacing impose a high per-packet cost on both the data packets and, in the case of TCP, TCP ACKs. For instance, one way TCP throughput can be no more than 5.78 mbps, and in practice it is somewhat less, due to random back off, lost packets, and hardware, firmware, and driver inefficiencies. The highest average TCP throughput we have observed thus far is 5.29 mbps. This is similar to the results of others [14].

Fairness is not well achieved by either of these tests. It is not guaranteed by 802.11, nor by UDP. 802.11b's unpredictable round trip time seems to interfere with TCP's congestion window size calculations, preventing competing stations from converging on a fair proportion of available throughput.

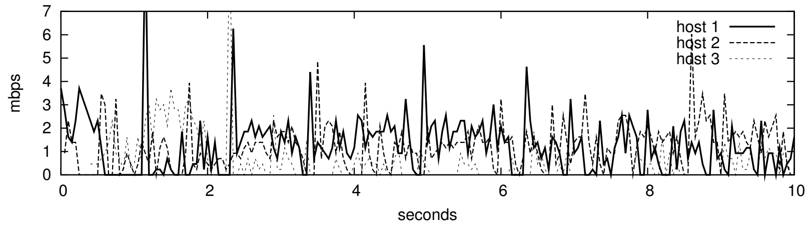
B. TDMA Experiments

For the next set of experiments, we used our TDMA protocol, with a simple round-robbin schedule. A SF is sent every third timeslot, containing a schedule for the next three timeslots. These were sent as broadcast UDP packets, and a process on each client received the SF and enabled or disabled transmission at the appropriate times. In these experiments, we controlled transmission purely from user space. In our UDP



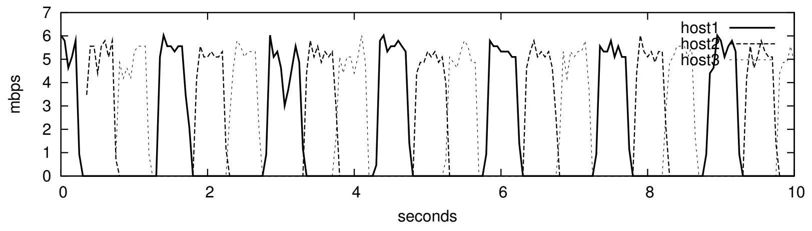
Host	Throughput (mbps)
1	2.216
2	1.500
3	1.085
Total	4.802
Standard deviation	0.572

Fig. 2. UDP Throughput. Host 1 receives a larger proportion of the network capacity than the others.



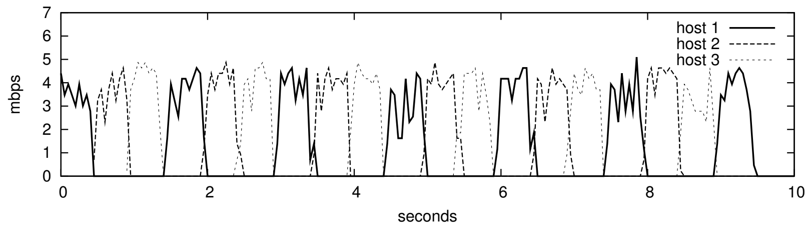
Host	Throughput (mbps)
1	1.432
2	1.345
3	0.422
Total	3.200
Standard deviation	0.560

Fig. 3. TCP throughput. TCP ACKs degrade throughput somewhat. Traffic is burstier, likely due to an erratic round-trip-time estimate.



Host	Throughput (mbps)
1	1.507
2	1.447
3	1.394
Total	4.347
Standard deviation	0.057

Fig. 4. UDP TDMA throughput with 12kB buffer. Throughput is close to regular UDP, with greater fairness.



Host	Throughput (mbps)
1	1.182
2	1.153
3	1.033
Total	3.368
Standard deviation	0.079

Fig. 5. TCP TDMA throughput with 20kB buffer. There is some minor overlap between timeslots.

Host	Throughput (mbps)	Schedule Frames	SF miss rate	Receiver On (seconds)	Receiver Off (seconds)	Receiver On Ratio
Controller	3.671	405 sent	n/a	Always	Never	1.00
1	1.233	373 received	0.0791	242.959	369.447	0.3967
2	1.180	361 received	0.1086	255.413	356.834	0.4172
3	1.258	374 received	0.0765	238.594	372.680	0.3903

Fig. 6. Throughput and receiver utilization of TCP traffic for approximately 10 minutes.

tests, the application stopped calling *send* 100 milliseconds before the end of the timeslot, to allow any buffered packets to drain. For TCP, we used 50 milliseconds. In addition, in both tests, the application limited the size of its socket buffer. In the UDP case (figure 4), we set the device's packet queue to hold at least as many packets as would fit in the socket buffer (set to 12kB), to prevent the kernel from discarding packets. In the TCP case (figure 5), the kernel does not discard data, so we set the device queue as small as we could without affecting performance. We found 6 packets, with a 20kB buffer, to work well. In both cases, timeslots were fixed at 500 milliseconds with no idle time slot.

We have not yet measured power savings directly. However, we have instrumented our clients to measure the amount of time they spend in the *on* or *receive-only* states versus the *sleeping* state, and count the number of schedule frames they receive. The controller was also instrumented to count the number of schedule frames it sent. We ran a TCP throughput test for approximately 10 minutes. The results are shown in figure 6.

In this test, there were a fair number of missed schedule frames. Station 2, which missed the most schedule frames, was also on the most and had the lowest throughput. This is an expected result, since a station that misses a schedule frame must stay on, without transmitting data, until the next schedule frame arrives. All stations were able to turn their radios off more than half of the time while maintaining throughput comparable to unmodified 802.11. From these results, it is reasonable to conclude that our protocol would achieve considerable power savings over an 802.11b network without power-save mode, in which all stations leave their receivers on at all times.

V. Conclusions and Future Work

We have shown that our TDMA protocol can achieve substantial power savings and improved fairness compared to standard 802.11b, without significantly affecting throughput, and can be implemented with relative ease. These gains come at the cost of high communications latency.

Lower communications latency will require smaller, more accurately controlled time slots. The main obstacle to achieving accurate timeslot enforcement is the difficulty of abruptly stopping transmission.

Our current implementation is limited to a one-hop wireless network. Larger multi-hop networks could be built of overlapping single-hop clusters. Since our protocol falls back on 802.11 for collision avoidance, it is not strictly necessary to ensure that nodes in multiple clusters do not have overlapping schedules. Other techniques such as GAF[15] and CEC[16] identify nodes with similar sets of in-range nodes, which take turns switching off to reduce redundancy while maintaining connectivity at all times (if possible).

References

[1] Wu chi Feng, Brian Code, Ed Kaiser, Mike Shea, Wu chang Feng, and Louis Bavoil. Panoptes: scalable low-power video sensor networking technologies. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 562–571. ACM Press, 2003.

[2] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *IEEE INFOCOM*, 2001.

[3] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks, 2002.

[4] IEEE Computer Society. *ANSI/IEEE Standard 802.11*, 1999.

[5] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 176–189. ACM Press, 2003.

[6] Suresh Singh and C.S. Raghavendra. Pamas: Power aware multi-access protocol with signalling for ad hoc networks. In *(to appear) ACM ComputerCommunications Review*, 1999.

[7] Srikant Sharma, Kartik Gopalan, Ningning Zhu, Gang Peng, Pradipta De, and Tzi cker Chiueh. Implementation experiences of bandwidth guarantee on a wireless LAN. In *ACM/SPIE Multimedia Computing and Networking (MMCN 02)*, 2002.

[8] Tzi cker Chiueh and Chitra Venkatramani. Design, implementation, and evaluation of a software-based real-time ethernet protocol. *ACM SIGCOMM*, 1995.

[9] Paul J.M. Havinga and Gerard J.M. Smit. Emac: an energy efficient mac protocol for multimedia traffic.

[10] Krishna M. Sivalingam, Jyh-Cheng Chen, Prathima Agrawal, and Mani B. Srivastava. Design and analysis of low-power access protocols for wireless and mobile ATM networks. *Wireless Networks*, 6(1):73–87, 2000.

[11] Jouni Malinen. Host ap driver. http://hostap.epitest.fi.

[12] Chris Friesen. Want opinions on possible glitch in 2.4 network error reporting. Linux Kernel Mailing List, February 6, 2002, http://www.ussg.iu.edu/hypermail/linux/kernel/0202.0/1175.html.

[13] Ashvin Goel, Luca Abeni, Charles Krasic, Jim Snow, and Jonathan Walpole. Supporting time-sensitive applications on general-purpose operating systems. In *Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, December 2002.

[14] James C Chen and Jeffrey M Gilbert. Measured performance of 5-ghz 802.11a wireless lan systems. Atheros Communications, http://www.atheros.com/pt/papers.html.

[15] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 70–84, Rome, Italy, July 2001. USC/Information Sciences Institute, ACM.

[16] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin. Topology control protocols to conserve energy in wireless ad hoc networks, 2003.