Ad hoc Routing for Multilevel Power Save Protocols

Technical Report

August 2006

Matthew J. Miller
Department of Computer Science, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
mjmille2@uiuc.edu

Nitin H. Vaidya
Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
nhv@uiuc.edu

***Abstract*— Designing energy efficient protocols for ad hoc networks is important since there has been little improvement in the amount of energy stored on these devices. Previous work [1]–[4] considers leaving a subset of nodes in a state with high energy consumption and low latency while the rest of the network remains in a power save state (i.e., low energy consumption and high latency). Our work is the first to generalize this concept for ad hoc networks by proposing the use of k levels of power save, each of which presents a different energy-latency tradeoff (i.e., a lower latency state requires more energy consumption). Thus, previous work only considered the case where $k = 1$ or $k = 2$. In this paper, we propose a link layer protocol to provide k levels of power save and a routing protocol to use this link layer effectively. Via simulation, we show that our protocols are able to maintain a desired end-to-end latency with a relatively low energy consumption.**

I. Introduction

Reducing energy consumption is important for wireless devices since they may need to operate for long periods on battery power. Unfortunately, the energy density of batteries has shown little improvement recently when compared to other performance metrics [5] (e.g., memory, disk storage, computation speed, and channel bitrate). This trend is further exacerbated as devices become smaller since there is less physical area available for a battery. Thus, given that the *amount* of energy stored is increasing rather slowly, it is beneficial to consider how to reduce the *rate* at which energy is consumed. This necessitates the need for *energy-efficient protocols* to balance how much energy the hardware consumes with acceptable performance for applications.

A complete solution to energy efficiency involves many areas of research, such as hardware, operating systems, networking, and applications [6]. Our work focuses on the networking component since it has been shown to be a significant power sink in devices with small or no displays [7] (e.g., sensors, cell phones).

Most work in this realm has been fairly restricted to homogeneous protocols in the sense that all nodes in a network use the same power save protocol (e.g., power save is either turned on or off for all nodes). Generally, turning power save on will consume less energy, but degrade the latency and throughput in the network when compared to switching it off. However, some work considers the scenario where a small subset of nodes turns power save off while the rest of the nodes remain in the power save on state [1]–[4]. How this subset of nodes is chosen differentiates these protocols and is discussed in Section II.

This work is supported in part by a NSF fellowship.

TABLE I
ENERGY CONSUMPTION AND LATENCY OF BIMODAL POWER SAVE STATES.

State	Energy Consumption	Latency
Power Save	Low	High
No Power Save (Always On)	High	Low

One common characteristic of all such ad hoc network power save protocols is that they are bimodal. That is all nodes are in one of the two states shown in Table I. The contribution of this work is to generalize the idea of heterogeneous power save protocols to support *multiple* power save states. In our work, each node uses one of k levels of power save at any given time (thus, previous work only focused on the $k = 1$ and $k = 2$ cases). While the idea of multilevel power save has been proposed for single hop networks with a base station [8], we are unaware of any comparable work for ad hoc networks. Obviously, these scenarios differ greatly since the latter requires distributed protocols as opposed to the centralized approach of the former. Additionally, the approach in [8] uses a separate out-of-band channel whereas our work only requires one channel.

In Section II, we survey related work. In Section III we describe our design in two parts: the link layer protocol (Section III-A) provides multilevel power save and the routing protocol (Section III-B) uses this link layer effectively. We evaluate our protocol via simulation in Section IV. Section V proposes some extensions to our scheme and Section VI concludes the paper.

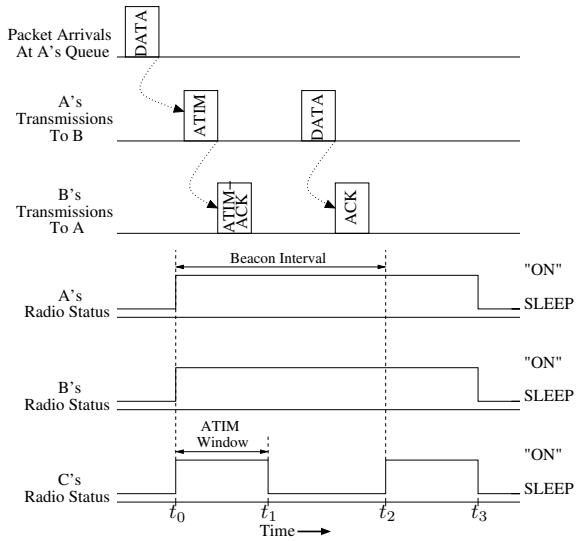

Fig. 1. IEEE 802.11 Power Save Mode (PSM) [9].

II. Related Work

We begin by describing IEEE 802.11 Power Save Mode (PSM) [9]. This protocol serves as the foundation for our protocol and much of the related work. It has a simple design and the most complete specification of any open standard power save protocol. Nodes are assumed to be synchronized and awake at the beginning of each *beacon interval (BI)*. After waking up, each node stays on for a period of time called the *Ad hoc Traffic Indication Message (ATIM) window*. During the ATIM window, since all nodes are guaranteed to be listening, packets that have been queued since the previous beacon interval are advertised via ATIM packets. When a node has a packet to advertise, it sends an ATIM packet to the intended receiver during the ATIM window. In response to receiving an ATIM packet, the destination will respond with an ATIM-ACK packet (unless the ATIM specified a broadcast destination address). When this ATIM handshake has occurred, both nodes will remain on after the ATIM window and try to send their advertised data packets before the next beacon interval. If a node remains on after the ATIM window, it must keep its radio on until the next beacon interval. If a node does not send or receive an ATIM, it will enter sleep mode at the end of the ATIM window until the next beacon interval. This process is illustrated in Figure 1. The dotted arrows indicate events that cause other events to occur. Node **A** sends a data packet to **B**, while **C**, not receiving any ATIM packets, returns to sleep for the rest of the beacon interval.

The protocol in [1] works with on-demand routing and uses 802.11's PSM when a node is *not* engaged in sending, receiving, or forwarding data. When a node is communicating, soft-timers are used to transition the node to an idle listening mode that reduces latency and preserves throughput better than using only 802.11's power save. However, the timers do not adjust to the traffic rate, so if traffic is not frequent enough to refresh the timers, then the benefits of the protocol are lost. TITAN [2] extends the work from [1]. In TITAN, route requests are delayed by sleeping nodes to allow the route discovery procedure to favor nodes that are already in the idle listening state. This helps reduce the overall energy consumption in the network. Both of these protocols only consider two power save levels whereas our work is designed for the more general scenario of k power save levels.

Another strategy is for nodes to remain awake based on their local topology and/or traffic [3], [4]. GAF [4] assumes the nodes have some location information and form virtual grids. The size of the grids is chosen such that the nodes in two adjacent grids are equivalent with respect to forwarding packets. Then, within each grid, a discovery protocol tries to ensure that most of the time one node remains active while the rest enter a low-power state. As mobility increases, the discovery process should be more frequent. SPAN [3] allows all nodes to enter power save mode except for elected *coordinators*. At the MAC layer, nodes periodically exchange messages that contain its set of neighbors, coordinators, and whether it is a coordinator. Nodes will then elect themselves coordinators if their neighbors would get better connectivity by it doing so. A random delay is introduced before nodes declare themselves coordinators. This delay varies inversely with the amount of connectivity that would be achieved and inversely with the amount of energy remaining at the node. For fairness, the coordinators will periodically withdraw. These protocols only consider two power save levels.

As we mentioned in Section I, all of the work mentioned above only places the nodes in one of two power save states. By contrast, our work places nodes in one of k power save states. In [8], a similar idea is explored in the context of single hop networks with a base station. Here, devices have a paging interface that is used by the base station to wake up certain nodes when it has data to send. The devices can be in any one of several sleep states. Each sleep state uses less power in steady state, but requires more delay and power when transitioning to the fully awake state. A device will remain in a power save state at least long enough to get a positive energy gain before transitioning to the next lower power state. The base station tracks this cycle for each device and when it has data to send, it waits as long as possible before waking the device and transmitting subject to QoS requirements. When the base station wishes to wake a device up, it pages all devices in that current sleep state. The non-target devices in the paged sleep state will then start the sleep cycle again once they determine that the data is not for them. This allows the size of the paging message to be on the order of the number of sleep states instead of the number of nodes.

III. Protocol Design

Our goal is to design a routing protocol for networks that use k levels of power save protocols. Each level of power save provides a different energy-latency tradeoff (i.e., a level with a lower latency requires more energy). As mentioned earlier, this paradigm is a generalization of the paradigm in [1]–[4] (discussed in Section II) where only two levels of

power save are assumed (the levels shown in Table I). By using multiple power save levels, we allow applications (e.g., sensor reports) to achieve an acceptable latency while reducing energy consumption in the network.

The idea of using multilevel design to achieve acceptable tradeoffs is prevalent in computer science (see [10] and references therein). For example, in computer architecture, accessing cache is much faster than main memory. However, main memory is cheaper in terms of cost per byte and is capable of storing much more data.

A. Link Layer Protocol

First, we need to specify how the link layer power save protocols can be designed to provide k levels of power save, each with different energy-latency characteristics. Many power save protocols can be adapted to achieve this as discussed later in this section. We use 802.11 PSM [9] as the underlying power save protocol, which is described in detail in Section II.

The 802.11 PSM protocol can be adapted to provide k levels of power save by changing how frequently a node wakes up to listen during an ATIM window based on its current power save level. We denote these k power save levels as $PS_0, \ldots, PS_{k-1}$. Without loss of generality, we assume that PS_0 corresponds to the "always on" state and PS_{k-1} uses the least amount of energy, but has the highest latency. In PS_0, the nodes never sleep and, thus, can receive a packet with the lowest latency, but also consume the most energy. The next level, PS_1 corresponds to the standard implementation of 802.11 PSM. That is, when a node is not sending or receiving any packets, it wakes up for every ATIM window and sleeps for the remainder of the beacon interval. In PS_2, nodes wake up only every other ATIM window. This allows them to save about twice as much energy as the nodes in level PS_1 while also doubling the latency to send or receive a packet.

Because we want to ensure that every node has their ATIM overlap with every other node periodically, we increase the sleep time for each level by a factor of two. This is a simple method to guarantee overlap, but more complicated schedules [11] may work as well. Thus, to calculate the beacon interval for level PS_i, we have:

$$BI_i = 2^{i-1} \times BI_{base} \quad \text{, when } i > 0 \tag{1}$$

where BI_i is the beacon interval for the i-th power save level and BI_{base} is the base beacon interval specified for the system (i.e., $BI_1 = BI_{base}$).

Figure 2 illustrates the multilevel link layer protocol with $k = 4$. In this figure, AW corresponds to the ATIM window size and we show only the case in which no traffic is being sent. The beacon intervals of the four power save levels are: $BI_0 = 0$, $BI_1 = t_1 - t_0$, $BI_2 = t_2 - t_0$, and $BI_3 = t_4 - t_0$. The base interval is t_1 (i.e., $BI_{base} = t_1$).

The largest possible beacon interval, BI_{k-1}, serves as the reference point for all of the nodes to ensure that they remain in phase. That is, the first ATIM window for which a node awakes in a cycle must always occur at the beginning of a reference point beacon interval (that is spaced BI_{k-1} time units after the previous reference point). The reference points in Figure 2 are at t_0 and t_4.

Since we assume that the nodes are synchronized, each node is initialized with the time of the previous reference point. Alternatively, if a node is added to the network later, it can learn the time of the previous reference point from older nodes in the network, along with the ATIM window size, BI_{base}, and the number of power levels the network is using via 802.11 management frames. This guarantees that for any two nodes, one with PS_i and the other with power level PS_j where $i < j$, the node with PS_i will be awake during every ATIM interval that the node with PS_j is awake since BI_j is divisible by BI_i.

Each node keeps track of its neighbors' power save state as follows. On every data and ACK packet a node sends, it attaches its current power save level. We do not test the consistency of a node's power save table. However, our protocol could use a scheme similar to the one in [1] whereby the first time a packet transmission fails, a node sets the intended receiver's power save state to PS_{k-1}. Recall that PS_{k-1} has the longest beacon interval and all nodes, no matter what power state, are guaranteed to wake up every BI_{k-1} time units. Thus, if the neighbor still exists near the node, communication should be possible during this beacon interval. If a transmission fails again for the receiver using PS_{k-1}, then the link is considered dead and reported to upper layers.

This is just one example of how a power save protocol can be modified to achieve multiple levels with different energy-latency tradeoffs. Other examples include adjusting the time between listening periods in protocols such as STEM [12] and WiseMAC [13]. Nodes using a longer sleeping time between listening periods would save more energy, but require a longer latency to be awakened by neighbors.

B. Routing Protocol Description

In Section III-A, we described how to *provide* multilevel power save. In this section, we describe a routing protocol to *efficiently use* multilevel power save. If energy consumption is the only concern, the optimal adaptive sleeping strategy is simply for every node to select PS_{k-1} as their power save state. However, this results in large delays due to the power save protocol that may be unacceptable for many applications.

Thus, our protocol works by taking an application-defined latency bound and trying to find a route to achieve the bound while attempting to minimize the increase in energy consumption. We focus on only the latency induced by the power save protocol because this delay tends to be large (e.g., hundreds of milliseconds or even seconds *per hop*) relative to contention and queuing delay in non-congested networks. In highly congested networks, power save protocols would most likely not be used. A vast body of QoS research deals with congestion and queuing delay which we view as orthogonal and complementary to our work.

If we are given a set of m flows to route $(F_1, F_2, \ldots, F_m)$ and a desired latency for each flow $(L_1, L_2, \ldots, L_m)$, finding routes that minimize the overall energy consumption increase

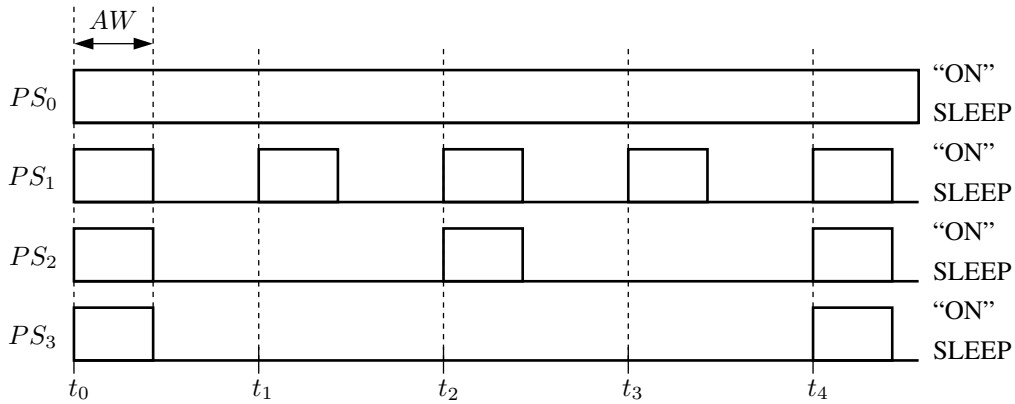

Fig. 2. Multilevel power save with 802.11 PSM [9].

for the flows is NP-complete. A proof of this is presented in Appendix I. In this work, therefore, we consider heuristics to address the problem.

We modify DSR (Dynamic Source Routing) [14] to obtain our routing protocol. We now give a brief overview of the salient aspects of DSR. When a source, S, wants to send packets to a destination, D, it must first discover a route. To do this, S broadcasts a route request packet ($RREQ$) that is flooded throughout the network specifying that it is trying to find a route to D. Each node, other than D, that receives S's $RREQ$ will add itself to a node list in the packet and rebroadcast the $RREQ$ (assuming that the TTL of the $RREQ$ has not expired).[1] Each $RREQ$ is rebroadcast only once by an intermediate node. So, if multiple copies of the same $RREQ$ are received by a node, as determined by a unique sequence number for the request, the node will forward only the first one that it receives. If the $RREQ$ reaches D, it generates a route reply ($RREP$) packet and sends it to the source.[2] The $RREP$ packet is generated by reversing the node list in the $RREQ$ and sending the $RREP$ along the path specified by the node list. The entire node list is included in the payload of the $RREP$ packet and is also used for source routing the packet to S. A node that receives a source-routed packet will only forward it if the node's ID is next on this node list. To do so, it transmits the packet to the next node ID specified on the list. In this manner, the $RREP$ makes its way back to S. At this point, S extracts the node list from the payload of the $RREP$ and uses it as the source route to forward data packets. That is, every data packet that S sends will have the node list appended to the routing header.

We modify DSR as follows. The $RREQ$ sender adds its desired latency, L, for the flow to the $RREQ$ packet. When forwarding the $RREQ$, each node will append its current power save state in addition to its node ID. When D receives its first $RREQ$ from S, it will set a timer for some specified time, T_{delay}.[3] D will not send a $RREP$ until this timer expires. While the timer is running, D will collect all $RREQ$s with the same sequence number that it receives from S. At the end of this T_{delay} time, D will evaluate all the paths that it has received from these $RREQ$s and send an $RREP$ along the "best" path. Next, we specify the routing metrics that are used to determine which path to use.

The goal of our routing metric is to find the path that can achieve the desired latency, L, (specified in the $RREQ$) while increasing the energy consumption in the network the least. To do this, we consider paths that have been collected during the $RREQ$ reception phase. For each path, we find the node on the path whose energy consumption will increase the least by moving to the next higher energy state (and, hence, lowering the latency for that hop). We continue iterating in this manner until the path's end-to-end power save-induced latency is less than L or all nodes are in the highest energy state. At this point, we store the total energy increase for the path that was necessary for the iteration to terminate. Once this has been done for all the paths, we send the $RREP$ on the path that requires the smallest total energy consumption increase. If two or more paths are tied for the minimum cost, then our protocol prefers routes with the lowest hop count.[4] Our algorithm is shown in Figures 3, 4, and 5.

Each node receiving the $RREP$ will check the requested power save level set for it by the destination. If the requested power save level is a higher energy level than its current level, then the node switches to the new power save level. Otherwise, it will remain in its current power save state since it is sufficient to maintain the desired latency of the path.

[1]Non-destination nodes replying to $RREQ$s using cached routes is one of *many* extensions that has been proposed for DSR. We do not use cached replies in our work.

[2]Another option in DSR is whether the destination replies to every $RREQ$ it receives or just the first one. In our protocol, the $RREP$ procedure is modified, but the destination will send only one $RREP$ per $RREQ$.

[3]Another option is that a node replies after receiving some number, say x, $RREQ$s even if the T_{delay} timer has not yet expired. For example, if $x = 1$, then a node would just calculate the power save state changes required for the path on the first $RREQ$ that it receives and use that path (and disregard all subsequent $RREQ$s for that route discovery). If $x = 2$, the node would consider only the first two $RREQ$s that it receives and cancel the T_{delay} timer if it has not yet expired.

[4]We note that other metrics such expected number of transmissions or packet loss [15] could be used instead.

```
FIND-ROUTE(R, L)
  /*****
   * Find route on which to send the RREP
   * given a list, R, of received RREQs
   * and latency threshold, L
   *****/

  isFirst ← true
  for each r in R
  do cost ← ENERGY-INCREASE(r, L)
      if isFirst or cost < min
         then min = cost
              minRREQ = r
              isFirst ← false

   /* Set requested power levels for chosen path */
  ARRAY-COPY(psLevels[minRREQ],
                 newPsLevels[minRREQ])

   /* Reply using the path from minRREQ */
  SEND-RREP(minRREQ)
```

Fig. 3. Algorithm for determining which path to use from collected $RREQ$s.

```
1  ENERGY-INCREASE(r, L)
2    /*****
3     * Find the minimum energy consumption increase
4     * required for the path in a RREP, r, to achieve
5     * a wake-up latency less than or equal to L.
6     *****/
7
8    /*****
9     * psLevels[r] contains the current power save level
10    * of each node along the path in r.
11    * psLevels[r] is an array with an element for
12    * each node on the path.
13    *****/
14
15   /* pathLen[r] is the length of the path in r */
16
17   ARRAY-COPY(newPsLevels[r], psLevels[r])
18   energyCost ← 0
19   /*****
20    * We assume that PATH-LATENCY ≤ L
21    * when newPsLevels[r][i] = 0 for all i on the path
22    *****/
23   while PATH-LATENCY(r) > L
24   do isFirst ← true
25      for i ← 1 to pathLen[r]
26      do cost ← ENERGY-DIFF(newPsLevels[r][i],
27                            (newPsLevels[r][i] − 1))
28         if cost ≠ 0 and (isFirst or cost < min)
29            then min = cost
30                 minIndex = i
31                 isFirst ← false
32      energyCost ← energyCost + min
33      newPsLevels[r][minIndex] ←
34                 newPsLevels[r][minIndex] − 1
35   return energyCost
```

Fig. 4. Algorithm for computing cost of a path to reach latency threshold L.

The FIND-ROUTE function in Figure 3 finds the route to use based on R, the set of $RREQ$s that have been collected. For each $RREQ$, FIND-ROUTE calls ENERGY-INCREASE (discussed below) to calculate the cost of using the $RREQ$'s route in terms of how much the energy consumption of the path must be increased to reach the latency threshold, L. At the end of the **for** loop, the least costly path is found and the power save states are set to the new power levels necessary to achieve the latency threshold ($newPsLevels$ is a global variable set in ENERGY-INCREASE). With these updated power levels, the $RREP$ is constructed and sent along the chosen path via the call to SEND-RREP.

The ENERGY-INCREASE function in Figure 4 computes the minimum increase in energy consumption necessary for the path in a $RREQ$, r, to achieve the desired latency, L. First, the function makes a copy of the power save levels of the nodes in r's path ($psLevels[r]$) since our algorithm needs to change this state. The $energyCost$ variable keeps a running total of the increase in energy consumption required for r's path to reach L. The **while** loop on line 22 will continue until the latency of the path is less than L (we assume that this will always terminate in the pseudocode). Each iteration of the **while** loop will calculate the difference in energy consumption that would result for each node in r's path if its current power save state was moved to the next lower latency power save state (i.e., moving from PS_i to PS_{i-1}). This calculation is done via the call to ENERGY-DIFF, which is discussed below. Once the **for** loop on line 24 has terminated, we have identified the node on r's path who can transition to a lower latency power save state with the smallest increase in energy consumption. At this point, we transition to the lower latency power save state (using the $newPsLevels$ variable) and increment $energyCost$ by the energy consumption increase required. When the path latency (calculated by the PATH-LATENCY function call) is less than L, the **while** loop terminates and returns $energyCost$.

The PATH-LATENCY function in Figure 4 (line 22) can be computed in terms of worst-case latency, average-case latency, or some other metric. We assume that a node j is using the k_j-th power level. Thus, PS_{k_j} denotes its power save level and BI_{k_j} is the length of its beacon interval. Thus, for a path of n nodes, and the worst-case latency metric, our protocol considers the route for use if:

$$BI_{k_1} + BI_{k_2} + \cdots + BI_{k_n} < L \tag{2}$$

The ENERGY-DIFF function in Figure 5 computes an energy cost for transitioning from one power save state to a lower latency power save state. We compute the energy consumption of a power save state as the ATIM window size ($atimSize$, whose value is set elsewhere) divided by the power save state's beacon interval size (i.e., BI_i). The $beaconIntervalSize$ variable is an array indexed by the beacon interval sizes for each power save state. Note that this energy consumption calculation considers only the energy consumption when nodes are not awake after the ATIM window. When nodes *are* awake following the ATIM window, the energy consumption used in

```
ENERGY-DIFF(oldLevel, newLevel)
/*****
 * Find the difference in energy consumption for
 * switching from the lower energy oldLevel
 * to the higher energy newLevel
 *****/

/*****
 * atimSize is a parameter specified elsewhere.
 * It is the size of the ATIM window in units of time.
 *****/

if oldLevel = 0 or newLevel > oldLevel
   then return 0

oldEnergy ← atimSize / beaconIntervalSize[oldLevel]
if newLevel > 0
   then newEnergy ← atimSize / beaconIntervalSize[newLevel]
   else newEnergy ← 1
return (newEnergy − oldEnergy)
```

Fig. 5. Algorithm for computing the energy consumption difference between two power save levels.

the subsequent beacon interval is the same regardless of the power save state. As an example, let $atimSize = 20\,\text{ms}$, and $BI_i = 200\,\text{ms}$ and $BI_{i-1} = 100\,\text{ms}$. In this case, ENERGY-DIFF(PS_i, PS_{i-1}) will return $\frac{20}{100} - \frac{20}{200} = 0.1$.

Though we do not test this in our simulations, each node must set a soft timer for each flow for which it forwards packets so that it can revert to lower energy states whenever that flow ceases or the route fails. Because the inter-arrival time for the packets on a flow is highly application dependent, we propose letting the application specify this timeout value and piggybacking it on data packets sent by the flow. Whenever a flow times out or explicitly indicates that it will no longer use the route, the node transitions into the lowest energy power save state that is still acceptable to the flows which continue to use that node on their route, as indicated by the power save levels specified for the node in RREPs that it has received.

C. Design Discussion

1) Wake-Up Schedules: As described in Section III-A, we use a simple link layer protocol to provide multiple levels of power save. Basically, the beacon interval either increases by a factor of two or decreases by half depending on whether the node is moving to a lower or higher energy state, respectively. An alternative is to use more complex wake-up schemes that provide overlap either deterministically or probabilistically.

In general, probabilistic protocols (e.g., [16]) are not appropriate in our design since they essentially add more uncertainty to an already unreliable channel. Additionally, these protocols make even soft real-time constraints more difficult to obtain. Thus, we do not consider probabilistic approaches for our protocol.

By contrast, protocols that give deterministic overlap in an asynchronous manner (e.g., [11]) do allow soft real-time latency bounds. The basic idea is that each node wakes up according to some pattern that is guaranteed to overlap within some bound with every other node even though they may be unsynchronized. The major advantage of this approach is that it makes synchronization less necessary. However, it can greatly increase the protocol complexity since the wake-up schedules have to be chosen appropriately and nodes still must probe to find out when the overlap occurs since they have no prior knowledge. Additionally, broadcast is a problem since there is no single time where a node is guaranteed to have all of its neighbors listening. We do not use a deterministic asynchronous protocol because we are not concerned with synchronization and we need a relatively reliable and low overhead broadcast mechanism for the route discovery in our work. This also frees us from the added complexity such a scheme would add to focus on the major idea of routing with multiple power save levels.

Another option is to have one, long "master" beacon interval in which everyone is awake (i.e., PS_{k-1} in our protocol). Then, each node chooses its own beacon interval independently based on the RREPs it receives and lets each communicating neighbor know the next time it is scheduled to awake. Nodes then keep track of the next wake-up time for each node with which they are communicating. This frees the nodes from the need to use specified discrete intervals and allows them to use any interval up to PS_{k-1}. Broadcast is still possible, as in our scheme, where broadcasts are sent only during the "master", or PS_{k-1}, interval. The disadvantage of this approach is that it requires the nodes to keep more per flow state. Also, it is more susceptible to nodes returning to sleep too early since nodes waking up experience contention from data packets, not just ATIM packets as in our scheme. Data packets tend to be significantly larger than ATIM packets. In future work, one could more fully explore this idea to see under what conditions the early sleep problem makes this protocol worse than our current version.

2) Soft Timers: As mentioned in Section III-B, we use soft timers per flow passing through a node to determine when it can revert to a lower energy state. We believe that this is acceptable since, in many environments, a node will have only a few flows passing through it. Of course, a node can always choose *not* to handle additional flows if its per flow state becomes excessive.

An alternative to this design decision is to require a sender to explicitly "delete" a flow by sending a packet along the path when it is finished. We feel that this method would be unacceptable in multihop wireless network settings due to the inherent underlying reliability of the channel and devices. Because links and flows can fail unexpectedly, a node would permanently keep state for dead flows for which the flow was not deleted. Eventually, this could exhaust the node's memory resources. Thus, overall, we feel that the per flow state required to maintains soft timers for this purpose is best for the environment we are considering.

3) Routing Techniques: We choose to use DSR [14], a source routing protocol, in our work as described in Section III-A. An alternative would be to use a distance vector

approach, like AODV [17]. The disadvantage of using AODV (or another distance vector protocol) is that nodes learn only aggregate information about the path during routing as opposed to DSR which provides *per node* information. In the algorithms discussed in Section III-A, we need per node information. In this aspect, DSR provides a superset of the information that AODV does. Because our algorithms do not work with the information from AODV, we use DSR in our work.

Another choice would be to use link state routing [18], such as OLSR [19]. Nodes could flood the network whenever their power save level changes or a link breaks. The obvious disadvantage of this approach is the high overhead to flood the network if power save states are changing relatively frequently. Also, as shown if Appendix I, even if the entire topology is accurately known, it is still NP-complete to find the minimal energy consumption increase required for a desired latency. Thus, the advantage of knowing the entire topology, as opposed DSR which learns just a few paths, is not easily exploited. At the very least, we could find the k shortest paths [20], given the entire topology, and run the algorithms from Section III-A on each of these paths. In future work, it would be interesting to test a link state routing protocol versus our DSR implementation to determine which performs better under different metric change frequencies and network sizes.

IV. Simulation Results

To evaluate our protocol, we simulated it using *ns-2* [21]. We test the following schemes, where the bold text is the name we use to refer to the scheme and the italicized text indicates the (Routing, MAC) tuple used:

- **Always On [9], [14]** *(DSR, 802.11)***:** This is the IEEE 802.11 protocol with no power save. It is the default, unmodified MAC protocol in *ns-2*. Because nodes never sleep, ALWAYS ON uses the most energy, but has the lowest latency.
- **802.11 PSM [9], [14]** *(DSR, 802.11 PSM)***:** This is the standard IEEE 802.11 protocol with power save enabled. 802.11 PSM is described in Section II. The beacon interval for this protocol is set to the longest beacon interval for a given k value.
- **CS-ATIM** *(DSR, CS-ATIM)***:** This is 802.11 PSM with our proposed carrier sensing modification described in [22]. The beacon interval for this protocol is set to the longest beacon interval for a given k value.
- **Multilevel PSM** *(Multilevel DSR, Multilevel 802.11 PSM)***:** This is our proposed multilevel power save protocol described in Section III using 802.11 PSM.
- **Multilevel CS-ATIM** *(Multilevel DSR, Multilevel CS-ATIM)***:** This is our proposed multilevel power save protocol described in Section III using the CS-ATIM protocol that we proposed in [22].

We use 2 Mbps radios that have a 250 m range. Each data point is averaged over 30 tests. The ATIM window is 20 ms and the base beacon interval, BI_{base}, is 100 ms. Our topologies are generated by placing 50 nodes uniformly at random in a 1000 m×1000 m area. Each scenario has five flows among randomly chosen source and destination pairs. Each flow sends at rate one packet per second using CBR traffic. We set T_{delay}, the time that a destination waits to collect $RREQ$s to be 500 ms. In our experiments, we set L to be the same value for all flows in the network and do not test the more general case where each flow could select its own L value.

TABLE II

Standard deviation as percentage of mean for latency figures (Average | Maximum).

	Figure 7		Figure 9		Figure 12	
Always On	29.06	29.06	25.99	25.99	29.00	29.00
802.11 PSM	33.77	52.54	29.39	29.39	56.94	56.94
Multilevel PSM	20.03	22.75	26.33	29.04	23.46	53.28
Multilevel CS-ATIM	17.61	19.43	24.86	35.56	22.04	61.39
CS-ATIM	36.06	52.01	26.94	26.94	43.72	43.72

Since our protocols are designed to only achieve soft real-time bounds on latency, it is important to consider the standard deviation of our latency results. This gives us an indication of how well the protocols are able to stay within the bounds over multiple runs. To avoid cluttering our figures with standard deviation bars, we provide the numerical values in Table II (we will refer to this table in our discussion of the results). In this table, we give the standard deviation for each protocol in each latency figure as a percentage of the mean for the corresponding data point. We use the percentage since the mean values can vary significantly which makes the absolute values of the standard deviations difficult to compare. We compute the standard deviation *averaged* over all data points for the protocol as well as the *maximum* standard deviation of any one data point on a protocol's curve. Additionally, we *have* plotted the standard deviation bars for the latency of the multilevel protocols to show their deviation relative to the desired latency bound.

Figure 6 shows energy consumption of the protocols when $L = 300$ ms. The horizontal axis is k, the maximum number of power save levels. Since $T_{base} = 100$ ms, $k = 2$ corresponds to the traditional 802.11 protocol where a node can either be on or using a power save protocol with a beacon interval of 100 ms. From the figure, we see that all the power save protocols use significantly less energy than the Always On protocol.

We see that the multilevel PSM protocol uses about 33% to 50% more energy than the traditional PSM protocol. However, this increase in energy comes with a huge reduction in latency as shown in Figure 7. In this figure, we measure only the latency for packets that are sent *after* the source has received the $RREP$. The source queues packets while waiting for the $RREP$, which makes their delay rather large and can skew the average end-to-end delay of the rest of the packets.

The multilevel protocols achieve a delay of around 140 ms to 180 ms, which is well within the $L = 300$ ms bound that was given. By contrast, the non-multilevel protocols have a

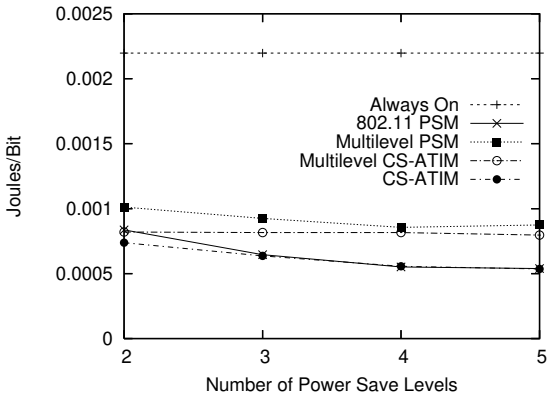

Fig. 6. Effects of the number of power save levels on energy.

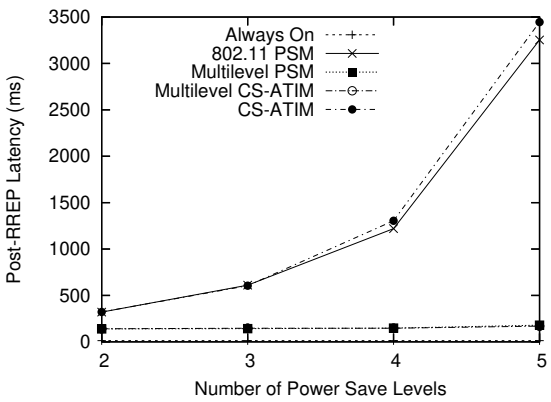

Fig. 7. Effects of the number of power save levels on latency.

latency of just over 300 ms when $k = 2$ and increase to over 3000 ms when $k = 5$. For $k = 3$ and $k = 4$, we notice that the average latency is approximately double that of using the next lower latency power save state (i.e., $k = 3$ latency is about double that of $k = 2$ and $k = 4$ is twice as much as $k = 3$). However, when $k = 5$, the latency more than doubles over that of $k = 4$. The reason for this is that ATIM window contention causes significant delays. Since the ATIM window size is static regardless of k and the traffic rate remains the same, more packets need to be advertised in the ATIM window when $k = 5$ as opposed to, say, $k = 2$. The increased contention reaches a point where some nodes are unable to send an ATIM when they first try and must wait another beacon interval. This greatly degrades latency since the beacon intervals are longer for larger values of k. When $k = 5$, *each hop* has a wake-up latency of 800 ms plus the increased ATIM contention. With the multilevel power save protocols, the routing protocol adjusts the power save level of nodes along a path to ensure that the latency is less than L.

Additionally, we can see from Table II that the multilevel protocols in Figure 7 have a lower deviation in their latency among different runs than the corresponding protocol without the multilevel extension. The multilevel protocols have a deviation of about 20% on average, whereas PSM and CS-ATIM without the multilevel extension have about a 35% deviation on average. This is due to the fact that a wider range of average latencies are possible in the non-multilevel protocols for different topologies and traffic patterns.

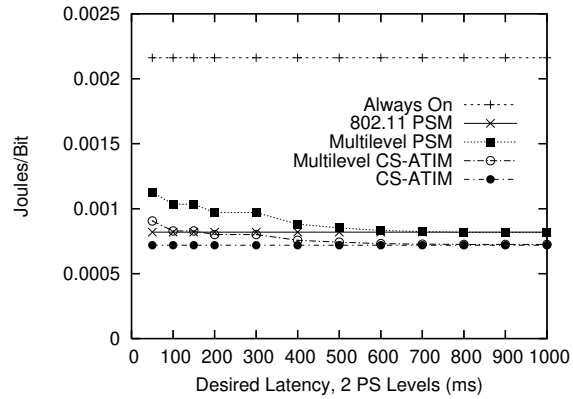

Fig. 8. Latency threshold versus energy consumption using two power save levels.

From Figure 6, we can see that our carrier sense techniques from [22] integrate nicely with the multilevel power save scheme. In particular, by using CS-ATIM, we are able to achieve virtually the same latency at using PSM (and well below the L threshold) while consuming less energy than the PSM version of multilevel power save. All of the protocols seem to plateau at a point where the utility of adding more power save levels diminishes. The multilevel CS-ATIM protocol seems to reach this plateau with only two power save levels and shows only a slight decrease in energy consumption after this point.

In Figure 8, Figure 9, and Figure 10, we set $k = 2$ and show the effects of changing L, the desired latency, on energy consumption and the observed latency, respectively. Again, we see that the multilevel power save protocols achieve the latency bound with only a slight increase in energy. In particular, we can see that, for $k = 2$, if a latency of less than about 300 ms is desired, then the power save protocols that do not use multilevel power save cannot achieve this. Without multilevel power save, the only option would be to turn off power save which, as we can see from Figure 8, substantially increases energy consumption by more than a factor of two. Furthermore, in Figure 10, we can see that virtually *none* of the individual runs exceed the latency bound when using the multilevel extension. We note that a few of the flows *do* exceed the latency bound by a small amount. This occurs because our protocol adjusts the *power save induced* latency and does not account for transmission times and queuing delays. Thus, our protocol occasionally sets the power save states such that they are close to or equal to L, but the extra delays make the observed latency slightly higher than L.

In Figure 11, Figure 12, and Figure 13, we show the effects of changing L for $k = 3$. We can see that the multilevel power save protocols use slightly more energy relative to the other power save protocols than for the $k = 2$ case. However, the

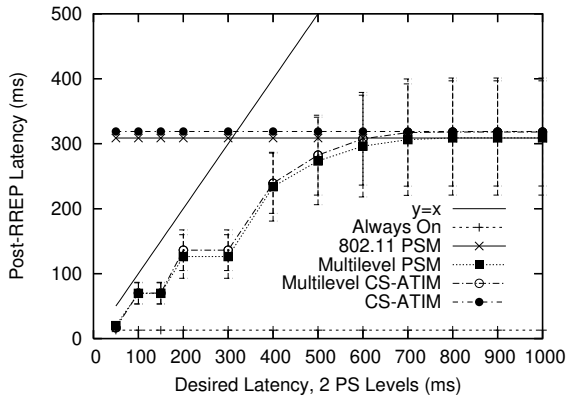

Fig. 9. Latency threshold versus observed latency using two power save levels.

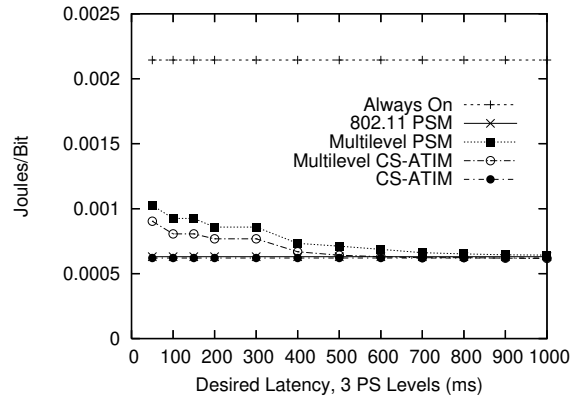

Fig. 11. Latency threshold versus energy consumption using three power save levels.

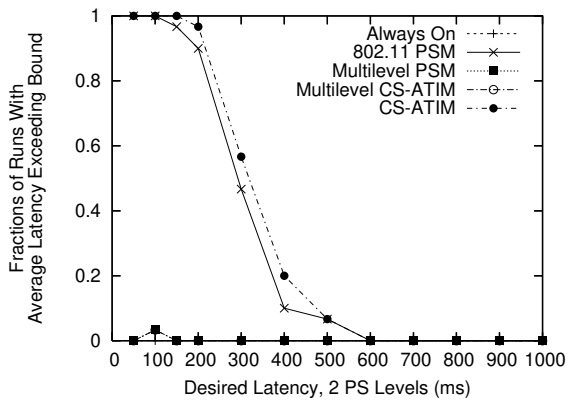

Fig. 10. Latency threshold versus observed latency using two power save levels.

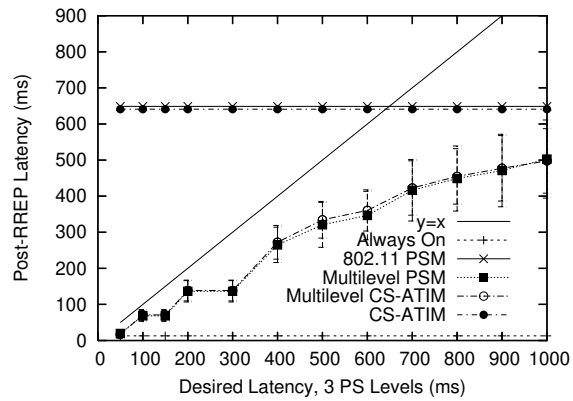

Fig. 12. Latency threshold versus observed latency using three power save levels.

multilevel power save protocols are also much more useful in achieving the latency bound. In Figure 12, we can see that an application with L up to about 600 ms cannot achieve its bound without the use of multilevel power save protocols or turning off power save all together. The 600 ms is a function of the average hop count in the network and beacon interval size. From Figure 9 and Figure 12, we can infer that the average hop count is approximately three in our scenarios since the latency with a 100 ms beacon interval is about 300 ms and with a 200 ms beacon interval is about 600 ms. As with the $k = 2$ case, we can see in Figure 13 that virtually *none* of the individual runs exceed the latency bound when using the multilevel extension. As discussed earlier, the bound is occasionally exceeded since our protocol only accounts for the power save induced latency whereas the observed value is also affected by the packet transmission time and queuing delay.

V. Extensions

A. Energy Load Balancing

As in previous work [23], it is still a concern that certain nodes that are chosen to have a high energy power save state early may end up receiving a disproportionate amount of the network's traffic because they have a favorable metric. To address this, we propose that higher energy nodes periodically try to "patch" their place on the route with another node with a power level less than or equal to it that can be reached by both its upstream and downstream neighbors on the route. A node could try this procedure when its residual energy falls below a specified level or when its recent energy consumption *rate* exceeds a certain level.

Such a situation may occur when two nodes, say A and B, are equivalent from a routing perspective and are in the same power save state when the $RREQ$ is initially broadcast. In this circumstance, node A may be selected, for example, because it wins access to the channel before B and rebroadcasts the $RREQ$ first. Thus, patching would allow A to eventually switch places with B to balance the energy consumption of the two nodes.

We note that others [2] propose delaying the $RREQ$ proportional to remaining energy. However, a node with more energy at the time of the $RREQ$ may eventually consume more energy than its neighbors and require load balancing. Also, such a scheme assumes a homogeneous environment where all devices have the same initial energy and/or they all consume energy at the same rate. In practice, this may not be

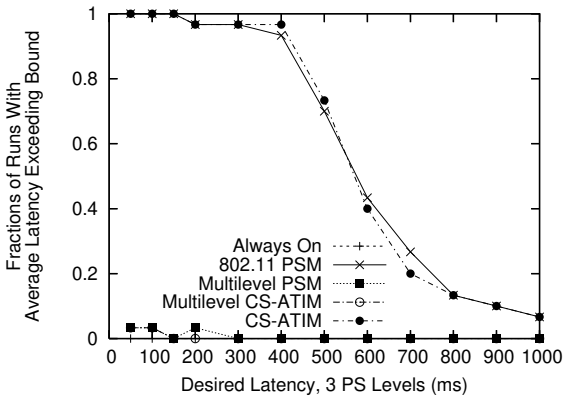

Fig. 13. Latency threshold versus observed latency using three power save levels.

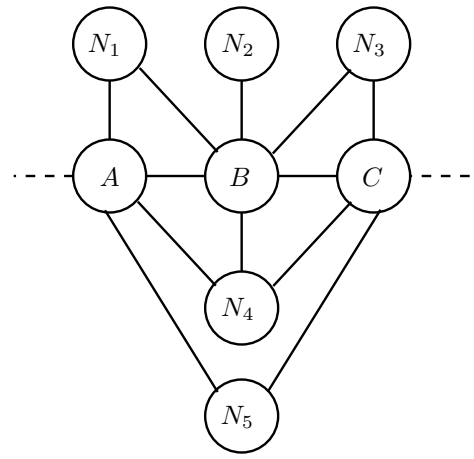

Fig. 14. Patching a route in multilevel power save.

true.

To do this, the node desiring the patch, say P, broadcasts a message that is received by both its upstream and downstream neighbors (nbr_{up} and nbr_{down}, respectively) asking them each to broadcast a packet to test which nodes are neighbors to both nbr_{up} and nbr_{down}. This packet also includes P's residual energy. Any node that receives both the packet broadcast by nbr_{up} and nbr_{down} and has more residual energy that P is a candidate to replace P on the path. Such nodes respond to P and then P can select the node with the highest remaining residual energy. Standard techniques such as choosing a back-off interval proportional to a node's residual energy can be used to ensure that nodes with a higher residual energy reply first.

The process of patching a route is shown in Figure 14. Here, we assume that traffic is being sent along the route $A \rightarrow B \rightarrow C$ and that B wants to try to remove itself from the path. Thus, B sends out a broadcast indicating that it wants to try to patch the route between A and C. In turn, A and C broadcast a packet to help other nodes determine their reachability. In this example, N_1, N_2, and N_3 cannot take B's place because they do not have both A *and* C as neighbors. The only two candidates to take B's place are N_4 and N_5, since both are neighbors of both A and C. In order for N_5 to take B's place, it would be necessary for it to communicate this to B via A and/or C. This is in contrast to N_4, which can communicate with B directly. This implies that the communication overhead and complexity for N_4 to be used is less than if N_5 is used. In order for N_4 or N_5 to take B's place on the route, they need to have more residual energy than B.

If a node is part of multiple, disjoint routes, it can still try this patch procedure incrementally by applying it to the path which requires the highest energy level until an acceptable level is reached. We note that in this scenario, a node may also need to account for the rate at which traffic is being forwarded on a given path since flows which require a lower energy power save level, say f_{low}, may still cause the node to consume more energy than a flow that requires a higher power save level, say f_{high}, if the f_{low} is sending at a higher rate that f_{high}. Another issue is instability in the route if two neighbors try to patch their place on the route simultaneously. If a node hears a patch request from one of its neighbors, it defers from issuing a patch request until the current one is resolved or a timeout occurs.

We have not evaluated this protocol extension. Adding it to the protocol and testing it via simulation and/or implementation is an area of future work.

VI. Conclusion

Motivated by the need for power save protocols (for reasons discussed in Section I), we have proposed a link layer technique and routing protocol that adapts to an application-defined latency in an energy efficient manner. Like previous work [1]–[4], we propose placing nodes in different power save states that tradeoff energy consumption and latency. The contribution of our work is that we design protocols to handle k levels of power save states whereas previous work only focused on the $k = 1$ and $k = 2$ cases. Our adaptive sleeping technique allows nodes to adjust their sleeping interval in response to the desired latency of data that it is forwarding.

We evaluate our protocols via simulation and find that they allow end-to-end latency bounds to be achieved with much less energy consumption than turning power save off. Also, traditional power save protocols (i.e., $k = 2$) are unable to achieve the latency bound in many cases despite consuming only slightly less energy than our multilevel protocol. Thus, our technique can maintain a desired latency bound with only a small increase in energy consumption over traditional power save protocols and with far less energy consumption than turning power save off.

References

[1] R. Zheng and R. Kravets, "On-demand Power Management for Ad Hoc Networks," in *IEEE Infocom 2003*, April 2003.

[2] C. Sengul and R. Kravets, "TITAN: On-Demand Topology Management in Ad Hoc Networks," *ACM Mobile Computing and Communications Review (MC2R)*, vol. 9, no. 1, pp. 77–82, January 2005.

[3] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks," in *ACM MobiCom 2001*, July 2001.

[4] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed Energy Conservation for Ad Hoc Routing," in *ACM MobiCom 2001*, July 2001.
[5] T. Starner, "Thick Clients for Personal Wireless Devices," *IEEE Computer*, vol. 35, no. 1, pp. 133–135, January 2002.
[6] D. G. Sachs, W. Yuan, C. J. Hughes, A. Harris, S. V. Adve, D. L. Jones, R. H. Kravets, and K. Nahrstedt, "GRACE: A Hierarchical Adaptation Framework for Saving Energy," University of Illinois at Urbana-Champaign, Tech. Rep. UIUCDCS-R-2004-2409, February 2004.
[7] N. Jain, Vodafone Symposium Presentation at the University of Illinois at Urbana-Champaign, April 8–10, 2005, the author was at Qualcomm Research at the time of the presentation.
[8] C. F. Chiasserini and R. R. Rao, "Combining Paging with Dynamic Power Management," in *IEEE Infocom 2001*, April 2001.
[9] IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.
[10] N. H. Vaidya, "A Case for Two-Level Distributed Recovery Schemes," in *ACM SIGMETRICS 1995*, May 1995.
[11] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous Wakeup for Ad Hoc Networks," in *ACM MobiHoc 2003*, June 2003.
[12] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, "Optimizing Sensor Networks in the Energy-Latency-Density Design Space," *IEEE Transactions on Mobile Computing*, vol. 1, no. 1, pp. 70–80, January–March 2002.
[13] A. El-Hoiydi and J.-D. Decotignie, "WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks," in *Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS) 2004*, July 2004.
[14] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*, T. Imielinski and H. Korth, Eds. Kluwer Academic Publishers, 1996, ch. 5, pp. 153–181.
[15] R. Draves, J. Padhye, and B. Zill, "Comparison of Routing Metrics for Static Multi-Hop Wireless Networks," in *ACM SIGCOMM 2004*, August–September 2004.
[16] M. J. McGlynn and S. A. Borbash, "Birthday Protocols for Low Energy Deployment and Flexible Neighbor Discovery in Ad Hoc Wireless Networks," in *ACM MobiHoc 2001*, October 2001.
[17] C. E. Perkins and E. M. Royer, "Ad-Hoc On Demand Distance Vector Routing," in *IEEE WMCSA 1999*, February 1999.
[18] Link-state routing protocol, `http://en.wikipedia.org/wiki/Link-state_routing_protocol`.
[19] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol," in *IEEE International Multi Topic Conference (INMIC) 2001*, December 2001.
[20] D. Eppstein, "Finding the k Shortest Paths," *SIAM Journal on Computing*, vol. 28, no. 2, pp. 652–673, 1998.
[21] `ns-2` – The Network Simulator, `http://www.isi.edu/nsnam/ns`.
[22] M. J. Miller and N. H. Vaidya, "Improving Power Save Protocols Using Carrier Sensing for Dynamic Advertisement Windows," in *IEEE MASS 2005*, November 2005.
[23] J.-H. Chang and L. Tassiulas, "Energy Conserving Routing in Wireless Ad-hoc Networks," in *IEEE Infocom 2000*, March 2000.
[24] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*. Elsevier Science Publishers, 1992.
[25] P. Crescenzi and V. Kann, "A compendium of NP optimization problems," `http://www.nada.kth.se/~viggo/problemlist/compendium.html`.

APPENDIX I
MINIMUM ENERGY ROUTING PROOF

Sections I-A, I-B, and I-C give instances of known NP-complete problems [24], [25]. We use these for the reduction in our proof in Section I-D.

A. Steiner Tree Problem (ST)

INSTANCE: An undirected graph $G = (V, E)$, an edge cost function $c : E \to N$, a subset $S \subseteq V$ of required vertices.

SOLUTION: A subtree of G that includes all the vertices in S. This is called a Steiner tree. Note that vertices in $V \setminus S$ may be included in the Steiner Tree and are called *Steiner vertices*.

MEASURE: Sum of the edge weights in the subtree.

B. Steiner Tree With Unit Edge Weights Problem (ST-UE)

A proof to show that *ST* is NP-complete is based on a reduction from the exact covering by 3-sets problem [24]. The *ST* proof [24] answers the following decision problem: given a bipartite graph $G = (V, E)$ (the bipartite property is a sufficient condition for being an undirected graph), a subset of vertices $S \subseteq V$, and an integer B, is there a tree T in G that spans all of the S terminals and has at most B edges?

By design, the *ST* proof [24] shows that *ST* is NP-complete even if the cost function is $c : E \to 1$. Thus, we know that even though *ST-UE* (defined below) is a limited case of *ST*, it is still NP-complete.

INSTANCE: An undirected graph $G = (V, E)$, an edge cost function $c : E \to 1$, a subset $S \subseteq V$ of required vertices.

SOLUTION: A subtree of G that includes all the vertices in S.

MEASURE: Sum of the edge weights in the subtree.

We can see that the measure in *ST-UE* is equivalent to the following measure:

MEASURE 2: The number of edges in the subtree.

Trivially, minimizing the number of edges in a subtree also minimizes the number of vertices in the subtree since $V_T = E_T + 1$.

C. Steiner Tree on Bidirected Graphs (ST-BG)

Any instance of *ST* (which, of course, includes *ST-UE*) can be reduced to *ST-BG* by replacing every undirected edge $e_{ij} \in E$ with two directed edges e_{ij} and e_{ji}[5] and giving both of the directed edges the same cost as the original undirected edge. Then, any one node in S, which we denote r, is chosen as the root.[6]

Thus, the *ST-BG* problem (also called the *Steiner arborescence problem* [24]) is defined as follows.[7]

INSTANCE: A bidirected graph $G = (V, E)$, an edge cost function $c : E \to 1$, a subset $S \subseteq V$ of required vertices, and a root vertex, r.

SOLUTION: A directed subtree of G such that there exists a path from r to every vertex in S.

MEASURE: Sum of the edge weights in the subtree.

The corresponding decision problem is: given an instance of *ST-BG*, is there a solution such that the sum of the edge weights is less than W?

[5]The notation e_{ij} denotes an edge between i and j in the undirected case and a directed edge from i to j in the directed case.

[6]In the undirected case, declaring a root is unnecessary since every node can reach every other node in the tree. In the directed case, we specify a root to create a structure which ensures that the root can reach every other node in the tree.

[7]We skip the general definition of *ST-BG*, where $c : E \to N$, and just focus on the version with unit edge weights.

D. Minimum Energy Routing for Multilevel Power Save (MER)

We now define the MER problem and show that it is NP-complete using a reduction from *ST-BG*. As described in Section III, we only consider the latency induced by the power saving protocol because this delay tends to be larger relative to contention and queuing delay in the networks that we consider. Thus, the l_i term mentioned below is only a function of a node's power save state and *not* a function of the number of flows that it and its neighbors are forwarding.

INSTANCE: A bidirected graph $G = (V, E)$, a set of flows F (i.e., a set of source-destination tuples), a maximum end-to-end latency threshold for a path L, and k the number of power save states available to each node. Each power save state has an associated latency, l_i, and energy consumption, g_i (where $1 \leq i \leq k$). For $i < j$, $l_i \leq l_j$ and $g_i \geq g_j$. When a node is in PS state i, its energy consumption is g_i and the latency cost of all its *incoming* edges is l_i.

SOLUTION: A set of power save states for each node such that each flow in F can be routed without L being violated for *any* of the flows.

MEASURE: Sum of the energy consumed by the power save state (i.e., g_i) of each node in the network.

The decision problem that we use for *MER* is: can we assign power save states for an instance of *MER* such that the sum of the energy consumed by the power save state of each node is less than Y?

It is easy to verify that *MER* is in NP. Given a set of PS states for each node, all of the link costs in the network can be fixed (i.e., the appropriate value of l_i for all incoming links to a node). Then, we do shortest path routing on the weighted graph obtained by using latencies as edge weights for each flow in F and verify that the cost of each path is less than L, which can be done in polynomial time. Additionally, we verify that the sum of all the power save states is less than Y which can be done in polynomial time.

For convenience, we consider a special case of *MER* where:

- $k = 2$
- $g_1 = 1$ and $g_2 = 0$
- $l_1 = 1$ and $l_2 = |V|$
- $L = |V| - 1$
- All flows originate from one sender
- A flow *is capable* of satisfying the latency constraint. This can be checked in polynomial time by placing all nodes in their highest energy state and computing a flow's shortest path cost. If this cost is greater than L, then we can immediately decide that the instance of *MER* is unsolvable.

By showing that the above special case of *MER* is NP-complete, we will have proved the general *MER* problem to be NP-complete. We do so with a reduction from *ST-BG*. We show that given any instance of the *ST-BG* problem, it is possible to construct an instance of the *MER* problem such that the instance of *ST-BG* has a total edge weight less than W if and only if the *MER* instance has a total energy consumption less than $W + 1$.

Given an instance of *ST-BG*, we convert it to an instance of *MER* as follows. The graph, G, from *ST-BG* is used as the graph in *MER*. The root, r, from *ST-BG* is the one sender in our special case of *MER* and each vertex in *ST-BG*'s S set corresponds to a receiver in *MER*.

Now, we need to show that an instance of *ST-BG* has a total edge weight less than W if and only if the corresponding *MER* instance has a total energy consumption less than $W + 1$.

- **If *ST-BG* Has Total Edge Weight $< W$:** Then, we select all of the nodes in *ST-BG*'s subtree to remain in PS state 1 while all other nodes are put in PS state 2. Since the cost of each edge in the tree is 1 and there can be at most $|V| - 1$ edges in the tree, then the latency must be less than or equal to L. This is because each of the selected nodes has an incoming latency of $l_1 = 1$ and there can be at most $|V| - 1$ edges in the tree since there are $|V|$ nodes total. Thus, the total latency is at most $L = |V| - 1$. Since there must be at most $W - 1$ edges in the subtree, there can be at most W nodes in PS state 1 and, thus, the sum of the energy consumption in the network is less than $W + 1$.
- **If *MER* Has Total Energy Consumption $< W + 1$ and the Latency $\leq L = |V| - 1$:** Then, all the nodes on every routing path must be in PS state 1 or else the latency would be greater than L (since one node in PS state 2 would make the latency at least $|V| > L$). Thus, each node on the routing paths is using one unit of energy. Therefore, if the total energy consumption is less than $W + 1$, then at most W nodes in the network are using one unit of energy and the source can reach all receivers. Since the source, each receiver, and all intermediate nodes on the paths form a tree with at most W nodes, we have a subtree with at most $W - 1$ edges.

■